O‘ZBEKISTON RESPUBLIKASI OLIY VA O‘RTA MAXSUS TA’LIM VAZIRLIGI

SAMARQAND DAVLAT UNIVERSITETI

O. R. Yusupov, I. Q. Ximmatov, E. Sh. Eshonqulov

ALGORITMLAR VA MA’LUMOTLAR STRUKTURALARI

5330100 – Kompyuter ilmlari va dasturlash texnologiyalari va 5130200 – Amaliy matematika yo‘nalishi talabalari uchun

O‘QUV QO‘LLANMA

SAMARQAND – 2021

UO‘K: 681.142
KBK: 22.12
Y 91

O. R. Yusupov, I. Q. Ximmatov, E. Sh. Eshonqulov. Algoritmlar va berilganlar strukturalari. Oliy o‘quv yurtlari uchun o‘quv qo‘llanma. – Samarqand: SamDU nashri. 2021-yil, 205 bet.

O‘quv qo‘llanmada algoritmlar va ma'lumotlar tuzilmalari rivojlanishiga asos bo'lgan asosiy nazariy tushunchalarni shakllantirish, ma'lumotlarning abstrakt turi modeli (paradigmasi) yordamida murakkab (dinamik) ma'lumotlar tuzilmalarini qurish va ulardan foydalanish: spetsifikatsiya → taqdimot → amalga oshirish, asosiy sinflar haqida tushuncha va bilimlarni shakllantirish, algoritmlar (ma'lumotlarni qidirish, kodlash, tezkor qidirish, saralash), ularda ishlatiladigan ma'lumotlar tuzilmalari(graflar, daraxtlar) va ularga asoslangan masalalarni yechishning umumiy sxemalari, tanlangan tilda (C#, C/C++, Java, Python) odatiy algoritmlar va ma'lumotlar tuzilmalarini va ularning modifikatsiyasini amalga oshirishga o'rgatish, algoritmlar va dasturlarning murakkabligini tahlil qilish to'g‘risida g‘oyalar va bilimlarni shakllantirishdan iborat.

Kompyuter texnologiyalari va informatika sohasidagi barcha yo‘nalish va mutaxassislik talabalari ham foydalanishlari mumkin.

UO‘K: 681.142
KBK: 22.12

Ma’sul muharrir:

E. Urunbayev – Samarqand davlat universiteti “Matematik modellashtirish” kafedrasi, t.f.n., dotsent

Taqrizchilar:

I. Xo‘jayorov – Toshkent axborot texnologiyalari universiteti Samarqand filiali “Axborot texnologiyalari” kafedrasi, PhD

I. N. Bozorov – Samarqand davlat universiteti “Optimal boshqaruv usullari” kafedrasi, f.-m.f.n., dotsent

ISBN 978-9943

MUNDARIJA

Kirish

“Algoritmlar va ma’lumotlar strukturalari” fanini o'zlashtirishning maqsadi dasturlashda ishlatiladigan ma'lumotlar tuzilmalarini, ularning spetsifikatsiyasi va amalga oshirilishini, ma'lumotlarni qayta ishlash algoritmlarini va ushbu algoritmlarni tahlil qilishni, algoritmlar va ma'lumotlar tuzilmalarining o'zaro bog‘liqligini o'rganishdir.

Algoritmlar va ma’lumotlar strukturalari fani har qanday dasturiy ta'minot tizimining asosidir: taqsimlangan tizimlar, mobil ilovalar, ma'lumotlar bazasi, veb-ilovalar. Ushbu kursda talaba kompyuter fanlari va dasturiy ta'minot muhandisligi sohasidagi barcha keyingi bilimlar uchun asos bo'lib xizmat qiladigan ma'lumotlar strukturalari va algoritmlarini o'zlashtiradi.

Fanni o‘qitishdan maqsad talabalarga algoritmlar va ma’lumotlar strukturalari fanini yetarli darajada o‘qitish, shu bilimlarga tayangan holda tanlab olingan tilda amaliy masalalarni yechish uchun kerak bo‘lgan algoritmlarni qo‘llashga o‘rgatish va ixtisoslik fanlarini o‘zlashtirishda tayanch bilimlarga ega bo‘lish.

Fanning vazifalariga algoritmlar va ma'lumotlar tuzilmalari rivojlanishiga asos bo'lgan asosiy nazariy tushunchalarni shakllantirish, ma'lumotlarning abstrakt turi (MAT) modeli (paradigmasi) yordamida murakkab (dinamik) ma'lumotlar tuzilmalarini qurish va ulardan foydalanish: spetsifikatsiya → taqdimot → amalga oshirish, asosiy sinflar haqida tushuncha va bilimlarni shakllantirish, algoritmlar (ma'lumotlarni qidirish, kodlash (siqish), tezkor qidirish, saralash), ularda ishlatiladigan ma'lumotlar tuzilmalari va ularga asoslangan masalalarni yechishning umumiy sxemalari, tanlangan tilda (C#, C/C++, Java, Python) odatiy algoritmlar va ma'lumotlar tuzilmalarini va ularning modifikatsiyasini amalga oshirishga o'rgatish, algoritmlar va dasturlarning murakkabligini tahlil qilish to'g‘risida g‘oyalar va bilimlarni shakllantirishdan iborat.

Algoritmlar nazariyasi bo‘yicha birinchi fundamental ishlar 1936-yilda paydo bo‘lgan. Tyuring mashinasi, Post va Chyorch tomonidan λ-

hisobi taklif etiladi. Ushbu mashinalar algoritmning formallashtirilgan rasmiylashtirilishi edi.

Algoritm tushunchasi aniq shaklda 20-asr boshlarida D. Gilbert, K. Gyodel, S. Klin, A. Chyorch, E. Post, A. Tyuring, N. Viner, A. A. Markov singari olimlarning asarlari tufayli shakllandi.

Ushbu qo‘llanmada yuqoridagi maqsad va vazifalarni bajarish uchun bir qator mavzular berilgan. Bu mavzularda berilgan ko‘plab algoritmlarni dasturiy ta’minotlarni ishlab chiqishda keng qo‘llash mumkin.

O‘quv qo‘llanmada berilgan mavzular talabalar bilimini oshirish uchun xizmat qiladi deb hisoblaymiz.

1-§. Kirish. Hisoblash modellari, algoritmlar va ularning murakkabligi

Algoritm tushunchasi. Avvalo **algoritm** tushunchasi IX asrda yashab ijod etgan buyuk bobokalonimiz Muhammad al-Xorazmiy nomi bilan uzviy bog‘liqligini tushuntirish lozim. **Algoritm** so‘zi al-Xorazmiyning arifmetikaga bag‘ishlangan asarining dastlabki betidagi **“Dixit Algoritmi” (“dediki al-Xorazmiy”** ning lotincha ifodasi) degan jumlalardan kelib chiqqan. Shundan so‘ng al-Xorazmiyning sanoq sistemasini takomillashtirishga qo‘shgan hissasi, uning asarlari algoritm tushunchasining kiritilishiga sabab bo‘lganligi ta’kidlab o‘tiladi.

Algoritm nima degan savolga, u asosiy tushuncha sifatida qabul qilinganligidan, uning faqat tavsifi beriladi, ya’ni biror maqsadga erishishga yoki qandaydir masalani yechishga qaratilgan ko‘rsatmalarning (buyruqlarning) aniq, tushunarli, chekli hamda to‘liq tizimi tushuniladi.

Algoritm tushunchasi aniq shaklda 20-asr boshlarida D. Gilbert, K. Gyodel, S. Klin, A. Chyorch, E. Post, A. Tyuring, N. Viner, A. A. Markov singari olimlarning asarlari tufayli shakllandi.

Eng qadimiy raqamli algoritmlardan biri Yevklid algoritmi (miloddan avvalgi III asr) deb hisoblanadi - ikki sonning eng katta umumiy bo‘luvchisini topish. Algoritmlarning zamonaviy nazariyasi nemis matematikasi Kurt Gyodel (1931) asarlari bilan boshlandi, ular o‘zlarining rasmiy, izchil aksiomalar tizimi doirasida yechib bo‘lmaydigan muammolar mavjudligini ko‘rsatdi.

Algoritmlar nazariyasi bo‘yicha birinchi fundamental ishlar 1936-yilda paydo bo‘lgan. Tyuring mashinasi, Post va Chyorch tomonidan λ-hisobi taklif etiladi. Ushbu mashinalar algoritmning formallashtirilgan rasmiylashtirilishi edi.

Algoritmni bajarayotgan kishi – ijrochi, asosiy algoritmni aniqlashtiruvchi algoritm – **yordamchi algoritm** ekanligini ham ta’kidlab o‘tish joiz. Umuman, algoritmning qanday maqsadga mo‘ljallanganligidan qat’i nazar uni muvaffaqiyat bilan bajarish mumkinligini aytib o‘tish lozimdir.

Algoritmning bir nechta ta’rifi mavjud. Ulardan ayrimlarini keltirib o‘tamiz:

– “**Algoritm** - bu belgilaydigan cheklangan qoidalar to‘plami, muayyan vazifalar to‘plamini hal qilish bo‘yicha amallar ketma-ketligi va beshta muhim xossaga ega: aniqlik, tushunarlilik, kiritish, chiqarish, samaradorlik”. (D. E. Knut).

– “**Algoritm** - bu qat’iy belgilangan qoidalar asosida bajariladigan har qanday hisoblash tizimidir, bu ma’lum bir qator bosqichlardan so‘ng, aniq qo‘yilgan masalani hal qilishga olib keladi" (A. Kolmogorov).

– “**Algoritm** - bu har xil boshlang‘ich ma’lumotlardan kerakli natijaga o‘tadigan hisoblash jarayonini belgilaydigan aniq ketma-ketlik" (A. Markov).

1950-yillarda algoritm nazariyasiga o‘z hissalarini Kolmogorov va Markov asarlari qo‘shgan. 1960-1970 yillarda algoritm nazariyasida quyidagi tadqiqot yo‘nalishlari shakllandi:

1) Algoritmlarning klassik nazariyasi (rasmiy tillar nuqtai nazaridan masalalarni shakllantirish, yechuvchanlik muammosi tushunchasi, murakkablik sinflarini kiritish, P = NP (?) masalasini shakllantirish, NP-ning to‘liq masalalarini sinfi va uni o‘rganish);

2) Algoritmlarni asimptotik tahlil qilish nazariyasi (algoritmning murakkabligi tushunchasi, algoritmlarni baholash kriteriyalari, asimptotik baholarni olish usullari, xususan, rekursiv algoritmlar uchun, murakkablikni yoki bajarilish vaqtini asimptotik tahlil qilish);

3) Hisoblash algoritmlarini amaliy tahlil qilish nazariyasi (funksiyalarning intervalli tahlili, algoritmlar sifatining amaliy mezonlari, ratsional algoritmlarni tanlash metodikasi).

Algoritmlar nazariyasida hal qilingan maqsad va vazifalar:

- "algoritm" tushunchasini formallashtirish (rasmiylashtirish) va formal (rasmiy) algoritmik tizimlarni o‘rganish;

- muammolarning algoritmik yechimini rasmiy tasdiqlash;

- vazifalarni tasniflash, murakkablik sinflarini aniqlash va tadqiq qilish;

- algoritmlarning murakkabligini asimptotik tahlil qilish;

- rekursiv algoritmlarni o‘rganish va tahlil qilish;
- algoritmlar sifatini qiyosiy baholash mezonlarini ishlab chiqish.

1.1. Algoritm tushunchasini formallashtirish

1-ta’rif. Algoritm - bu ma’lum bir tilda berilgan, mumkin bo‘lgan dastlabki ma’lumotlar sinfi uchun masalani hal qilish uchun mumkin bo‘lgan elementar amallarning cheklangan ketma-ketligi.

Masalaning dastlabki ma’lumotlarining to‘plami **D** bo‘lsin va **R** - mumkin bo‘lgan natijalar to‘plami, shunda algoritm **D** → **R** ko‘rinishida tasvirlanadi. Bu tasvirlanish to‘liq bo‘lmasligi mumkin.

Agar natija faqat ba’zi $d \in D$ uchun olingan bo‘lsa, algoritm **qismiy** algoritm va agar barcha $d \in D$ uchun to‘g‘ri natija olsa **to‘liq** algoritm deyiladi.

2-ta’rif. Algoritm - bu cheklangan vaqt ichida masalani yechish natijasiga erishish uchun ijrochining harakatlari tartibini tavsiflovchi aniq ko‘rsatmalar to‘plami.

Algoritmning aniq yoki bilvosita turli xil ta’riflari bir qator talablarni keltirib chiqaradi:

- algoritmda cheklangan miqdordagi elementar bajarilishi mumkin bo‘lgan ketma-ketlik bo‘lishi kerak, ya’ni **yozuvning aniqligi** talabi bajarilishi kerak;
- algoritm masalani yechishda cheklangan sonli bosqichlarni bajarishi kerak, ya’ni **harakatlarning aniqligi** talabi bajarilishi kerak;
- barcha qabul qilingan kirish ma’lumotlari uchun algoritm bir xil bo‘lishi kerak, ya’ni **universallik** talabiga javob berish;
- algoritm qo‘yilgan vazifaga nisbatan to‘g‘ri yechimga olib kelishi kerak, ya’ni to‘g‘rilik talabi bajarilishi kerak.

Algoritmning asosiy xossalari haqida quyidagilarni ta’kidlash mumkin:

1-xossa. **Diskretlilik,** ya’ni algoritmni chekli sondagi oddiy ko‘rsatmalar ketma-ketligi shaklida ifodalash mumkin.

2-xossa. Tushunarlilik, ya’ni ijrochiga tavsiya etilayotgan ko‘rsatmalar uning uchun tushunarli bo‘lishi shart, aks holda ijrochi

oddiy amalni ham bajara olmay qolishi mumkin. Har bir ijrochining bajara olishi mumkin bo‘lgan ko‘rsatmalar tizimi mavjud.

3-xossa. Aniqlik, ya’ni ijrochiga berilayotgan ko‘rsatmalar aniq mazmunda bo‘lishi lozim hamda faqat algoritmda ko‘rsatilgan tartibda bajarilishi shart.

4-xossa. **Ommaviylik,** ya’ni har bir algoritm mazmuniga ko‘ra bir turdagi masalalarning barchasi uchun yaroqli bo‘lishi lozim. **Masalan,** ikki oddiy kasr umumiy maxrajini topish algoritmi har qanday kasrlar umumiy maxrajini topish uchun ishlatiladi.

5-xossa. Natijaviylik, ya’ni har bir algoritm chekli sondagi qadamlardan so‘ng albatta natija berishi lozim.

Bu xossalar mohiyatini o‘rganish va konkret algoritmlar uchun qarab chiqish talabalarning xossalar mazmunini bilib olishlariga yordam beradi.

Algoritmning tasvirlash usullari haqida gapirganda algoritmning berilish usullari xilma-xilligi va ular orasida eng ko‘p uchraydiganlari quyidagilar ekanligini ko‘rsatib o‘tish joiz:

1. Algoritmning so‘zlar orqali ifodalanishi.

2. Algoritmning formulalar yordamida berilishi.

3. Algoritmning jadval ko‘rinishida berilishi, masalan, turli matematik jadvallar, lotereya yutuqlari jadvali, funksiyalar qiymatlari jadvallari bunga misol bo‘ladi.

4. Algoritmning dastur shaklida ifodalanishi, ya’ni algoritm kompyuter ijrochisiga tushunarli bo‘lgan dastur shaklida beriladi**.**

5. Algoritmning algoritmik tilda tasvirlanishi, ya’ni algoritm bir xil va aniq ifodalash, bajarish uchun qo‘llanadigan belgilash va qoidalar majmui algoritmik til orqali ifodalashdir. Ulardan o‘quv o‘rganish tili sifatida foydalanilmoqda. Bo‘lardan E-praktikum yoki E-tili algoritm ijrochisi algoritmik tili ham mavjud.

6. Algoritmlarning grafik shaklda tasvirlanishi. **Masalan**, grafiklar, sxemalar ya’ni blok - sxema bunga misol bo‘la oladi. Blok sxemaning asosiy elementlari quyidagilar: **oval (ellips shakli)**-algoritm boshlanishi va tugallanishi, **to‘g‘ri burchakli to‘rtburchak**-qiymat berish yoki tegishli ko‘rsatmalarni bajarish. **Romb - shart** tekshirishni

belgilaydi. Uning yo‘naltiruvchilari tarmoqlar bo‘yicha biri ha ikkinchisi yo‘q yo‘nalishlarni beradi, **parallelogramm**- ma’lumotlarni kiritish yoki chiqarish, **yordamchi algoritmga murojaat** - parallelogramm ikki tomoni chiziq**, yo‘naltiruvchi chiziq** - blok-sxemadagi harakat boshqaruvi, **nuqta-to‘g‘ri chiziq (ikkita parallel)** - qiymat berish.

Algoritmda bajarilishi tugallangan amallar ketma-ketligi **algoritm qadami** deb yuritiladi. Har bir alohida qadamni ijro etish uchun bajarilishi kerak bo‘lgan amallar haqidagi ko‘rsatma buyruq deb aytiladi.

Algoritmlarni ko‘rgazmaliroq qilib tasvirlash uchun blok-sxema, ya’ni geometrik usul ko‘proq qo‘llaniladi. Algoritmning blok-sxemasi algoritmning asosiy tuzilishining yaqqol geometrik tasviri: algoritm bloklari, ya’ni geometrik shakllar ko‘rinishida, bloklar orasidagi aloqa esa yo‘naltirilgan chiziqlar bilan ko‘rsatiladi. Chiziqlarning yo‘nalishi bir blokdan so‘ng qaysi blok bajarilishini bildiradi. Algoritmlarni ushbu usulda ifodalashda vazifasi, tutgan o‘rniga qarab quyidagi **geometrik shakl(blok)** lardan foydalaniladi.

Algoritmlar berilishi va ifodalanishiga qarab: **chiziqli, tarmoqlanuvchi va takrorlanuvchi** turlarga bo‘linadi.

Algoritmning turlari bilan tanishtirganda, avvalo hech qanday shart tekshirilmaydigan va tartib bilan faqat ketma-ket bajariladigan jarayonlarni ifodalaydigan **chiziqli algoritmlar** aytib o‘tiladi.

Algoritm xato natijalarni keltirib chiqaradigan yoki umuman natija bermaydigan bo‘lsa, xatolarni o‘z ichiga oladi.

Algoritm har qanday to‘g‘ri kirish uchun to‘g‘ri natijalarni beradigan bo‘lsa, xatosiz bo‘ladi.

Ma’lumotlarni kirish va chiqish turlari bo‘yicha farqlash mumkin

- Raqamli masalalarni yechish algoritmlari (birinchi bo‘lib paydo bo‘ldi);
- Raqamli bo‘lmagan algoritmlar.

1.2. Hisoblash modellari

Hisoblash nazariyasi va hisoblash murakkabligi nazariyasi hisoblash modelini nafaqat hisoblash uchun foydalaniladigan qabul qilinadigan

amallar to‘plamining ta’rifi, balki ularni qo‘llashning nisbiy xarajatlari sifatida ham ko‘rib chiqadi. Kerakli hisoblash manbalarini - ijro etish vaqtini, xotira hajmini, shuningdek algoritmlarning cheklanishlarini yoki kompyuterni xarakterlash mumkin - faqat ma’lum bir hisoblash modeli tanlangan taqdirda.

Modelga asoslangan muhandislikda hisoblash modeli va uning tanlovi, agar uning alohida qismlarining xatti-harakatlari ma’lum bo‘lsa, umuman tizim qanday ishlaydi degan savolga javob beradi.

Hisoblash murakkabligining asimptotik bahosida hisoblash modeli ma’lum narx bilan qabul qilinadigan primitiv amallar orqali aniqlanadi.

Ma’lum amallar to‘plamiga va ularning hisoblash murakkabligiga qarab bir qator hisoblash modellari ma’lum. Ular quyidagi keng toifalarga bo‘linadi: algoritm hisoblashning murakkabligini yuqori chegarasini olish uchun foydalaniladigan abstrakt mashinalar va algoritmik masalalar uchun hisoblash murakkabligining pastki chegarasini olish uchun ishlatiladigan qaror modellari.

Tyuring tezisi. Chyorch tezisi. Algoritm tushunchasini aniqlashga yondashishlar. Algoritm tushunchasini aniqlash bo‘yicha yondashishlarni uch asosiy yo‘nalishga bo‘lish mumkin.

Birinchi yo‘nalish effektiv hisoblanuvchi funksiya tushunchasini aniqlash bilan bog‘liq. Bu yo‘nalish bo‘yicha A.Chyorch, K.Gyodel, S.Klini[1] tadqiqot ishlarini olib borishgan.

1932-1935-yillar davomida A.Chyorch va S.Klini tomonidan o‘rganilgan va “λ-aniqlanuvchi funksiyalar” deb atalgan, to‘g‘ri aniqlangan hisoblanuvchi nazariy-sonli funksiyalar sinfining “λ-aniqlanuvchi funksiyalar” sinfi bizning intuitiv tasavvurimiz bo‘yicha hisoblanuvchi deb qaraladigan hamma funksiyalarni qamrab olishi mumkin degan fikr tug‘ilganligi 1935-yilda e’lon qilindi. Bu kutilmagan natija edi.

J.Erbranning[2] bir g‘oyasi asosida 1934-yilda K.Gyodel tomonidan aniqlangan va “umumrekursiv funksiyalar” deb atalgan boshqa

[1] Stiven Koul Klini (Stephen Cole Kleene, 1909-1994) – AQSh matematigi. U o‘z familiyasini “Kleyni” shaklda talaffuz etishiga qaramasdan, sobiq Sovet Ittifoqida uning ilmiy ishlarini rus tiliga (rus tilidan esa o‘zbek tiliga ham) “Klini” nomi bilan tarjima qilishgan.

[2] Jak Erbran (Jacques Herbrand, 1908-1931) – fransuz matematigi.

hisoblanuvchi funksiyalar sinfi ham "λ-aniqlanuvchi funksiyalar" xossalariga o‘xshash xossalarga ega edi.

1936-yilda A.Chyorch va S.Klini tomonidan bu ikkita sinf bir xil sinf ekanligi isbotlandi, ya’ni har qanday λ-aniqlanuvchi funksiya umumrekursiv funksiya bo‘lishi va har qanday umumrekursiv funksiya λ-aniqlanuvchi funksiya ekanligi tasdiqlandi.

1936 yilda Chyorch quyidagi tezisni e’lon qildi: har qanday intuitiv effektiv (samarali) hisoblanuvchi funksiyalar umumrekursiv funksiyalardir.

Bu teorema emas, balki tezisdir: tezis tarkibida intuitiv aniqlangan effektiv hisoblanuvchi funksiya tushunchasi, aniq matematik atamalarda aniqlangan umumrekursiv funksiya tushunchasi bilan aynan tenglashtirilgan. Shuning uchun bu tezisni isbotlash mumkin emas. Ammo Chyorch va boshqa olimlar tomonidan bu tezisni quvvatlovchi ko‘p dalillar ko‘rsatildi.

Ikkinchi yo‘nalish algoritm tushunchasini bevosita aniqlash bilan bog‘liq: 1936-1937-yillarda, A.Tyuring[3] Chyorch ishlaridan bexabar holda yangi funksiyalar sinfini kiritdi. Bu funksiyalarni “Tyuring bo‘yicha hisoblanuvchi funksiyalar” deb atadilar. Bu sinf ham yuqorida aytilgan xossalarga ega edi va buni **Tyuring tezisi** deb aytamiz. 1937-yilda A.Tyuring isbotladiki, uning hisoblanuvchi funksiyalari λ-aniqlanuvchi funksiyalarning o‘zi va, demak, umumrekursiv funksiyalarning xuddi o‘zi ekan. Shuning uchun Chyorch tezisi bilan Tyuring tezisi ekvivalentdir.

1936-yilda E. Post (Tyuring ishlaridan bexabar holda) aynan Tyuring erishgan natijalarga mos keladigan natijalarni e’lon qildi va 1943-yilda, 1920-1922-yillardagi nashr etilmagan ishlariga asoslanib, to‘rtinchi ekvivalent tezisni nashr etdi. Shunday qilib, algoritm tushunchasini bevosita aniqlashga va so‘ngra uning yordamida hisoblanuvchi funksiya tushunchasini aniqlashga birinchi bo‘lib bir-biridan bexabar holda E. Post va A. Tyuring erishdilar.

Post va Tyuring algoritmik jarayonlar ma’lum bir tuzilishga ega bo‘lgan “mashina” bajaradigan jarayonlar ekanligini ko‘rsatishdi. Ular

[3] Tyuring Alan Matison (Turing Alan Mathison, 1912-1954) – Ingliz matematigi, mantiqchisi, kriptografi.

ushbu “mashinalar” yordamida barcha hisoblanuvchi funksiyalar sinfi bilan barcha qismiy rekursiv funksiyalar sinfi bir xil ekanligini koʻrsatdilar va demak, Chyorch tezisining yana bitta fundamental tasdigʻi yuzaga keldi.

Uchinchi yoʻnalish – Rossiya matematigi A.Markov[4] tomonidan ishlab chiqilgan normal algoritmlar tushunchasi bilan bogʻliq.

1.3. Algoritmlarning murakkabligi

Algoritmlarning murakkabligi. Hisoblash muammolari cheklangan xotira resurslaridan foydalangan holda oqilona vaqt ichida yechilishi kerak. Bu algoritmning vaqt va fazoviy murakkabligi tushunchasiga olib keladi. Qoida tariqasida, algoritm turli vaqtlarni bajarishi mumkin boʻlgan turli xil amallarni oʻz ichiga oladi.

Algoritmlarni baholash uchun koʻpgina mezonlar mavjud. Odatda kirituvchi berilganlarni koʻpayishida masalani yechish uchun kerak boʻladigan **vaqt** va **xotira** hajmlarini oʻsish tartibini aniqlash muammosi qoʻyiladi. Har bir aniq masala bilan kiritiladigan berilganlarni miqdorini aniqlovchi qandaydir sonni bogʻlash zarur. Bunday son masalaning kattaligi deb qabul qilinadi. Masalan, ikkita matritsani koʻpaytirish masalasining oʻlchami boʻlib, matritsalar kattaligig xizmat qilishi mumkin. Graflar haqidagi masalada oʻlcham sifatida graf uchlarining soni boʻlishi mumkin.

Algoritm sarflanayotgan vaqt masalaning oʻlchami funksiyasi sifatida algoritmni **vaqt boʻyicha qiyinligi** deb nomlanadi. Bunday funksiyaga masalaning kattaligi oshganda limit ostidagi oʻzgarish **asimptotik** qiyinlik deb aytiladi.

Murakkablikni baholash. Algoritmlarning murakkabligi odatda bajarilish vaqti yoki ishlatilgan xotira boʻyicha baholanadi. Ikkala holatda ham, murakkablik kiritilgan ma’lumotlarning hajmiga bogʻliq: 100 ta elementdan iborat massivi xuddi shunga oʻxshash 1000 ta elementdan iborat massivga qaraganda tezroq qayta ishlanadi. Shu bilan birga, aniq vaqt bilan bir necha kishi qiziqadi: bu protsessorga bogʻliq,

[4] Bu yerda bir xil Markov Andrey Andreyevich (Марков Андрей Андреевич) ism-sharifga ega Rossiya matematiklari ota-bola A. A. Markovlarning kichigi (1903-1979) nazarda tutilgan. Ensiklopediyalarda A. A. Markovlarning kattasini (1856-1922) rus, kichigini esa sovet matematigi deb ham atashadi.

ma’lumotlar turi, dasturlash tili va boshqa ko‘plab parametrlarga ham bog‘liq. Faqatgina **asimptotik** murakkablik muhim, ya’ni kirish ma’lumotlarining kattaligi cheksizlikka intilayotgandagi murakkablik.

Masalan, ba’zi bir algoritmga kirish ma’lumotlarining n ta elementlarini qayta ishlash uchun $4n^3 + 7n$ ta shartli amallarni bajarish kerak. n ning ortishi bilan ishning umumiy davomiyligi n ning kubi uni 4 ga ko‘paytirgandan yoki 7n ni qo‘shgandan ko‘ra ko‘proq ta’sir qiladi. Ushbu algoritmning vaqt murakkabligi $O(n^3)$, ya’ni u kubik bilan kiritilgan ma’lumotlarning hajmiga bog‘liq bo‘ladi.

Bosh harf **O** dan foydalanish matematikadan kelib chiqadi, bu yerda ushbu belgi funksiyalarning asimptotik harakatlarini taqqoslash uchun ishlatiladi. Rasmiy ravishda **O(f(n))** algoritmning ishlash vaqti (yoki egallagan xotira miqdori), kiritilgan ma’lumotlarning hajmiga qarab, f(n) ga ko‘paytiriladigan ba’zi konstantalardan tezroq emasligini anglatadi.

O (n) - chiziqli murakkablik. Bunday murakkablik, masalan, saralanmagan massivdagi eng katta elementni topish algoritmiga ega. Qaysi biri maksimal ekanligini aniqlash uchun massivning barcha n elementlaridan o‘tishimiz kerak bo‘ladi.

O (log n) - logaritmik murakkablik. Eng oddiy misol - ikkilik qidirish. Agar massiv saralangan bo‘lsa, uni yarmiga bo‘lish orqali ma’lum bir qiymatga ega ekanligini tekshirishimiz mumkin. O‘rta elementni tekshirib ko‘ramiz, agar u kattaroq bo‘lsa, unda massivning ikkinchi yarmini tashlab yuboramiz. Agar kichikroq bo‘lsa, unda aksincha - biz dastlabki yarmini tashlaymiz va shu tarzda ikkiga bo‘linishni davom ettiramiz, natijada (logn) elementlarini tekshiramiz.

O (n^2) - kvadratik murakkablik. Bunday murakkablik, masalan, element qo‘shilishi natijasida yangi saralash algoritmiga ega. Kanonik dasturda bu ikkita ichki sikldan iborat: biri butun massivni bosib o‘tish, ikkinchisi esa allaqachon ajratilgan qismdan keyingi element uchun joy topish. Shunday qilib, amallar soni n*n, ya’ni n^2 kabi massiv o‘lchamiga bog‘liq bo‘ladi.

Murakkablikning boshqa ko‘rinishlari ham mavjud, ammo ularning barchasi bir xil prinsipga asoslanadi.

Algoritmning ishlash vaqti umuman kiritilgan ma’lumotlarning hajmiga bog‘liq emasligi ham sodir bo‘ladi. Bu holda murakkablik **O(1)** bilan belgilanadi. Masalan, massivning uchinchi elementi qiymatini aniqlash uchun elementlarni eslab qolishingiz yoki ular orqali bir necha bor o‘tishingiz shart emas. Siz har doim ma’lumotlarni kiritish oqimidagi uchinchi elementni kutishingiz kerak va bu esa siz uchun natija bo‘ladi, bu har qanday ma’lumot uchun hisoblash uchun bir xil vaqtni oladi.

Baholash muhim bo‘lgan taqdirda xotiradan xuddi shu tarzda amalga oshiriladi. Biroq, algoritmlar kirish ma’lumotlarining hajmi boshqalarga nisbatan kattalashganda sezilarli darajada ko‘proq xotiradan foydalanishi mumkin, ammo ular tezroq ishlaydi va aksincha. Bu hozirgi sharoit va talablar asosida muammolarni hal qilishning eng yaxshi usullarini tanlashga yordam beradi.

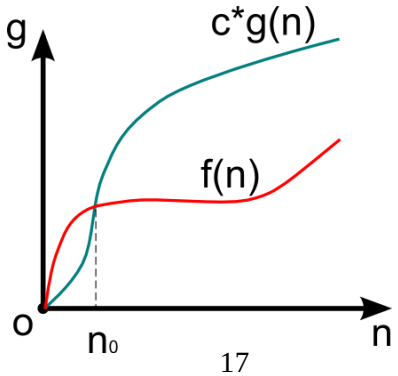
Algoritmlar murakkabligining o‘sish tartibi

Murakkablikning o‘sish tartibi (yoki aksiomatik murakkablik) katta kirish hajmi uchun algoritmning murakkablik funksiyasining taxminiy xatti-harakatini tavsiflaydi. Bundan kelib chiqadiki, vaqt murakkabligini baholashda elementar amallarni ko‘rib chiqishning hojati yo‘q, algoritm qadamlarini ko‘rib chiqish kifoya.

Algoritm qadami – bu ketma-ket joylashtirilgan elementar amallar to‘plami, uning bajarilish vaqti kirish qadamiga bog‘liq emas, ya’ni yuqoridan qandaydir doimiy bilan chegaralangan.

Asimptotik baholashning ko‘rinishlari. F(n)>0 murakkabligini, bir xil tartibdagi g(n)>0 funksiyasini, kirish n>0 o‘lchamini ko‘rib chiqaylik.

Agar f(n) = O (g(n)) va n> n_0 uchun c>0, n_0> 0 konstantalar mavjud bo‘lsa, u holda 0 <f(n)<c*g(n).



Bu holda g(n) funksiyasi f(n) uchun asimptotik-aniq baho hisoblanadi. Agar f(n) algoritmning murakkablik funksiyasi bo‘lsa, unda murakkablik tartibi f(n) uchun - O(g(n)) deb belgilanadi. Ushbu ibora doimiy koeffitsiyentgacha g(n) dan tez o‘smaydigan funksiyalar sinfini belgilaydi.

1-jadval.

Asimptotik funksiyalarga misollar

f(n)	g(n)
$2n^2 + 7n - 3$	n^2
$98n * \ln(n)$	$n * \ln(n)$
$5n + 2$	n
8	1

1.4. Algoritmlarning yomon, o‘rta, yaxshi holatlari tushunchalari

Algoritm murakkabligining o‘sish tartibi O(n) deb aytganda nimani nazarda tutamiz? Bu o‘rtacha? Yoki eng yomoni? Ehtimol, eng yaxshisi?

Agar eng yomon holat va o‘rtacha ko‘rsatkichlar bir-biridan farq qilmasa, odatda, eng yomon holat nazarda tutiladi. Masalan, biz o‘rtacha O(1) o‘sadigan, lekin vaqti-vaqti bilan O(n) ga aylanadigan algoritmlarning misollarini ko‘rib chiqamiz (masalan, massivga element qo‘shish). Bunday holda, algoritm o‘rtacha vaqt ichida doimiy ishlashini ko‘rsatamiz va murakkablik oshadigan holatlarni tushuntiramiz.

Algoritmlar va ma’lumotlar tuzilmalarining murakkabligini o‘lchashda odatda ikkita narsa haqida gaplashamiz: ishni bajarish uchun zarur bo‘lgan amallar soni (hisoblash murakkabligi) va algoritm zarur bo‘lgan resurslar, xususan, xotira (fazoviy murakkablik).

O‘n baravar tezroq ishlaydigan, ammo o‘n barobar ko‘proq joy ishlatadigan algoritm ko‘proq xotirali server mashinasi uchun yaxshi

boʻlishi mumkin. Ammo xotira hajmi chekli oʻrnatilgan tizimlarda ushbu algoritmdan foydalanib boʻlmaydi.

Odatda, quyidagi amallar hisobga olinadi:

1) taqqoslashlar ("katta", "kichik", "teng");
2) oʻzlashtirish (ta’minlash);
3) xotira ajratish.

Qaysi amalni hisoblash esa, odatda kontekstda aniqlanadi.

Masalan, ma’lumotlar tarkibidagi elementni topish algoritmini tavsiflashda biz deyarli taqqoslash amallarini nazarda tutamiz. Qidirish, avvalambor, oʻqish jarayonidir, shuning uchun ta’minlash yoki xotira ajratishda hech qanday ma’no yoʻq.

Tartiblash haqida gapirganda esa, taqqoslash, xotira ajratish va ta’minlash amallarini hisobga olishimiz mumkin. Bunday hollarda biz qaysi amallarni koʻrib chiqayotganimizni aniq koʻrsatib beramiz.

Algoritmlar nazariyasining asoslarini bilish har qanday dasturchi uchun juda muhimdir, chunki aynan shu fan algoritmlarning umumiy xususiyatlarini va ularni namoyish etishning rasmiy modellarini oʻrganadi. Hatto informatika darslaridan boshlab ham bizga jadvallarni tuzishni oʻrgatmoqdalar, bu keyinchalik maktabga qaraganda ancha murakkab masalalarni yozishda yordam beradi. Bundan tashqari, ma’lum bir muammoni hal qilishning deyarli har doim bir necha yoʻli borligi sir emas: ba’zilari koʻp vaqt, boshqa resurslarni sarflashni oʻz ichiga oladi, boshqalari esa faqat taxminiy yechim topishga yordam beradi.

Siz har doim topshiriqqa muvofiq ravishda eng maqbul narsani izlashingiz lozim, xususan, muammolar sinfini hal qilish algoritmlarini ishlab chiqishda.

Algoritmni har xil hajm va miqdorlarning boshlangʻich qiymatlari bilan qanday ishlashini, qanday manbalarga ehtiyoj borligini va yakuniy natijani chiqarish uchun qancha vaqt ketishini baholash ham muhimdir.

Koʻpincha, algoritm tahlili bir xil masalani yechish uchun ikki xil algoritmlarni taqqoslash yoki algoritmning amaliy qoʻllanilishini aniqlash uchun ishlatiladi. Algoritmlarni bajarish vaqti boʻyicha baholashga toʻxtalamiz. Algoritmlarni bajarish vaqtiga qarab

baholashning yondashuvlaridan biri bu algoritmni oddiygina kompyuterda ishga tushirish va uni u yoki bu tarzda vaqtga solishdir.

Ushbu yondashuvning koʻplab kamchiliklari mavjud. Birinchidan, bajarish vaqti algoritm ishlayotgan kompyuterga juda bogʻliq. Ikkinchidan, bunday taxmin kiritish ma’lumotlarining ma’lum bir oʻlchovi uchun faqat bitta qiymatni beradi. Agar bizda har xil oʻlchovlar boʻyicha taxminiy jadval mavjud boʻlsa ham, undan ish vaqtining kirish ma’lumotlarining oʻlchamiga funksional bogʻliqligini olish juda muammoli.

Shuning uchun algoritmni bajarilish vaqti boʻyicha baholash uchun kirish ma’lumotlarining oʻlchamlari boʻyicha bajarilgan elementar amallar sonining funksional bogʻliqligini topishga harakat qilishdir.

Algoritm murakkabligining asosiy koʻrsatkichi bu muammoni hal qilish uchun sarflanadigan **vaqt** va kerakli **xotira hajmi**.

Shuningdek, muammolar sinfi uchun murakkablikni tahlil qilishda ma’lum bir ma’lumot miqdori - **kirish kattaligi**ni tavsiflovchi ma’lum bir raqam aniqlanadi. Shunday qilib, biz algoritmning murakkabligi kirish oʻlchamining funksiyasi degan xulosaga kelishimiz mumkin.

Yomon, **oʻrtacha** yoki **eng yaxshi darajadagi** murakkablik tushunchalari mavjud. Odatda, eng yomon holatning murakkabligi baholanadi.

Eng yomon holatda **vaqt murakkabligi** - bu berilgan kattalikdagi masalani yechishda algoritm ishlashi davomida bajariladigan amallarning maksimal soniga teng boʻlgan kirish kattaligining funksiyasidir.

Eng yomon **sigʻimli murakkablik** - bu kirish hajmining ma’lum hajmdagi muammolarni yechishda erishilgan maksimal xotira yacheykalari soniga teng funksiyasi.

Algoritm murakkabligini baholash kriteriyalari. Bir xil me’yorda oʻlchash kriteriyasi algoritmning har bir bosqichi bir vaqt birligida, xotira yacheykasi esa hajmning bir birligida (konstanta boʻyicha aniqlikda) bajarilishini nazarda tutadi.

Logarifmik o‘lchash kriteriyasi ma’lum amal bilan qayta ishlanadigan operand o‘lchamini va xotira yacheykasida saqlanadigan qiymatni hisobga oladi.

$$l(i) = \begin{cases} [\log_2(|i|)] + 1, i \neq 0 \\ 1, i = 0 \end{cases}$$

Misol. Faktorialni hisoblashning murakkabligini baholash.

```
#include <iostream>
using namespace std;
int main()
{
  int n = 20;
  long long result=1;
  for (int i=2; i<=n; i++)
   result *=i;
  cout<<result;
   return 0;
}
```

Berilgan masalaning kirish kattaligi n ekanligini aniqlash oson. Qadamlar soni (n - 1). Shunday qilib, bir xil me’yorda o‘lchash kriteriyasi uchun vaqt murakkabligi O(n) dir.

Logarifmik o‘lchash kriteriyasi bilan vaqt murakkabligi. Shu nuqtada, baholanishi kerak bo‘lgan amallarni ajratib ko‘rsatish kerak. Birinchidan, bu **taqqoslash** amallari. Ikkinchidan, o‘zgaruvchi qiymatiga ta’sir qiluvchi amallar (qo‘shish, ko‘paytirish, bo‘lish, ayirish). O‘zlashtirish (ta’minlash) amali hisobga olinmaydi, chunki ular bir zumda amalga oshiriladi deb taxmin qilinadi.

Shunday qilib, ushbu masala uchta amal ajratilgan:

1) $i \leq n$ i-qadamda $\log(n)$ ni olamiz. Qadamlar soni $n - 1$ ta bo‘lgani uchun ushbu amalning murakkabligi $(n - 1) * \log(n)$ ga tengdir.

2) i++; i-qadamda $\log(n)$ ni olamiz.

Ushbu holatda, quyidagicha yig‘indi hosil bo‘ladi:

$$\sum_{i=2}^{n} \log_2 i = \log_2(i!)$$

3) result *=i; i-qadamda $\log\big((i-1)!\big)$ ni olamiz.

Ushbu holatda, quyidagi yig‘indi hosil bo‘ladi:

$$\sum_{i=2}^{n} \log_2\big((i-1)!\big) = \log_2\left(\prod_{i=2}^{n}(i-1)!\right)$$

Agar biz barcha olingan qiymatlarni yig‘sak va n ning o‘sishi bilan asta sekin o‘sadigan atamalarni bekor qilsak,

$$O(\log_2\left(\prod_{i=2}^{n}(i-1)!\right))$$

biz yakuniy ifodasini olamiz.

Bir xil me’yorda o‘lchash kriteriyasi bo‘yicha sig‘imning murakkabligi. Bu yerda hamma narsa oddiy. O‘zgaruvchilar sonini hisoblashingiz kerak. Agar topshiriq massivlardan foydalansa, massivdagi har bir yacheyka o‘zgaruvchi hisoblanadi. O‘zgaruvchilar soni kirish kattaligiga bog‘liq bo‘lmaganligi sababli, murakkablik O (1) bo‘ladi.

Logarifmik o‘lchash kriteriyasi ega bo‘lgan sig‘imning murakkabligi. Bunday holda, siz xotira yacheykasida bo‘lishi mumkin bo‘lgan maksimal qiymatni hisobga olishingiz kerak. Agar qiymat aniqlanmagan bo‘lsa (masalan, $i > 10$ operand bo‘lganda), u holda V_{max} chegaraviy qiymati bor deb hisoblanadi.

Ushbu masalada qiymati n (i) dan oshmaydigan va n (result) qiymatidan oshmaydigan o‘zgaruvchi mavjud. Shunday qilib, $O(\log(n!))$ ga teng.

2-misol. Massiv elementlari yig‘indisi. Bir o‘lchovli massivning barcha elementlari qiymatlari yig‘indisini hisoblaydigan quyidagi algoritm bor deylik:

```
(1)  S=0;
(2)  for(int i=0; i<n; i++)
(3)  S=S+A[i];
```

Algoritmni bitta satrda bitta operator bo‘ladigan tarzda yozish kerak. Bundan tashqari, har bir bajarilgan operator yonida, ushbu operatorning necha marta bajarilishini ko‘rsatadigan kirish ma’lumotlarining o‘lchamiga bog‘liq bo‘lgan ifoda yozishingiz kerak. Ushbu baholash ozmi-ko‘pmi aniqroq bo‘lishi mumkin, asosiysi siz uni xuddi shu tarzda bajarishingiz kerak. Masalan, har bir bayonot bir mavhum vaqt birligida bajarilgan deb taxmin qilishingiz mumkin. Yoki har bir operatorning bajarilishini elementar amallar ketma-ketligiga ajrating: xotiradan o‘qing, xotiraga yozing, arifmetik amalni bajaring.

Birinchi yondashuvda biz quyidagi taxminlarni olamiz. Birinchi ifoda bir marta bajariladi va u kiritilgan ma’lumotlarning o‘lchamiga bog‘liq emas. Ikkinchi operatorning bajarilish soni kiritilgan ma’lumotlarning o‘lchamiga bog‘liq (xususan, n – massivning uzunligiga). Ushbu holatda bu n + 1 (for siklining boshi uning tanasidan bir marta ko‘proq bajarilishini unutmang). Shunga ko‘ra uchinchi operator n mavhum vaqt birligida bajariladi. Shunday qilib, bizda:

```
S=0;                    (1)
for(int i=0; i<n; i++)  (n+1)
S=S+A[i];               (n)
```

Barcha operatorlarning algoritmlar murakkabligi yig‘indisini hisoblash natijasida 2n + 2 murakkablikni olish mumkin.

Algoritmlarni baholashda ko‘pincha quyidagi funksiyalar qo‘llaniladi: $\log_2 n$, n, $n \cdot \log_2 n$, n^2, n^3 , $2n$, $10n$, $n!$. $O(\log n)$ ko‘rinishida baholangan algoritmlar, har qanday sababga ko‘ra, juda tez algoritmlar deb nomlanadi. Bunday algoritmlar unchalik ko‘p emas. Aslida, adabiyotda odatda $O(\log_2 n)$ bahoga ega bo‘lgan faqat bitta algoritm zikr qilinadi - bu ikkilik qidiruv algoritmi. Buni keyinroq ko‘rib chiqamiz. $O(n)$ va $O(n \log n)$ deb baholangan algoritmlar **tezkor algoritmlar** deb ataladi.

$O(n^2)$, $O(n^3)$ yoki umumiy holatda $O(n^C)$ bo‘lgan algoritmlar **polinomial algoritmlar** deyiladi, O(2n), O(10n), O (n!) baholangan algoritmlar esa **polinomial bo‘lmagan** algoritmlar deyiladi.

3-misol. Amaliy muhim masalalarning o‘lchamlari odatda ancha katta. Algoritmlarning misollarini ko‘rib chiqamiz, ularning baholanishi nafaqat kirish ma’lumotlarining o‘lchamiga bog‘liq. Bu bir o‘lchovli massivning elementlari orasida eng katta (eng kichik) qiymatni topish uchun keng qo‘llaniladigan algoritm:

```
(1)    max = a[0];                    1
(2)    for(int i=2; i<=n; i++)        n
(3)    if (max<a[i])                  n-1
(4)      max = a[i];                  0 dan n-1 ta
```

Operatorlarning 1, 2 va 3 baholarini topish to‘g‘ri bo‘lishi kerak. Ammo 4-operatorning bajarilish soni massiv tarkibiy qismlarining o‘ziga xos qiymatlariga bog‘liq, shuning uchun biz aniq baho berolmaymiz. Bunday holda, quyidagicha harakat qiling. Ular bitta baho bilan emas, balki uchta baho bilan baholanadi: eng yaxshi, eng yomon va o‘rtacha. Ushbu uchta baholashdan eng qiyini o‘rtacha qiymatni topishdir (hatto o‘rtacha nimani anglatishini shakllantirish ham), garchi amaliy nuqtai nazardan bu eng muhimi bo‘lsa ham. Birinchi kurs talabalari uchun bu, ehtimol, qiyin muammo, shuning uchun biz bu yerda ko‘rib chiqmaymiz.

Eng yaxshi va yomon baholarni topish osonroq. Tegishli operator mos ravishda eng kam va ko‘p marta bajariladigan bunday kirish ma’lumotlarini tasavvur qilish kerak.

Bizning misolimiz uchun eng yaxshi kirish ma’lumoti birinchi raqami maksimal bo‘lgan massiv bo‘lishi mumkin. Bunday holda, 4-operator hech qachon bajarilmaydi, chunki 3-operatordagi shart har doim yolg‘on bo‘ladi. Eng yomon kirish ma’lumotlari esa eng katta element eng oxirida bo‘lgan massiv bo‘lishi mumkin. Bunday holda, 3-operator shart har safar rost bo‘ladi va 4-operator har safar bajariladi.

Shunday qilib, bizning algoritmimizning eng yaxshi bahosi 2n, eng yomoni esa 3n-1.

4-misol. Ichki sikllarni o‘z ichiga olgan yanada murakkab algoritmlarni baholashning misoli sifatida ko‘rib chiqamiz.

Masalaning sharti quyidagicha bo‘lsin. a1, a2, ..., an butun sonlar massivi berilgan. Massiv tarkibiy qismlarini kamaymaydigan tartibda joylashtiring.

Birinchi algoritm massiv elementlarni tanlash usuli yordamida saralash. Uning mohiyati quyidagicha. Butun massivda minimal element qidiriladi va birinchi o‘ringa qo‘yiladi, massivning qolgan qismida minimal qidiriladi va ikkinchi o‘ringa qo‘yiladi va hokazo, massivning oxirgi elementi ko‘rib chiqilmaguncha bu ish davom ettiriladi. Ushbu algoritmdan odamlar kundalik hayotda foydalanadilar. Bu algoritm ancha "yomon", ya’ni samarasiz ko‘rinadi. Keling, ushbu algoritmni batafsil ko‘rib chiqaylik:

```
(1)     for(int i=0; i<n; i++)        n+1
    {
(2)       min1=A[i];            n
(3)       k=i;                n-1
(4)       for(int j=i+1; j<n; j++)    (n2+3n-2)/2
(5)         if(min1>A[j]){              (n2+3n)/2
(6)           min1=A[j];              0 dan (n2+3n)/2 gacha
(7)            k=j;                   0 dan (n2+3n)/2 gacha
(8)           }
(9)         A[k]=A[i];                n-1
(10)          A[i]=min1;              n-1
(11)    }
```

4-operatorning bahosi quyidagicha olinadi. Ushbu sikl sarlavhasi har bir i uchun bajariladi va i = 0 uchun n+1 marta, i = 1 uchun n marta, n-1, n-2, ..., 0 marta bajariladi. Ya’ni, bizda a_1, a_2, ..., a_m arifmetik progressiyasi bor, ularning yig‘indisi

$$s_m = \frac{(a_1 + a_m) \cdot m}{2}$$

Bu yerda m = n+1, a_1 = n+1, a_m = 1 va yig‘indisi $(n^2+3n-2)/2$.

5-operatorning bahosi xuddi shu tarzda olinadi, shunchaki unutmang for siklining asosiy qismi har doim tanasiga qaraganda yana bir marta bajariladi. Bu yerda $\frac{n^2+3n-2}{2} - 1 = \frac{n^2+3n}{2}$.

6- va 7-operatorlarning baholariga kelsak, biz bunga oldinroq duch kelganmiz.

Kiritilgan ma’lumotlarga qarab, ushbu operatorlar boshqacha marta bajarilishi mumkin. "Yaxshi" ma’lumotlar uchun (massiv allaqachon saralangan bo‘lsa), bu operatorlar hech qachon bajarilmaydi. "Yomon" ma’lumotlar uchun ushbu operatorlar har bir tekshirishda min>a[j] bajariladi. Algoritmlarni saralash uchun "yomon" ma’lumotlar teskari tartibda saralangan massivdir.

Shunday qilib, saralash algoritmi uchun eng yaxshi ball n^2 + 8n-5, eng yomon ko‘rsatkich esa $2n^2$+11n-5. Yomonmi yoki yaxshimi? Hozircha faqat massiv allaqachon saralangan bo‘lsa, algoritm ko‘p vaqt sarflashini sezishingiz mumkin. Bu ushbu algoritmning aniq kamchiliklaridir.

Mavzu yuzasidan savollar:

1. Algoritm tushunchasiga ta’rif bering.
2. Algoritm xossalari va uni tasvirlash usullariga to‘xtalib o‘ting
3. Algoritm yaxshi, o‘rtacha, yomon bahosiga misollar keltiring
4. Polinomial va Polinomial bo‘lmagan bahoga ega bo‘lgan algoritmlarga misollar keltiring
5. Tezkor algoritmlarga misollar keltiring

Mustaqil ishlash uchun masalalar:

1. Natural son berilgan. Undagi oxirgisiga teng bo‘lgan raqam necha marta uchrashini aniqlash dasturini tuzing va algoritm murakkabligini baholang.

2. 0 bilan tugaydigan bo‘sh bo‘lmagan a_1, a_2, … musbat butun sonlar ketma-ketligi berilgan. a_1,$a_1 \cdot a_2$,$a_1 \cdot a_2 \cdot a_3$,…,0 ko‘rinishidagi ketma-ketlikni hosil qilish dasturini tuzing va algoritm murakkabligini baholang

3. Natural son berilgan. Kattasiga teng bo‘lgan raqamlar necha marta uchrashini aniqlash dasturini tuzing va algoritm murakkabligini baholang

4. Navbat bilan tanlash orqali ikkita bir o‘lchamli massivlarni qo‘shish dasturini tuzing va algoritm murakkabligini baholang

5. A[N] bir o‘lchamli massiv berilgan. $\max(a_2,a_4,\ldots,a_{2k})+\min(a_1,a_3,\ldots,a_{2k+1})$ ni topish dasturini tuzing va algoritm murakkabligini baholang

6. N ta butun sonlar ketma-ketligi berilgan. Tartib nomeri o‘zining qiymatiga mos keladigan massiv elementlarini yig‘indisini hisoblash dasturini tuzing va algoritm murakkabligini baholang

7. N ta haqiqiy sonli ketma-ketlik berilgan. Ularning ichida K dan kichik, K ga teng va K dan katta sonlar qanchaligini aniqlang

8. a_1,a_2,… ,a_n haqiqiy sonlar ketma-ketligi berilgan. Uning berilgan Z sonidan katta barcha hadlarini shu son bilan almashtiring. Almashishlar miqdorini hisoblash dasturini tuzing va algoritm murakkabligini baholang

9. Berilgan natural sonni tub ko‘paytuvchilarga ajratish dasturini tuzing va algoritm murakkabligini baholang

2-§. Ma'lumotlarning abstrak turlari va ma'lumotlar strukturalari

“Yomon dasturchilar kod haqida o‘ylashadi. Yaxshi dasturchilar esa ma’lumotlar strukturasi va ularning aloqalari haqida o‘ylashadi.”
Linus Torvalds, Linux yaratuvchisi.

Ma’lumotlar strukturasi (ing. data structure) - bu hisoblashda turli xil bir tipli va (yoki) mantiqiy bog‘liq ma’lumotlarni saqlash va qayta ishlashga imkon beradigan dastur birligi. Ma’lumotlarni qo‘shish, izlash, o‘zgartirish va yo‘q qilish uchun ma’lumotlar tarkibi uning interfeysini tashkil etadigan funksiyalar to‘plamini taqdim etadi.

“Ma’lumotlar strukturasi” atamasining bir-biriga yaqin bo‘lgan bir nechta ma’nolarini anglatuvchi variantlari mavjud:

- Ma’lumotlarning abstrakt turi;
- Ma’lumotlarning ba’zi bir abstrakt turlarini realizatsiya qilish;
- Ma’lumotlar tipining nusxasi, masalan, aniq bir **ro‘yxat**;
- Funksional dasturlash kontekstida o‘zgarishlarda davom etadigan noyob identifikator.

Turli xil versiyalar mavjud bo‘lishiga qaramay, u norasmiy ravishda **ma’lumotlar strukturasi** deb nomlanadi.

Ma’lumotlar strutkturasi ma’lumotlar turlari, havolalar va ular ustida amallar yordamida tanlangan dasturlash tilida shakllanadi.

Turli xil ma’lumotlar strukturalari turli xil ilovalar uchun mos keladi; ularning ba’zilari ma’lum vazifalar uchun tor ixtisoslashgan. Masalan, B-daraxtlar odatda ma’lumotlar bazalarini yaratishga yaroqli bo‘lsa, xesh jadvallar hamma joyda har xil lug‘atlarni yaratish uchun ishlatiladi, masalan, domen nomlarini kompyuterlarning internet-manzillariga xaritalash uchun.

Dasturiy ta’minotni ishlab chiqishda, amalga oshirishning murakkabligi va dasturlarning ishlash sifati ma’lumotlar strukturalarning to‘g‘ri tanlanishiga sezilarli darajada bog‘liqdir. Ushbu tushuncha dasturiy ta’minot arxitekturasining boshida algoritmlar emas, ma’lumotlar strukturalari joylashtirilgan rasmiy ishlab chiqish usullari va dasturlash tillarini keltirib chiqardi. Ushbu tillarning aksariyatida

ma’lumotlar tuzilmalarini turli xil ilovalarda xavfsiz qayta ishlatishga imkon beradigan modullikning bir turi mavjud. Java, C# va C++ kabi obyektga yo‘naltirilgan tillar ushbu yondashuvga misoldir.

Ko‘pgina klassik ma’lumotlar strukturalari dasturlash tillarining standart kutubxonalarida taqdim etiladi yoki to‘g‘ridan-to‘g‘ri dasturlash tillariga kiritilgan. Masalan, xesh jadvali ma’lumotlar tuzilishi Lua, Perl, Python, Ruby, Tcl va boshqa dasturlash tillariga kiritilgan bo‘lib, C++ standart shablonlar kutubxonasida (STL) keng qo‘llaniladi.

Ko‘pgina ma’lumotlar tuzilmalari uchun asosiy qurilish bloklari massivlar, yozuvlar (C tilida strukturalar va Paskaldagi yozuvlar), birlashmalar (C da union) va havolalardir.

Funksional va imperativ dasturlashda ma’lumotlar strukturalarini taqqoslash. Kamida ikkita sababga ko‘ra funksional tillar uchun ma’lumotlar tuzilmalarini loyihalash imperativ tillarga qaraganda ancha qiyin:

1. Deyarli barcha ma’lumotlar strukturalari aniq funksional uslubda ishlatilmaydigan o‘zlashtirishlardan og‘ir foydalanadilar;

2. Ma’lumotlarning funksional strukturalari yanada moslashuvchan, shuning uchun imperativ dasturlashda eski versiya yo‘qoladi, shunchaki yangisi bilan almashtiriladi, funksional ravishda u avtomatik ravishda mavjud bo‘lib qoladi. Boshqacha qilib aytganda, imperativ dasturlashda (agar siz dasturni jiddiy ravishda murakkablashtirishi mumkin bo‘lgan maxsus choralarni ko‘rmasangiz) ma’lumotlar strukturalari **vaqtinchalik (ing. ephemeral)**, funksional dasturlarda ular odatda **doimiydir**.

Abstrakt ma’lumotlar turi (ADT – Abstract Data Type) - bu ma’lumotlar turlari uchun matematik model, bu yerda ma’lumotlar turi xatti-harakatlari (semantikasi) bilan foydalanuvchi nuqtai nazaridan aniqlanadi, ya’ni mumkin bo‘lgan qiymatlar, ushbu ma’lumotlar bo‘yicha mumkin bo‘lgan amallar turi va ushbu amallarning harakati.

Rasmiy ravishda, ADTni komponentalar ro‘yxati bilan belgilanadigan obyektlar to‘plami (bu obyektlarga taalluqli amallar va ularning xususiyatlari) deb ta’riflash mumkin. Ushbu turdagi barcha ichki tuzilish dasturiy ta’minot ishlab chiqaruvchisidan yashirilgan

holatda bo‘ladi - bu abstraksiyaning mohiyati. Abstrakt ma’lumotlar turi, uning qiymatlari bo‘yicha ishlash uchun tipning aniq bajarilishidan mustaqil funksiyalar to‘plamini belgilaydi. ADTlarning aniq tatbiq etilishi ma’lumotlar strukturasi deb ataladi.

Dasturlashda abstrakt ma’lumotlar turlari odatda tegishli turdagi amallarni yashiradigan interfeyslar sifatida ifodalanadi. Dasturchilar mavhum ma’lumotlar turlari bilan faqat o‘z interfeyslari orqali ishlaydi, chunki kelajakda dastur o‘zgarishi mumkin. Ushbu yondashuv obyektga yo‘naltirilgan dasturlashda inkapsulyatsiya tamoyiliga mos keladi. Ushbu texnikaning kuchli tomoni - bu dasturni yashirish. Faqatgina interfeys tashqarida namoyish etilganidan so‘ng, ma’lumotlar tuzilishi ushbu interfeysni qo‘llab-quvvatlagan ekan, mavhum ma’lumotlar turining berilgan tuzilishi bilan ishlaydigan barcha dasturlar ishlashni davom ettiradi. Ma’lumotlar tuzilmalarini ishlab chiquvchilar tashqi interfeys va funksiyalar semantikasini o‘zgartirmasdan, algoritmlarni tezligi, ishonchliligi va ishlatilgan xotirasi jihatidan takomillashtirib, tatbiq etishni bosqichma-bosqich takomillashtirishga harakat qilishadi.

Ma’lumotlarning abstrakt turlari dasturiy mahsulotlarning modulliligiga erishishga va alohida modulning bir-birining o‘rnini bosadigan bir nechta muqobil dasturlariga ega bo‘lishga imkon beradi.

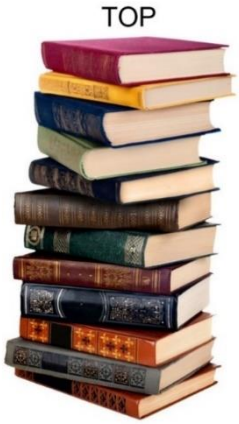
Tez-tez muammolarni hal qilish yoki dasturlarni optimallashtirish uchun ishlatiladigan ma’lumotlar strukturasini tahlil qilamiz.

2.1. Stek

Stek – Stack inglizchadan uyum, g‘aram, dasta, bog‘lam degan ma’noni anglatadi.

Stek - bu LIFO (last in – first out; oxirgi kelgan – birinchi ketadi) prinsipi bo‘yicha ishlaydigan ma’lumotlar strukturasi.

Bu juda aniq ta’rif, ammo yangi o‘rganuvchilar uchun tushunish biroz qiyin bo‘lishi mumkin. Shuning uchun, hayotning narsalar ko‘rinishidagi to‘plamni taqdim etish haqida to‘xtalib o‘taylik. Xayolga kelgan birinchi narsa - bu kitoblar to‘plami ko‘rinishidagi talqin, bu yerda eng yuqori kitob tepada joylashgan.



1-rasm. Kitoblar ustuni (Stek ma’lumotlar strukturasi)

Aslida, stek har qanday narsaning to‘plami sifatida ifodalanishi mumkin, u daftar, ruchka va shunga o‘xshash narsalar to‘plami bo‘lishi mumkin, ammo kitoblar bilan misol eng maqbul bo‘ladi.

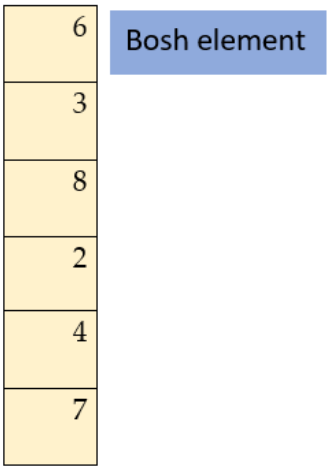
Shunday qilib, stek nimadan iborat? Stek katakchalardan iborat (masalan, bular kitoblar), ular ba’zi ma’lumotlarni o‘z ichiga olgan tuzilish shaklida va ushbu strukturaning turiga keyingi elementga ko‘rsatkich sifatida taqdim etiladi.

Stekka birinchi bo‘lib kiritilgan element eng so‘nggisi bo‘ladi. Agar stekka uchta element qo‘shsangiz, avval qo‘shilgan oxirgi element o‘chiriladi.

2-rasmda siz 6 ta raqamni ko‘rishingiz mumkin: 6, 3, 8, 2, 4, 7. Shunga diqqat qilingki, biz ularni bir xil tartibda chiqaramiz. Masalan, 8 raqamini chiqarish uchun avval 6 va 3 raqamlarini, so‘ngra 1 ni ajratib olishimiz kerak, chunki bu stek, biz bu raqamlarni teskari tartibda qo‘shdik. Aniqroq qilib aytganda: 7, 4, 2, 8, 3, 6.

Stekda massivdagi kabi indekslar mavjud emas, demak ma’lum bir elementga murojaat qila olmaysiz. Buning sababi, stek bog‘langan ro‘yxatlar asosida tuzilgan. Bu shuni anglatadiki, har bir element (oxirgisidan tashqari qolgan elementlar NULL-ga ishora qiladi, oddiy so‘zlar bilan aytganda, hech narsaga ishora qilmasa NULL bo‘ladi) keyingi elementga ko‘rsatgichga ega. Ammo ko‘rsatgich bo‘lmagan element mavjud - birinchisi (yoki uni bosh element deb ham atashadi).

Shu o‘rinda savol paydo bo‘lishi mumkin? Nima uchun massivlarni ishlatish mumkin bo‘lganda stekni ishlatamiz. Sababi stek to‘plamining



2-rasm. Stek ma’lumotlar strukturasi

asosiy kuchi elementlarni qo‘shish va olib tashlashdan iborat ekanligida. Ushbu amallar **doimiy vaqt** ichida amalga oshiriladi (bu yaxshi plyus). Ba’zi dasturchilar massivda stek qilishadi. Stekdan foydalanishning bu usuli haqida biroz keyinroq gaplashamiz.

C ++ tilida stekni realizatsiya qilish. Dastur boshida stek shablonidan foydalanish uchun <**stack**> kutubxonasini yoqishimiz kerak.

Stek yaratish uchun biz quyidagi sxema bilan ishlashimiz kerak:

stack <ma’lumot_turi> <nom>;

Yangi satrda **stack** kalit so‘zini yozishimiz kerak.

<ma’lumotlar turi> - bu yerda stekda saqlanadigan ma’lumotlar turini yozishimiz kerak.

<nom> - bu stek nomi.

Steklar bilan ishlash metodlari. Metodlar - navbat va stek kabi konteynerlar uchun ishlatiladigan funksiyalar. Quyida stekda ishlatiladigan metodlarni qarab chiqamiz:

```
#include <iostream>
#include <stack>  //stek kutubxonasini ulash
```

```
using namespace std;
int main()
{
  stack <int> stek;  // Stek yaratish
  int i = 0;
  cout << "Istalgan oltita son kiriting: " << endl;
  while (i != 6)
   {
      int a;
      cin >> a;
      stek.push(a);  // Sonni stekka qo‘shish
      i++;
   }

   if (!stek.empty())
      cout << "Stek bo‘sh emas";  //Stekni bo‘shligini tekshirish
  cout << "Stekning yuqori elementi: " << stek.top() << endl; // Eng yuqori elementni chop etish
  cout << "Eng yuqori elementni olib tashlash " << endl; //
  stek.pop();  // yuqori elementni o‘chirish
  cout << "Bu endi yangi yuqori element: " << stek.top();
  return 0;
}
```

Dasturda berilgan **push()** funksiyasi yordamida stekka element qo‘shamiz. Qavslar ichida biz qo‘shmoqchi bo‘lgan qiymat bo‘lishi kerak. Dastur kodida stek bo‘shligini tekshirish uchun **empty()** metodidan foydalanamiz. Agar bu funksiya natijasi **true** bo‘lsa, u holda stek bo‘sh bo‘ladi. Agar natija false bo‘lsa, unda stekda element mavjud bo‘ladi. Stekning eng yuqori elementini o‘chirish uchun **pop()** funksiyasi ishlatilgan.

pop() funksiyasida, push() funksiyasidan farqli o‘laroq, qavs ichida biror narsani ko‘rsatishning hojati yo‘q, lekin qavsning o‘zi bo‘lishi kerak.

Stekning eng yuqori elementini olish uchun **top()** funksiyasidan foydalanamiz.

peek() funksiyasi. stack kutubxonasiga yangi peek () funksiyasi qo‘shildi, u yordamida stekning N-elementiga murojaat qilishingiz mumkin (yuqoridan).

Shu holatda endi stek massivga o‘xshash holatda bo‘ladi.

Quyida peek() funksiyasidan foydalanib uchinchi elementni chiqaramiz. Ushbu funksiya C++11 standartidan keyin qo‘shilgan.

```
#include <iostream>
#include <stack>  //stek kutubxonasini ulash
using namespace std;
int main()
{
  stack <int> stek;  // Stek yaratish
  stek.push(2);
  stek.push(3);
  stek.push(9);
  stek.push(10);
  cout<<"Stekning uchinchi elementi:"<<stek.peek(3);
  return 0;
}
```

peek() funksiyasidan dasturchilarning kichik doirasi foydalanadi va aytish mumkinki, bu funksiya yaratuvchilar undan kutganidek ommalashmagan.

Massiv yordamida stek hosil qilish. Ko‘plab dasturchilar stek shablonidan foydalanmaydilar, aksincha massivlar orqali stek bilan ishlashadi. Endi massiv yordamida stekni qanday amalga oshirishni ko‘ramiz:

Quyida biz 20 ta elementdan iborat - stek nomli massiv yaratdik, shuningdek, i o‘zgaruvchini yaratdik, bu esa stekning yuqori elementiga ishora qiladi. Element qo‘shish uchun biz *i* ni birma-bir oshiramiz va elementni stek[i] katakchasiga yozamiz. Elementni olib tashlash uchun biz shunchaki *i*-ni kamaytiramiz. Ehtimol, allaqachon taxmin qilganingizdek, stekning yuqori elementiga murojaat qilish uchun biz shunchaki qatorning *i* elementiga murojaat qilamiz. *i* o‘zgaruvchisi **push()** funksiyasi va **top()** funksiyasini almashtirdi. Stekning bo‘shligini

ko‘rish uchun biz shunchaki i == -1 holatini tekshiramiz: agar u rost bo‘lsa, demak stek bo‘sh, aks holda bo‘sh emas.

Stekni massiv yordamida amalga oshirish quyida keltirilgan:

```
#include <iostream>
using namespace std;
int main() {
  int stek[20];
  int i = -1;

  for (int j = 0; j < 6; j++) {
    int a;
    cin >> a;
    i++;
    stek[i] = a;
  }
  if (i == -1) cout << "Stek bo‘sh";

  cout << stek[i] << " stekning yuqori elementi"<<endl;

  cout << "Yuqori elementini o‘chirish";

  i--;
  return 0;
}
```

Yuqorida stekni realizatsiya qilishning ikkita usulini ko‘rib chiqdik:

1) C++ shablonidan foydalanish.
2) Massivdan foydalanish.

Agar siz dasturingizda stekdan foydalansangiz va uni iloji boricha tezroq bajarishni afzal bilsangiz, unda stekni amalga oshirishning birinchi usulidan foydalaning.

Agar siz dasturning ishlashi haqida qayg‘urmasangiz, unda massiv orqali stek yaratilishidan foydalanishingiz mumkin. Birinchi usulda ishlatish va e’lon qilish tez va oson.

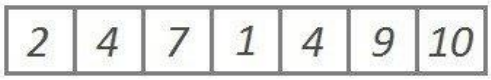
Quyida yana bir juda muhim ma’lumotlar tuzilishini - navbatni o‘rganamiz. Ushbu ma’lumotlar tuzilishi ko‘plab messenjerlarda (masalan, telegramda) foydalaniladi.

2.2. Navbat

Navbat. Navbat - bu FIFO (First In - First Out - "birinchi kelgan – birinchi ketadi") prinsipi bo‘yicha qurilgan ma’lumotlar strukturasi.

Navbatda, agar siz avval kiritilgan elementni qo‘shsangiz, u birinchi bo‘lib chiqadi. Agar 4 ta element qo‘shsangiz, birinchi qo‘shilgan element birinchi bo‘lib chiqadi.

Navbat qanday ishlashini tushunish uchun siz xarid qilish navbatini tasavvur qilishingiz mumkin va siz uning o‘rtasida turibsiz, shunda siz kassaga qarama-qarshi bo‘lasiz, avval siz oldingizdagi barcha odamlarga xizmat qilishingiz kerak bo‘ladi. Ammo navbatda turgan oxirgi odam uchun kassir o‘zidan boshqa barcha odamlarga xizmat qilishi kerak.



3-rasm. Navbat ma’lumotlar strukturasi

Rasmda 7 ta raqam mavjud: 2, 4, 7, 1, 4, 9, 10. Agar ularni ajratib olishimiz kerak bo‘lsa, biz ularni rasmdagi kabi tartibda chiqaramiz!

Masalan, 4-raqamni ajratib olish uchun avval 2-raqamga, so‘ngra 4-raqamga xizmat ko‘rsatishimiz kerak.

Stekda peek() funksiyasi mavjud bo‘lsa-da (bu elementga indeks bo‘yicha kirishga imkon beradi), navbat shablonidagi ma’lum bir elementga murojaat qilish mumkin emas.

Agar siz navbatning barcha elementlariga kirishingiz kerak bo‘lsa, unda siz navbatni massiv orqali amalga oshirishingiz mumkin. Quyida biz buni qanday bajarilishini ko‘rib chiqamiz.

C++ tilida navbatni realizatsiya qilish. Agar siz C++da navbat shablonidan foydalanmoqchi bo‘lsangiz, unda avval <**queue**> kutubxonasini kiritishingiz kerak. Bundan tashqari, navbatni e’lon qilish uchun quyidagi strukturani ishlatishingiz kerak.

queue <ma’lumot turi> <nom>;

Misol uchun:

queue <int> navbat;

Navbatning metodlari. Navbat bilan ishlash uchun funksiyalarni bilish kerak: push(), pop(), front(), back(), empty().

1. Navbatga yangi element qo‘shish uchun **push()** funksiyasidan foydalanish kerak. Qavslar tarkibida biz qo‘shmoqchi bo‘lgan qiymat bo‘lishi kerak.

2. Agar biz birinchi elementni olib tashlashimiz kerak bo‘lsa, **pop()** funksiyasi bilan ishlashimiz kerak. Qavslar ichida endi ko‘rsatilishi kerak bo‘lgan narsa yo‘q, lekin qoidalarga ko‘ra, ular albatta mavjud bo‘lishi kerak. Ushbu funksiyalarga argument kerak emas: empty(), back() va front().

3. Agar navbatning birinchi elementiga murojaat qilishingiz kerak bo‘lsa, unda **front()** funksiyasi kerak.

4. **back()** funksiyasi navbatdagi oxirgi elementga kirishga yordam beradi.

5. Navbatning bo‘shligini bilish uchun **empty()** funksiyasidan foydalanish mumkin.

- Agar sizning navbatingiz bo‘sh bo‘lsa, u **true** qiymatini qaytaradi.
- Agar unda biror narsa bo‘lsa, u false qaytadi.

Quyida biz yuqoridagi metodlarning barchasini qo‘llaymiz:

```
#include <iostream>
#include <queue>  // Queue kutubxonasini ulash
using namespace std;
int main() {
  queue <int> N;  // Navbat yaratish
  cout << "Yettita son kiriting: " << endl;
  for (int h = 0; h < 7; h++)
    {
```

```
    int a;
    cin >> a;
    N.push(a);  // Navbatga element qo‘shish
  }
  cout << endl;
  cout << "Eng birinchi elementi: " << N.front() << endl;
  N.pop();  // Navbatdan element o‘chirish
  cout << "Birinchi element: " << N.front() << endl;
  if (!N.empty()) cout << "N bo‘sh emas!";
  return 0;
}
```

Massiv yordamida navbat yaratish. Yuqorida aytib o‘tganimizdek, navbatni massiv orqali amalga oshirish mumkin. Odatda, agar bunday navbat yaratilsa, massiv navbat deyiladi.

```
#include <iostream>
#include <queue>  // queue kutubxonasini ulash
using namespace std;
int main()
{
  int N[7];
  int start = 0, ends = 0;
  cout << "7 ta son kiriting: " << endl;
  for (int h = 0; h < 7; h++)
    {
       int a;
       cin >> a;
       N[ends++] = a;  // Navbatga (massivga) element qo‘shish
  }
    cout << "Eng birinchi element: " << N[start] << endl;
    start++;
  cout << "Navbatning eng oxirgi elementi: " << N[ends - 1];
  if (start != ends) cout << "Navbat to‘lgan!";
}
```

2.3. Vektor

Vektorlar. Vektor - bu dinamik massiv modeli bo‘lgan ma’lumotlar strukturasi.

Dinamik massivni yaratish uchun (qo‘lda) yangi konstruktor va qo‘shimcha ko‘rsatkichlardan foydalanish kerakligini eslaylik. Biroq, vektorlarga qaraganda, bularning barchasini qilishingiz shart emas.

C++ tilida vektorlar yaratish. Birinchi navbatda vektorlar yaratish uchun <vector> kutubxonasini bog‘lash kerak. Xuddi stek va navbat konstruksiyasi kabi u ham quyidagicha e’lon qilinadi:

```
vector <ma’lumot turi> <VektorNomi>
```

Bundan tashqari vektorga boshlang‘ich qiymatlar berishingiz mumkin.

Masalan:

```
vector <int> V = {7, 4, 3};
```

Vektor yacheykasiga murojaat qilishning ikkinchi usuli. Bilamizki, vektor yacheykasiga murojaat qilish uchun indekslardan foydalanadi. Odatda biz ularni kvadrat qavslar [] bilan birgalikda ishlatamiz.

Ammo C++ da **at()** funksiyasi tufayli buni amalga oshirishning yana bir usuli mavjud. Qavslar ichida biz murojaat qilishimiz kerak bo‘lgan katak indeksini ko‘rsatishimiz kerak.

Bu amalda qanday ishlaydi:

```
vektor <int> V = {1, 2, 3};
V.at(1) = 8; // ikkinchi element qiymatini o‘zgartiradi
cout << V.at (1); // uni ekranda ko‘rsatish
```

Vektor uchun kataklar sonini qanday belgilash mumkin. Vektor o‘lchamini har xil usulda belgilashingiz mumkin. Siz buni ishga tushirish paytida ham qilishingiz mumkin yoki hatto dasturning oxirida

ham qilishingiz mumkin. Masalan, boshida vektor uzunligini aniqlashning bir usuli:

```
vector <int> V(5);
```

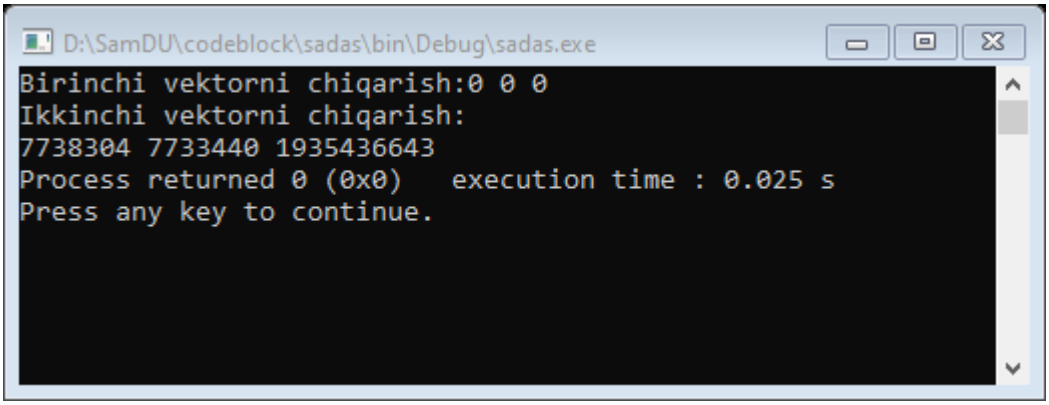
Shunday qilib, vektor nomidan keyin qavs ichida () biz boshlang‘ich uzunligini ko‘rsatamiz. Mana bu esa ikkinchi yo‘l:

```
vector <int> V;  // Vektor yaratish
reserve.V(5);  // Vektor uzunligini ko‘rsatish
```

Birinchi qator biz uchun allaqachon tanish. Ammo ikkinchisida noma’lum so‘z bor - **reserve**, bu kompilyatorga qancha yacheykadan foydalanishimiz kerakligini aytadigan funksiya.

Shu o‘rinda mantiqiy savol paydo bo‘lishi mumkin: "Bu ikki usulning farqi nima?". Quyida ikkita vektorni yaratamiz va ularning yacheykalar sonini boshqacha ko‘rsatamiz.

```
#include <iostream>
#include <vector>  //kutubxonani ulash
using namespace std;
int main() {
  vector <int> V1(3);
  vector <int> V2;
  V2.reserve(3);
  cout << "Birinchi vektorni chiqarish:";
  for (int i = 0; i < 3; i++) {
    cout << V1[i] << " ";
  }
  cout<<endl;
  cout << "Ikkinchi vektorni chiqarish: " << endl;
  for (int i = 0; i < 3; i++) {
    cout << V2[i] << " ";
  }
  return 0;
}
```



4-rasm. Vektorlar uzunligini aniqlashning ikki usuli uchun olingan natijalar

Ko‘rib turganingizdek, birinchi holatda biz uchta nolni, ikkinchisida: 7738304, 7733440, 1935436643 chop etilgani ko‘rishimiz mumkin. Buning sababi shundaki, birinchi usuldan foydalanganda barcha yacheykalar avtomatik ravishda nol bilan to‘ldirilgan.

Biror narsani e’lon qilayotganda (massiv, vektor, o‘zgaruvchi va boshqalar) biz kompyuter uchun keraksiz chiqindini o‘z ichiga olgan xotira yacheykalarining ma’lum bir qismini ajratamiz. Bizda bu chiqindi holatidagi raqamlardir.

Ikki vektorni qanday taqqoslash mumkin? Agar dastur o‘rtasida biz ikkita massivni taqqoslashimiz kerak bo‘lsa, albatta for siklidan foydalanamiz va barcha elementlarni birma-bir tekshiramiz.

Mana shu holatda vektorning yana bir yutuq tarafi bor. Ikki vektorni taqqoslash uchun biz faqat if tarmoqlanish operatorini qo‘llashimiz kerak.

```
if (V1 == V2) {  // Taqqoslash
  cout << "Ular bir xil!";
}
else {
  cout << "Ular ikki xil";
}
```

Albatta, kompilyator baribir yacheykalarni tekshirib, ushbu ikkita vektorni taqqoslab chiqadi.

Vektorlar vektori qanday yaratiladi?

Raqamlarni ikki o‘lchovli massivga yozishingiz kerak bo‘lishi aniq. Vektorlar vektori quyidagicha e’lon qilinadi:

```
vector <vector <ma’lumot_turi >> Vektor_nomi;
```

Misol uchun:

```
vector <vector <int>> V;
V.resize(10);
```

Vektorga vektorlarni qo‘shishning yana bir usuli bor. Ushbu usul uchun biz **push_back()** funksiyasidan foydalanamiz:

```
vec.push_back(vector <int>());
```

Ikki o‘lchovli vektor quyidagicha initsializatsiya qilinishi mumkin:

```
vector < vector <int> > V = {{1, 4, 7},
                             {2, 5, 8},
                             {3, 6, 9}};
```

Vektorlar uchun aniqlangan metodlar.

1) size() va empty(). Agar biz vektorning uzunligini bilishimiz kerak bo‘lsa, bizga size() funksiyasi kerak. Ushbu funksiya deyarli har doim for sikli bilan birgalikda ishlatiladi.

```
for(int i = 0; i < V.size(); i++)
{
//
}
```

Bundan tashqari, agar vektor bo‘shligini bilishimiz kerak bo‘lsa, biz - empty() funksiyasidan foydalanishimiz mumkin. Agar yacheykada qiymatlar bo‘lmasa, bu funksiya true qiymat qaytaradi. Aks holda yolg‘on qiymat qaytadi.

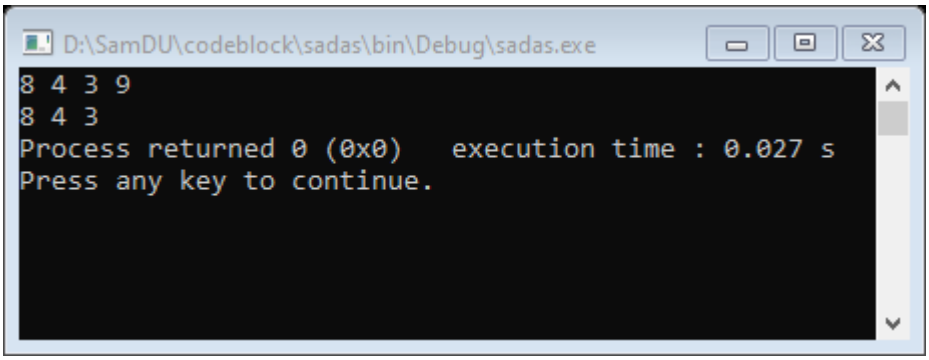
2) push_back() va pop_back().

2.1) push_back() funksiyasi yordamida biz vektorning oxiriga yacheyka qo‘sha olamiz.

2.2) pop_back() funksiyasi buning aksini qiladi - u vektor oxirida bitta yacheykani olib tashlaydi.

push_back() funksiyasidan yacheyka qiymatini ko‘rsatmasdan foydalanish mumkin emas.

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector <int> V3;
  V3.push_back(8);
  V3.push_back(4);
  V3.push_back(3);
  V3.push_back(9);
  for (int i = 0; i < V3.size(); i++) {
    cout << V3[i] << " ";
  }
  cout << endl;
  V3.pop_back();  // elementini o‘chirish
  for (int i = 0; i < V3.size(); i++) {
    cout << V3[i] << " ";
  }
  return 0;
}
```



5-rasm. Vektorlar ustida pop_back() va push_back() metodlarining foydalanish natijasi

3) insert() funksiyasi. Yuqorida biz push_back() funksiyasini ko‘rsatdik, lekin insert() funksiyasi yordamida ham shunday qilish mumkin. Faqatgina u yordamida siz vektorning boshiga elementlarni qo‘shishingiz mumkin.

insert() funksiyasini quyidagi dasturda qarab chiqamiz:

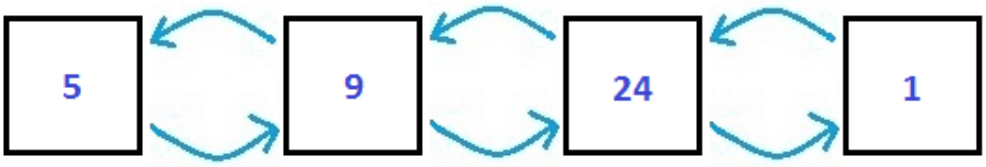
```
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector <int> V(2);

  V[0] = 2;
  V[1] = 3;

  for (int i = 0; i < V.size(); i++) {
    cout << V[i] << " ";
  }
  cout << endl;
  V.insert(V.begin(), 1);  // boshiga element qo‘shish

  for (int i = 0; i < V.size(); i++) {
    cout << V[i] << " ";
  }

  cout << endl;
  V.insert(V.end(), 4);  // Oxiriga element qo‘shish
```

```
  for (int i = 0; i < V.size(); i++) {
    cout << V[i] << " ";
  }
  return 0;
}
```

front() va back() funksiyalari. Vektorning birinchi va oxirgi yacheykalarining qiymatlarini bilish uchun front() va back() funksiyalarini ishlatamiz.

2.4. Ro‘yxat

Ro‘yxat - bu ikki tomonlama bog‘langan ro‘yxatlarga asoslangan ma’lumotlar strukturasi. Bu shuni anglatadiki, har qanday element faqat oldingi va keyingi elementlar haqida biladi. Quyidagi rasm bu qanday ishlashini ko‘rsatib beradi:

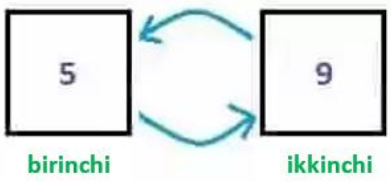


6-rasm. Ro‘yxat ma’lumotlar strukturasi

Ikki tomonlama bog‘langan ro‘yxatda indekslar mavjud emas, ammo C++ da uning o‘rniga iteratorlar mavjud.

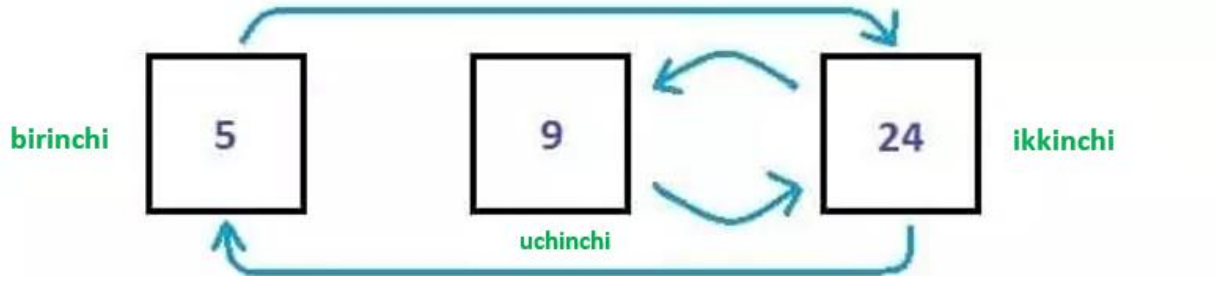
List1[2] = 8; // Bu xato!

Dasturchilar tezkor qo‘shish va olib tashlash sababli ushbu konteynerdan foydalanadilar. Bu juda tez sodir bo‘ladi, chunki elementlarni bir-birining orasidan siljitish shart emas, ko‘rsatkichlarni to‘g‘ri boshqarishingiz kerak.



7-rasm. Ro‘yxat ma’lumotlar strukturasi elementlari bog‘lanishi

Yuqoridagi misolda, boshida ikkita element bo‘lgan, keyin biz ular orasida bitta element qo‘shishga qaror qildik.



8-rasm. Ro‘yxat ma’lumotlar strukturasiga element qo‘shish

O‘chirish xuddi shu tarzda amalga oshiriladi.

9-rasm. Ro‘yxatdan elementni o‘chirish

Ro‘yxat hosil qilish. Dastlab list kutubxonasini ulash lozim.

```
#include <list>
```

Oldingi konstruktorlar kabi ro‘yxatni e’lon qilamiz:

```
list <ma’lumot_turi> <Ro‘yxat_nomi>
```

Masalan:

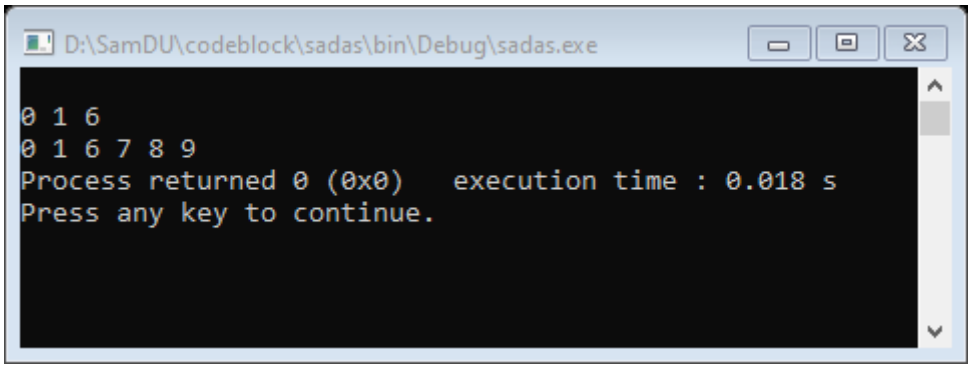
```
list <int> L = {4, 6, 3, 2}
```

Ro‘yxat bilan ishlash metodlari:

Funksiya nomi	Tavsif
pop_front()	Boshlang‘ich elementini o‘chirish
pop_back()	Oxirgi elementini o‘chirish
push_front()	Boshidan element qo‘shish
push_back()	Oxiridan element qo‘shish
front()	Birinchi elementiga murojaat
back()	Oxirgi elementiga murojaat
insert()	Ko‘rsatilgan joyga element qo‘shish
unique()	Barcha dublikatlarni o‘chirish
merge()	boshqa ro‘yxatni qo‘shish

```
#include <iostream>
#include <list>
#include <iterator>
using namespace std;
int main() {
  list <int> mylist;
  list <int> listmerge = {7, 8, 9};
  for (int i = 0; i < 2; i++) {
    for (int j = 1; j < 6; j++) {
      mylist.push_back(i); // 10 ta element qo‘shish
    }
  }
  mylist.insert(mylist.end(), 6);  // yangi element qo‘shish
    cout<<endl;
  mylist.unique();  // Barcha dublikatlarni o‘chirish
  list <int> :: iterator it;
  for (it = mylist.begin(); it != mylist.end(); it++) {
    cout << (*it) << " ";
  }
    cout<<endl;
  mylist.merge(listmerge);  // Ro‘yxatni o‘zlashtirish
```

```
  for (it = mylist.begin(); it != mylist.end(); it++) {
    cout << (*it) << " ";
  }
  return 0;
}
```



10-rasm. Ro‘yxat bilan ishlash metodlari asosida olingan natijalar

Elementlarni o‘chirish. pop_front () va pop_end () usullaridan foydalanib, boshida va oxirida o‘chirishga qo‘shimcha ravishda, siz quyidagilarni o‘chirishingiz mumkin:

1) Yacheykalar diapazonini;
2) Ixtiyoriy yacheykani;
3) Qanday shart asosida biror yacheykani;
4) X qiymatiga ega bo‘lgan barcha yacheykalarni o‘chirib tashlash mumkin.

Mavzu yuzasidan savollar:

1. Ma’lumot strukturasi tushunchasi nimani anglatadi?

2. Abstrakt ma’lumotlar strukturasi haqida gapiring.

3. Stek ma’lumotlar strukturasiga doir misollar keltiring.

4. Navbat va ro‘yxat ma’lumotlar strukturasi bir-biridan qanday farq qiladi?

5. Vektor ma’lumotlar strukturasi uchun aniqlangan metodlarni ko‘rsating.

Mustaqil ishlash uchun masalalar:

1. 10 ta elementdan iborat stek hosil qiling. Stekning yuqori elementini o‘chirish metodidan foydalaning

2. 100 ta elementdan iborat stek hosil qiling. Ushbu stekka yana 100 ta element qo‘shing

3. 100 ta elementdan iborat stek hosil qiling. Stek metodlaridan foydalanib amallar bajaring

4. 20 ta elementdan iborat Navbat hosil qiling. Uning 10 ta elementini o‘chiring. Uning oxirgi va birinchi elementlarni qo‘shing

5. 50 ta elementdan iborat Navbat hosil qiling. Uning birinchi elementini olib tashlash metodidan foydalaning

6. 20 ta elementdan iborat Navbat hosil qiling. Uning 10 ta elementini o‘chiring. Uning oxirgi va birinchi elementlarni qo‘shing

3-§. Saralash algoritmlari. Eng oddiy algoritmlar. Past baho

Mazkur mavzuda algoritmlarning yangi sinfi - saralash algoritmlarini o‘rganamiz. Ma’lumotlarni qayta ishlashda ma’lumotning axborot (**info**) maydonini bilish va uni mashinada joylashishini tasavvur qilish juda muhimdir.

Saralashning ichki va tashqi saralash turi mavjud:

1.ichki saralash - bu tezkor xotiradagi ma’lumotlarni saralash;

2. tashqi saralash - tashqi xotira (fayllar)dagi ma’lumotlarni saralash.

Saralash deganda ma’lumotlarni xotirada muayyan tartibda uning kalitlari bo‘yicha joylashtirish, muayyan tartib deganda esa kalit qiymati bo‘yicha o‘sish (yoki kamayish) tartibida ro‘yxatning boshidan oxirigacha joylashtirish tushuniladi.

Saralash algoritmlarining samaradorligini bir necha parametrlari bo‘yicha farqlash mumkin. Bu parametrlarning asosiylari quyidagilar hisoblanadi:

- saralash uchun sarflanadigan vaqt;
- saralash uchun talab qilinadigan tezkor xotira hajmi.

Saralash algoritmlarini baholashda faqat “joyida” saralash usullarini qarab chiqamiz, ya’ni saralash jarayoni uchun qo‘shimcha xotira zahirasi talab qilinmaydi. Saralash uchun sarf qilinadigan vaqtni esa, saralash bajarilishi jarayonida amalga oshiriladigan taqqoslashlar va o‘rin almashtirishlar soni orqali hisoblash mumkin. Ixtiyoriy saralash usulida taqqoslashlar soni $O(n\log_2 n)$ dan $O(n^2)$ gacha bo‘lgan oraliqda yotadi.

Ma’lumotlarni saralashning **qat’iy (to‘g‘ri) va yaxshilangan** usullari mavjud bo‘lib, qat’iy usullariga quyidagilarni misol qilib olish mumkin:

1) to‘g‘ridan-to‘g‘ri qo‘yish orqali saralash usuli;
2) to‘g‘ridan-to‘g‘ri tanlash orqali saralash usuli;
3) to‘g‘ridan-to‘g‘ri almashtirish orqali saralash usuli.

Bu uchala saralash usullarining samaradorligi deyarli bir xil.

Lug‘atlarda "saralash" (sorting) so‘zi "toifalarga ajratish, tartiblash, baholash" deb ta’riflanadi, ammo dasturchilar odatda bu so‘zni tor ma’noda ishlatishadi, ularga ba’zi bir aniq tartibda elementlarni qayta joylashtirishga murojaat qilishadi. Bu jarayonni, ehtimol, saralash deb emas, balki tartiblash (ordering) yoki ketma-ketlik (sequencing) deb atash kerak. Biroq, saralash so‘zi dasturlash jargonida allaqachon mustahkam o‘rnashgan, shu sababli kelajakda "saralash" so‘zini tor ma’noda "tartiblash" dan foydalanamiz. Bu shuni anglatadiki, endi "saralash" ta’rifini shakllantirishimiz mumkin, bu kelgusida ishlatiladi.

Tartiblash – bu berilgan obyektlar to‘plamini muayyan tartibda qayta tartibga solish jarayoni. Saralashning maqsadi elementlarni topishni osonlashtirishdir.

Saralash algoritmi – bu ro‘yxatdagi elementlarni saralash algoritmi. Agar ro‘yxat elementida bir nechta maydon bo‘lsa, saralash amalga oshiriladigan maydon **saralash kaliti** deb ataladi. Amalda raqam ko‘pincha kalit sifatida ishlatiladi va ba’zi ma’lumotlar algoritm ishlashiga hech qanday ta’sir ko‘rsatmaydigan qolgan maydonlarda saqlanadi.

Ehtimol, boshqa hech qanday muammo saralash muammosi kabi juda ko‘p turli xil yechimlarni keltirib chiqarmagan. Butun dunyoda tan olingan eng yaxshi algoritm bormi? Umuman aytganda, yo‘q. Biroq, kirish ma’lumotlarining taxminiy xususiyatlarini hisobga olgan holda, siz eng yaxshi ishlaydigan usulni tanlashingiz mumkin.

Saralash usullari juda ko‘p, ularning har biri o‘zining afzalliklari va kamchiliklariga ega. Tartiblash algoritmlari katta amaliy ahamiyatga ega, ular o‘zlari uchun qiziqarli. Bu juda chuqur o‘rganilgan informatika sohasi axborot qidirish tizimlarida, harbiy sohada va bank sohalarida ko‘proq qo‘llaniladi. Ammo hozirgi kunda axborot oqimini tartiblash masalasi deyarli har bir sohaga kirib bordi.

Algoritmlarni saralashga bo‘lgan umumiy ilmiy qiziqishdan tashqari, har bir algoritmda uning **murakkabligi** deb ataladigan narsani baholash qiziq. Murakkablik algoritmning boshlang‘ich bosqichlarining maksimal soni sifatida tushuniladi. Tartiblash misollari algoritmni murakkablashtirish orqali ko‘rsatilishi mumkin, garchi hozirda aniq

usullar mavjud boʻlsa-da, siz samaradorlikda sezilarli yutuqqa erishishingiz mumkin.

Massivlarni saralash masalasini yechishda odatda qoʻshimcha xotiradan foydalanishni minimallashtirish talabi qoʻyiladi, bu esa qoʻshimcha massivlardan foydalanishga yoʻl qoʻyilmasligini anglatadi.

Quyidagi koʻrsatkichlar ham muhimdir:

• **Xotira.** Bir qator algoritmlar ma’lumotlarni vaqtincha saqlash uchun qoʻshimcha xotira ajratishni talab qiladi. Amaldagi xotirani baholashda boshlangʻich massiv egallagan joy, kirish ketma-ketligidan mustaqil sarflangan xotira, masalan, dastur kodini saqlash uchun ajratilgan joy hisobga olinmaydi.

• **Turgʻunlik**. Doimiy saralash teng elementlarning nisbiy holatini oʻzgartirmaydi. Ushbu xususiyat juda foydali boʻlishi mumkin, agar ular bir nechta maydonlardan iborat boʻlsa va saralash ulardan biri tomonidan amalga oshirilsa.

Barcha saralash usullarini ikkita katta guruhga boʻlish mumkin:

• Saralashning bevosita usullari;

• Takomillashtirilgan saralash usullari;

Toʻgʻridan-toʻgʻri tartiblash usullari usul asosida yotadigan prinsipga koʻra, uchta kichik guruhga boʻlinadi:

• oddiy qoʻshimchalar boʻyicha saralash (qoʻshish);

• tanlash (saralash) boʻyicha saralash;

• Almashish boʻyicha saralash ("Pufakchali" saralash).

Takomillashtirilgan tartiblash usullari toʻgʻridan-toʻgʻri prinsiplarga asoslanadi, ammo saralash usulini tezlashtirish uchun ba’zi bir original gʻoyalardan foydalaniladi. Toʻgʻridan-toʻgʻri saralash usullari amalda kamdan kam qoʻllaniladi, chunki ular nisbatan past koʻrsatkichlarga ega. Biroq, ular shu usullar asosida kelib asoslangan takomillashtirilgan usullarning mohiyatini yanada yaxshi namoyish etadi. Bundan tashqari, ba’zi hollarda (odatda massivning uzunligi kichik boʻlsa yoki yoki massiv elementlarining boshlangʻich joylashuvi bilan) toʻgʻridan-toʻgʻri usullar takomillashtirilgan usullardan ham ustun boʻlishi mumkin.

3.1. Ichki saralash muammosining bayoni va samaradorlikni baholash yondashuvlari

Koʻpgina hollarda, ma’lumotlarning ba’zi bir mezonlarga muvofiq tartiblanishi ma’lumotlarni qayta ishlashni soddalashtiradi. Masalan, ikkilik qidiruvni ketma-ket qidirishni amalga oshirishda vaqtni sezilarli darajada tejash, ikkilik yoki boshqa turdagi qidiruv algoritmlarini amalga oshirishda katta yutuqlarni ta’minlash uchun ma’lumotlar toʻplamini oldindan saralashga vaqt sarflash uchun yetarli sababdir.

Eng muhimi, in situ (joylashtirish) boʻyicha tartiblash algoritmlari boʻlib, ular tartiblangan ketma-ketlikni egallagan xotira maydoni ichidagi elementlarning almashinishini ta’minlaydi. Bunday holda, ma’lumotlar ketma-ketligi odatda tezkor xotirada joylashgan massiv sifatida tushuniladi - bunday massivlarni saralash ichki saralash deb ataladi, aksincha tashqi saralash - ba’zi tashqi manbalardan ma’lumotlarni olish (masalan, diskdagi fayl) sifatida tushuniladi.

Odatda ma’lumotlar ba’zi bir muhim qiymatlarning koʻtarilish yoki kamayish tartibida saralanadi.

Algoritmni tahlil qilish ushbu algoritm yordamida muammoni hal qilish uchun qancha vaqt sarflanishi haqida tasavvurga ega boʻlishni oʻz ichiga oladi. Algoritmni baholashda, ma’lum bir algoritm uchun uning ishlashi davomida eng muhim boʻlgan amallar sonidan kelib chiqadi. Saralash algoritmlari uchun bunday amallar asosiy taqqoslash amallari va tartiblangan elementlarning uzatish amallari hisoblanadi.

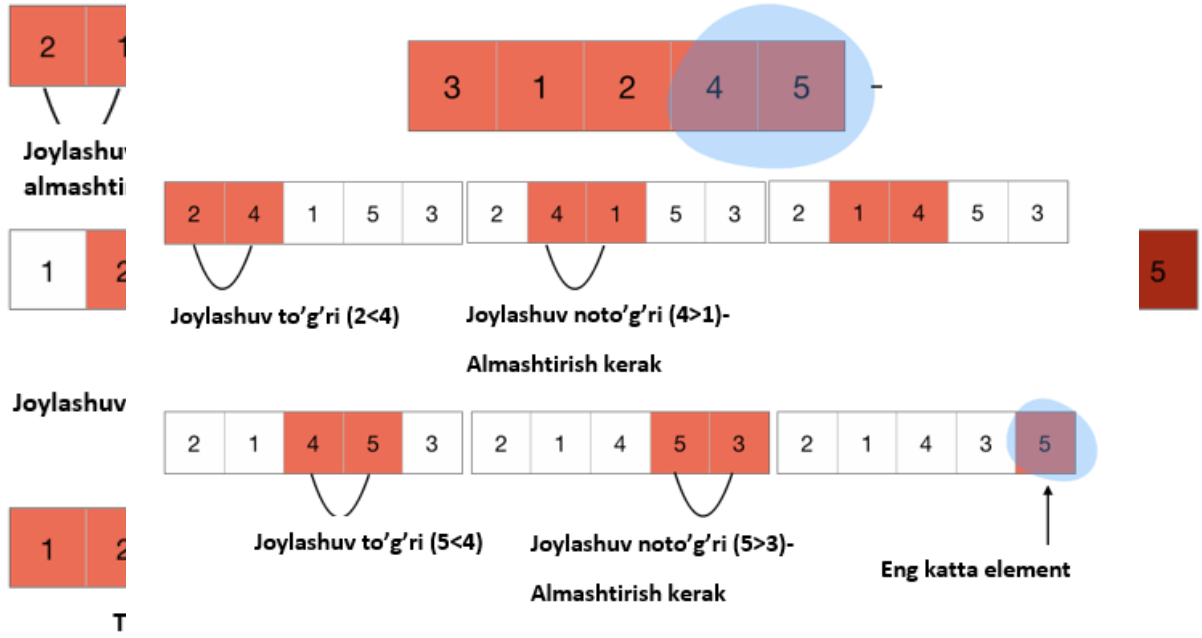
Algoritm samaradorligini baholashda, odatda, uchta variantni baholashga harakat qilinadi: eng yaxshi holat (vazifa eng qisqa vaqt ichida amalga oshirilganda), eng yomon holat (vazifa maksimal vaqt ichida amalga oshirilganda) va oʻrta holat , (buni odatda tahlil qilish eng qiyin). Algoritmlarni saralashning asosiy koʻrsatkichlari bu algoritm ishlashi davomida amalga oshirilgan taqqoslash va almashtirishlarning oʻrtacha soni.

Shu bilan birga, algoritm tomonidan bajariladigan amallar sonini aniq bilish samaradorlikni tahlil qilish nuqtai nazaridan juda muhim

emas. N - massiv elementlari sonining ko‘payishi bilan amallar sonining o‘sish sur’ati muhimroqdir.

3.2. Oddiy saralash algoritmlari va ularning tahlili

Pufakchali saralash (Bubble sort). Ushbu algoritmda massivni takrorlash bilan boshlaymiz va birinchi elementni ikkinchisiga taqqoslaymiz va agar ular noto‘g‘ri tartibda joylashgan bo‘lsa, ularni



11-rasm. Pufakchali saralash algoritmining ishlash prinsipi

almashtiramiz, keyin ikkinchi va uchinchisini taqqoslaymiz va hokazo. Ushbu takrorlashdan so‘ng eng katta element quyida keltirilgan rasmda ko‘rsatilgandek massivning oxiriga o‘tadi.

Endi eng katta element o‘z joyini egallaydi, shuning uchun biz yana ushbu jarayonni takrorlaymiz va allaqachon o‘z pozitsiyalarida bo‘lgan elementlarni qoldiramiz va bu tartiblangan massivni beradi.

Butun jarayonni quyidagi bosqichlarda yozishimiz mumkin:

1) Massiv bo‘yicha takrorlashni boshlash.

2) Qo‘shni elementlarni solishtirish. Masalan, birinchi va ikkinchi, ikkinchi va uchinchi va boshqalar.

3) Agar ular tartiblangan bo‘lmasa, ularni almashtirish.

4) To‘g‘ri pozitsiyalariga joylashtirilgan elementlardan tashqari, ushbu amallarni takrorlash.

Pufakchali saralash (Bubble sort) algoritmining C++ dastur kodi. Ushbu algoritm ta’limiy hisoblanadi va samaradorligi pastligi sababli amalda deyarli hech qachon qo‘llanilmaydi: u kichik elementlar (ular "toshbaqalar" deb nomlanadi) massiv oxirida joylashgan testlarda sekin ishlaydi. Biroq, ko‘plab boshqa usullar bunga asoslangan, masalan, Sheyker saralash va taroqsimon saralashlari.

Vaqt bo‘yicha murakkabligi:

Eng yomon baho: $O(n^2)$

O‘rtacha baho: $O(n^2)$

Eng yaxshi baho: O(n)

```
void bubbleSort(int arr[], int n)
{
  int i, j;
  bool swapped;
  for (i = 0; i < n-1; i++)
  {
    swapped = false;
    for (j = 0; j < n-i-1; j++)
    {
      if (arr[j] > arr[j+1])
      {
        swap(&arr[j], &arr[j+1]);
        swapped = true;
      }
    }

    if (swapped == false)
      break;
  }
}
```

Sheyker saralashi. Sheyker (kokteyl) saralashi - bu Pufakchali (Bubble Sort) algoritmining varianti. Bubble sort algoritmi har doim chapdan elementlarni aylanib o‘tadi va birinchi iteratsiyada eng katta elementni to‘g‘ri holatiga, ikkinchisida esa ikkinchi takrorlashda va

hokazo. Kokteyl saralash berilgan massiv orqali har ikki yo‘nalishda ham muqobil ravishda harakatlanadi.

Algoritmning har bir takrorlanishi 2 bosqichga bo‘linadi:

Birinchi bosqich massivni xuddi Bubble Sort singari chapdan o‘ngga aylantiradi. Sikl davomida qo‘shni elementlar taqqoslanadi va agar chapdagi qiymat o‘ngdagi qiymatdan katta bo‘lsa, qiymatlar almashtiriladi. Birinchi takrorlash oxirida eng katta son massivning oxirida bo‘ladi.

Ikkinchi bosqich massivni qarama-qarshi yo‘nalishda aylantiradi - bu eng so‘nggi saralangan elementdan oldin va massivning boshiga qaytish. Bu yerda qo‘shni elementlar taqqoslanadi va agar kerak bo‘lsa almashtiriladi.

Quyidagi massivni ko‘rib chiqaylik (6 1 4 2 8 0 2).

Birinchi oldinga o‘tish:

(6 1 4 2 8 0 2)? (1 6 4 2 8 0 2), 6> 1 bilan almashtirish
(1 6 4 2 8 0 2)? (1 4 6 2 8 0 2), 6> 4 bilan almashtirish
(1 4 6 2 8 0 2)? (1 4 2 6 8 0 2), 6> 2 bilan almashtirish
(1 4 2 6 8 0 2)? (1 4 2 6 8 0 2)
(1 4 2 6 8 0 2)? (1 4 2 6 0 8 2), 8> 0 bilan almashtirish
(1 4 2 6 0 8 2)? (1 4 2 6 0 2 8), 8> 2 bilan almashtirish

Birinchi oldinga o‘tishdan so‘ng, massivning eng katta elementi massivning oxirgi indeksida bo‘ladi.

Birinchi orqaga o‘tish:

(1 4 2 6 0 2 8)? (1 4 2 6 0 2 8)
(1 4 2 6 0 2 8)? (1 4 2 0 6 2 8), 6> 0 bilan almashtirish
(1 4 2 0 6 2 8)? (1 4 0 2 6 2 8), 2> 0 bilan almashtirish
(1 4 0 2 6 2 8)? (1 0 4 2 6 2 8), 4> 0 bilan almashtirish
(1 0 4 2 6 2 8)? (0 1 4 2 6 2 8), 1> 0 bilan almashtirish

Birinchi orqaga uzatgandan so‘ng, massivning eng kichik elementi massivning birinchi indeksida bo‘ladi.

Ikkinchi oldinga o‘tish:
(0 1 4 2 6 2 8)? (0 1 4 2 6 2 8)
(0 1 4 2 6 2 8)? (0 1 2 4 6 2 8), 4>2 bilan almashtirish
(0 1 2 4 6 2 8)? (0 1 2 4 6 2 8)
(0 1 2 4 6 2 8)? (0 1 2 4 2 6 8), 6>2 bilan almashtirish

Ikkinchi orqaga o‘tish:

(0 1 2 4 2 6 8)? (0 1 2 2 4 6 8), 4>2 bilan almashtirish

Endi, massiv allaqachon saralangan, ammo bizning algoritmimiz tugallanganligini bilmaydi. Algoritm bu saralanganligini bilish uchun barcha o‘tishlarni hech qanday almashtirishsiz bajarishi kerak.

(0 1 2 2 4 6 8) ? (0 1 2 2 4 6 8)
(0 1 2 2 4 6 8) ? (0 1 2 2 4 6 8)

Quyida yuqoridagi algoritmning bajarilishi keltirilgan:

```
void CocktailSort(int a[], int n)
{
   bool swapped = true;
   int start = 0;
   int end = n - 1;

   while (swapped)
   {
      swapped = false;

      for (int i = start; i < end; ++i)
      {
         if (a[i] > a[i + 1]) {
            swap(a[i], a[i + 1]);
            swapped = true;
         }
      }
```

```
    if (!swapped)
       break;

    swapped = false;

    --end;

    for (int i = end - 1; i >= start; --i)
    {
       if (a[i] > a[i + 1]) {
          swap(a[i], a[i + 1]);
          swapped = true;
       }
    }
    ++start;
} }
```

Vaqt bo‘yicha murakkabligi:
Eng yomon baho: $O(n^2)$
O‘rtacha baho: $O(n^2)$
Eng yaxshi baho: O(n)

3. Taroqsimon saralash. Comb sort. Taroqsimon saralash – “Pufakchali” saralashning yaxshiroq varianti. Uning g‘oyasi algoritmni sekinlashtiradigan qator oxiridagi kichik qiymatlarga ega elementlarni "yo‘q qilish". Agar pufakchali va Sheyker saralashlarida, massiv bo‘ylab takrorlanganda qo‘shni elementlar taqqoslansa, u holda "tarash" paytida avval taqqoslangan qiymatlar orasida yetarlicha katta masofa olinadi, so‘ngra u minimal darajaga tushadi.

Dastlabki bo‘shliq tasodifiy tanlanmasligi kerak, lekin maxsus qiymatni hisobga olgan holda - kamaytirish qiymati, uning optimal qiymati 1,247 ga teng. Dastlab elementlar orasidagi masofa massivning kattaligiga 1,247 ga teng bo‘ladi; har bir keyingi bosqichda masofa yana qisqartirish koeffitsiyentiga bo‘linadi - va shunga o‘xshash jarayon algoritm oxirigacha davom etadi.

Vaqt bo‘yicha murakkabligi:
Eng yomon baho: $O(n^2)$
O‘rtacha baho: $\Omega(n^2/2^p)$, p – inkrementlar miqdori
Eng yaxshi baho: O(n logn)

```
void combSort(int a[], int n)
{
   int k = n;

   bool swapped = true;

   while (k != 1 || swapped == true)
   {
      k = getNextk(gap);

      swapped = false;

      for (int i=0; i<n-k; i++)
      {
         if (a[i] > a[i+k])
         {
            swap(a[i], a[i+k]);
            swapped = true;
         }
      }
   }
}
```

4. Joylashtirish bo‘yicha saralash (Insertion sort). Joylashtirish bo‘yicha saralashda massiv asta-sekin chapdan o‘ngga takrorlanadi. Bunday holda, har bir keyingi element minimal va maksimal qiymatga ega bo‘lgan eng yaqin elementlar orasida bo‘lishi uchun joylashtiriladi.

Vaqt bo‘yicha murakkabligi:
Eng yomon baho: $O(n^2)$
O‘rtacha baho: $O(n^2)$,

Eng yaxshi baho: O(n)

```
void insertionSort(int arr[], int n)
{
   int i, key, j;
   for (i = 1; i < n; i++)
   {
      key = arr[i];
      j = i - 1;


      while (j >= 0 && arr[j] > key)
      {
         arr[j + 1] = arr[j];
         j = j - 1;
      }
      arr[j + 1] = key;
   }
}
```

5. Tanlash bo‘yicha saralash (Selection sort). Birinchidan, siz massivning kichik qismini ko‘rib chiqishingiz va undagi maksimal (yoki minimal) miqdorni topishingiz kerak. Keyin tanlangan qiymat birinchi saralanmagan elementning qiymati bilan almashtiriladi. Ushbu qadam massivning saralanmagan ichki qismlari tugamaguncha takrorlanishi kerak.

Vaqt bo‘yicha murakkabligi:

Eng yomon baho: $O(n^2)$

O‘rtacha baho: $O(n^2)$,

Eng yaxshi baho: $O(n^2)$

```
void selectionSort(int arr[], int n)
{
   int i, j, min_idx;

   for (i = 0; i < n-1; i++)
```

```
    {
        min_idx = i;
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;

        swap(&arr[min_idx], &arr[i]);
    }
}
```

Ushbu algoritmlarning elementlari soni bir xil bo‘lgan holatda qanday vaqt ichida bajarilishi va sarflangan xotira hajmi haqidagi qiyosiy jadval quyida berilgan.

Sinov o‘tkaziladigan kompyuter quyidagi xususiyatlarga ega: AMD A6-3400M 4x1.4 GHz, 8 Gb operativ xotira, Windows 10 x64 build 10586.36.

2-jadval.

Turli saralash algortimlarining to‘liq saralanmagan massiv uchun ishlash vaqti va ishlatilgan xotira sig‘imi

Saralash usuli		10 ta element uchun		50 ta element uchun		200 ta element uchun		1000 ta element uchun	
		Vaqt (ms)	Xotira (K)	Vaqt (ms)	Xotira (K)	Vaqt (ms)	Xotira (K)	Vaqt (ms)	Xotira (K)
Tanlash bo‘yicha saralash	**Selection sort**	13	510K	37	637	118	854	550	936
Pufakchali saralash	**Bubble sort**	11	524	37	629	116	863	564	932
Joylashtirish bo‘yicha saralash	**Insertion sort**	12	512	38	641	116	849	556	928
Taroqsimon saralash	**Comb sort**	12	505	37	632	117	854	560	936

Qisman tartiblangan massiv (elementlarning yarmi saralangan):

3-jadval.

Turli saralash algortimlarining qisman saralangan massiv uchun ishlash vaqti va ishlatilgan xotira sig‘imi

Saralash usuli		10 ta element uchun		50 ta element uchun		200 ta element uchun		1000 ta element uchun	
		Vaqt (ms)	Xotira (K)	Vaqt (ms)	Xotira (K)	Vaqt (ms)	Xotira (K)	Vaqt (ms)	Xotira (K)
Tanlash bo‘yicha saralash	**Selection sort**	10	501	32	612	113	823	498	942
Pufakchali saralash	**Bubble sort**	9	498	32	601	110	812	509	939
Joylashtirish bo‘yicha saralash	**Insertion sort**	9	482	31	597	112	802	502	920
Taroqsimon saralash	**Comb sort**	10	498	33	563	116	601	505	907

Mavzu yuzasidan savollar:

1. Tartiblash va saralash tushunchalariga ta’rif bering
2. Tanlash bo‘yicha saralash algoritmining murakkabligini baholang
3. Taroqsimon saralash va pufakchali saralash o‘rtasidagi o‘xshashliklarni keltiring
4. Saralash algoritmlari qanday yondashuvlar asosida baholanadi?
5. Yuqorida keltirilganlardan tashqari yana qanday saralash algoritmlarini bilasiz?

4-§ Birlashtirib saralash algoritmlari

Merge sort algoritmi. Birlashtirib saralash (Merge sort) – tartiblashning tezkor bajariladigan algoritmlaridan biri. Ushbu tartiblash “bo‘lib tashla va hukmronlik qil” prinsipining yaxshi namunasidir. Birinchidan, vazifa bir nechta kichik topshiriqlarga bo‘linadi. Keyin ushbu vazifalar rekursiv chaqiruv yordamida yoki to‘g‘ridan-to‘g‘ri ularning hajmi yetarlicha kichik bo‘lsa hal qilinadi. Nihoyat, ularning yechimlari birlashtirilib, asl muammoning echimi olinadi.

Algoritmning bajarilishi. Saralash muammosini hal qilish uchun uch bosqich quyidagicha bo‘ladi:

1. Saralanadigan massiv taxminan bir xil o‘lchamdagi ikki qismga bo‘linadi;
2. Olingan qismlarning har biri alohida saralanadi (masalan, xuddi shu algoritm bo‘yicha saralanadi);
3. Yarim kattalikdagi ikkita saralangan massivlar birlashtiriladi.

Bu eng mashhur saralash algoritmlaridan biri bo‘lib, rekursiv algoritmlarni yaratishda ishonchni rivojlantirishning ajoyib usuli hisoblanadi.

“Bo‘lib tashla va hukmronlik qil” strategiyasi. “Bo‘lib tashla va hukmronlik qil” strategiyasi yordamida muammoni qismiy jarayonlarga ajratamiz. Har bir kichik topshiriq uchun yechimga ega bo‘lsak, pastki vazifalarni yechish uchun pastki vazifalardan olingan natijalarni "birlashtiramiz".

Aytaylik, biz A massivni saralashni xohladik. Kichik vazifa bu p indeksidan boshlanib, r indeksida tugagan, A [p..r] bilan belgilangan kichik qismini ajratishdir.

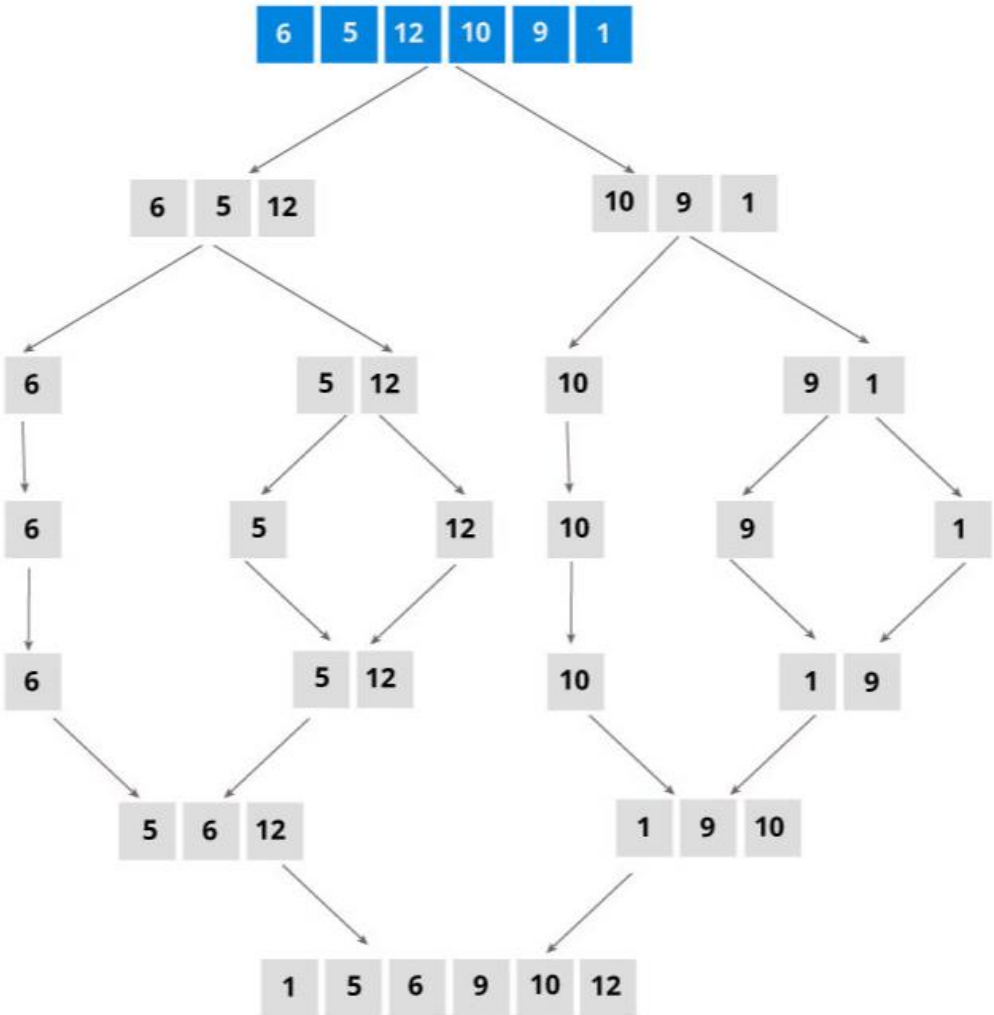
“Bo‘lib tashlash”. Agar q qiymati p va r orasida bo‘lsa, biz A [p..r] massivni ikkita A [p..q] va A [q + 1, r] kichik massivlarga bo‘lishimiz mumkin.

“Hukmronlik qilish”. “Hukmronlik qilish” bosqichida biz ikkala A [p..q] va A [q + 1, r] kichik massivlarni saralashga harakat qilamiz. Agar hali ham boshlang‘ich darajaga yetib bormagan bo‘lsak, yana ikkala quyi qismni ajratib, ularni saralashga harakat qilamiz.

Birlashtirish bosqichi. Birlashtirish bosqichi asosiy pog‘onaga yetib borganida va biz A [p..r] massivi uchun ikkita tartiblangan A [p..q] va A [q + 1, r] kichik massivlarni olsak, natijalarni A [p..r] massiviga birlashtiramiz. Bu ikkita tartiblangan A [p..q] va A [q + 1, r] massivlarning birlashmasidir (12-rasm).

Birlashtirib saralash algoritmi. MergeSort funksiyasi massivni ketma-ket ikki qismga ajratadi, biz 1-darajali ichki massivda MergeSort-ga o‘tishga harakat qiladigan bosqichga yetguncha ya’ni p == r bo‘lguncha.

Shundan so‘ng, birlashtirish funksiyasi ishga tushadi, bu tartiblangan massivlarni butun massiv birlashguncha katta massivlarga birlashtiradi.

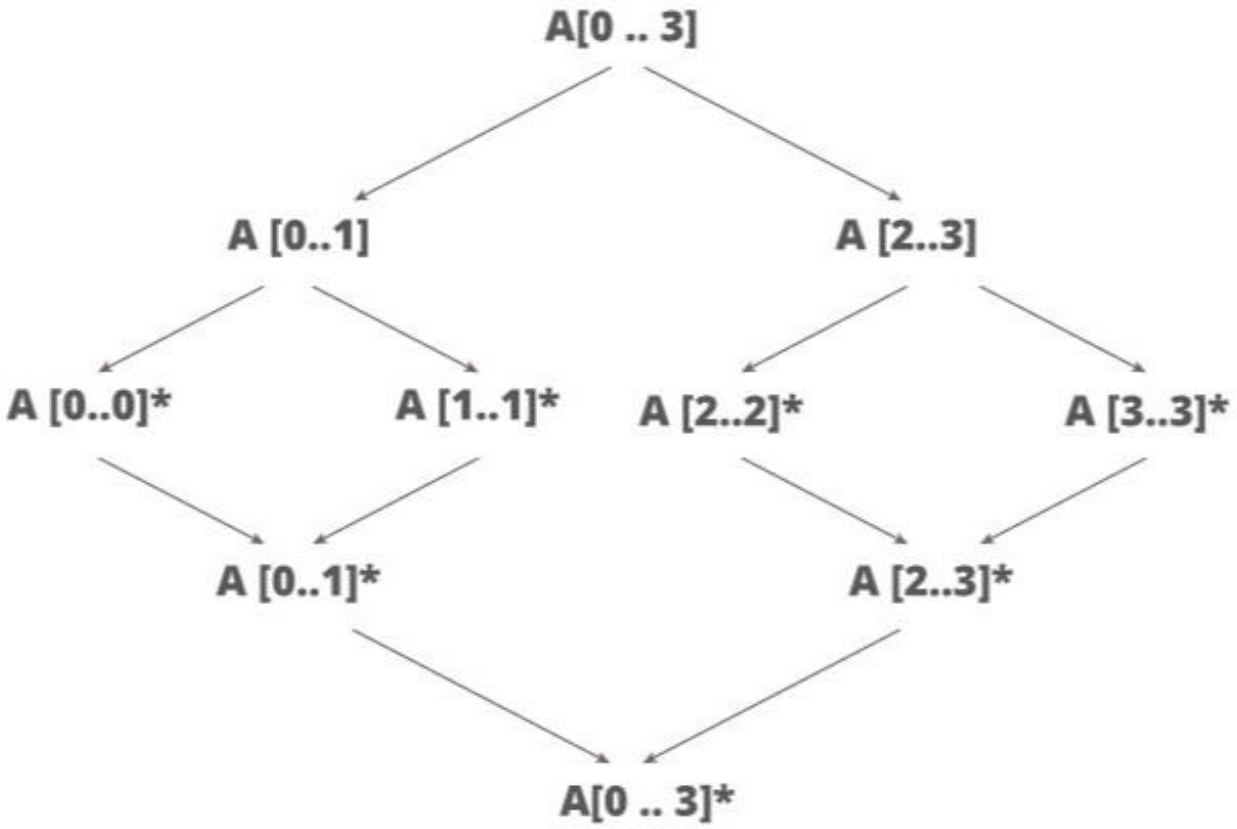


12-rasm. Merge Sort algoritmining ishlash prinsipi

```
1. MergeSort(A, p, r)
2.     If p > r
3.         return;
4.     q = (p+r)/2;
5.     mergeSort(A, p, q)
6.     mergeSort(A, q+1, r)
7.     merge(A, p, q, r)
```

Butun massivni saralash uchun MergeSort (A, 0, length (A) -1) ga murojaat qilishimiz kerak.

13-rasmda ko‘rsatilgandek, birlashtirib saralash algoritmi 1 elementli massivning asosiy holatiga kelgunimizcha massivni rekursiv ravishda ikkiga bo‘ladi. So‘ngra birlashtirish funksiyasi saralangan ichki massivlarni tanlaydi va butun qatorni asta-sekin saralash uchun ularni birlashtiradi.



13-rasm. Merge Sort algoritmida massivni qismlarga bo‘lish jarayoni

Algoritmning eng muhim qismi bu "birlashtirish" bosqichidir. Birlashish bosqichi - ikkita katta ro‘yxat (massiv) yaratish uchun ikkita tartiblangan ro‘yxatni (massivlarni) birlashtirish bo‘yicha oddiy muammoning yechimi.

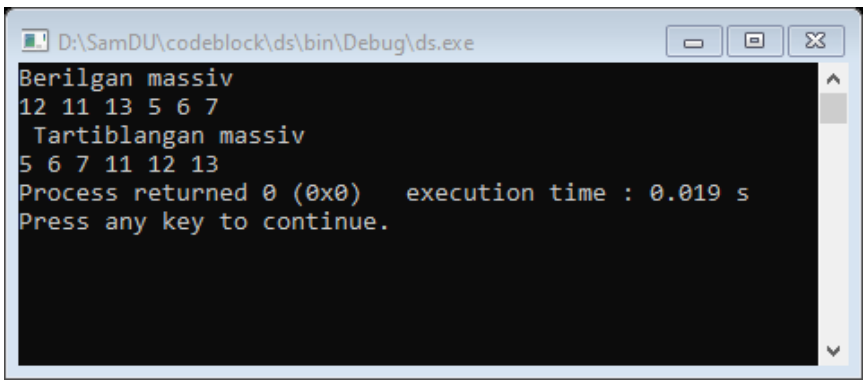
Ikkinchi massivda boshqa elementlar qolmaganligi va ikkala massiv ham ishga tushirilganda saralanganligini bilganimiz uchun qolgan massivlarni to‘g‘ridan-to‘g‘ri birinchi massivdan nusxalashimiz mumkin.

Birlashtirib saralash algoritmi uchun dastur kodi

```
#include <iostream>
using namespace std;

// Array[] ikkita ichki massivni birlashtiradi.
//Birinchi ichki massiv - Array[l..m]
//  Ikkinchi ichki massiv Array[m+1..r]
void merge(int Array[], int l, int m, int r)
{
        int n1 = m - l + 1;
        int n2 = r - m;

        // Vaqtinchalik massivlarni yaratish
        int L[n1], R[n2];

        // Ma’lumotlarni vaqtinchalik L[] va R[] massivlariga nusxalash
        for (int i = 0; i < n1; i++)
                L[i] = Array[l + i];
        for (int j = 0; j < n2; j++)
                R[j] = Array[m + 1 + j];

        // Vaqtinchalik massivlarni yana arr [l..r] ga birlashtirish.

        // Birinchi ichki massivning boshlang‘ich ko‘rsatkichi
        int i = 0;
```

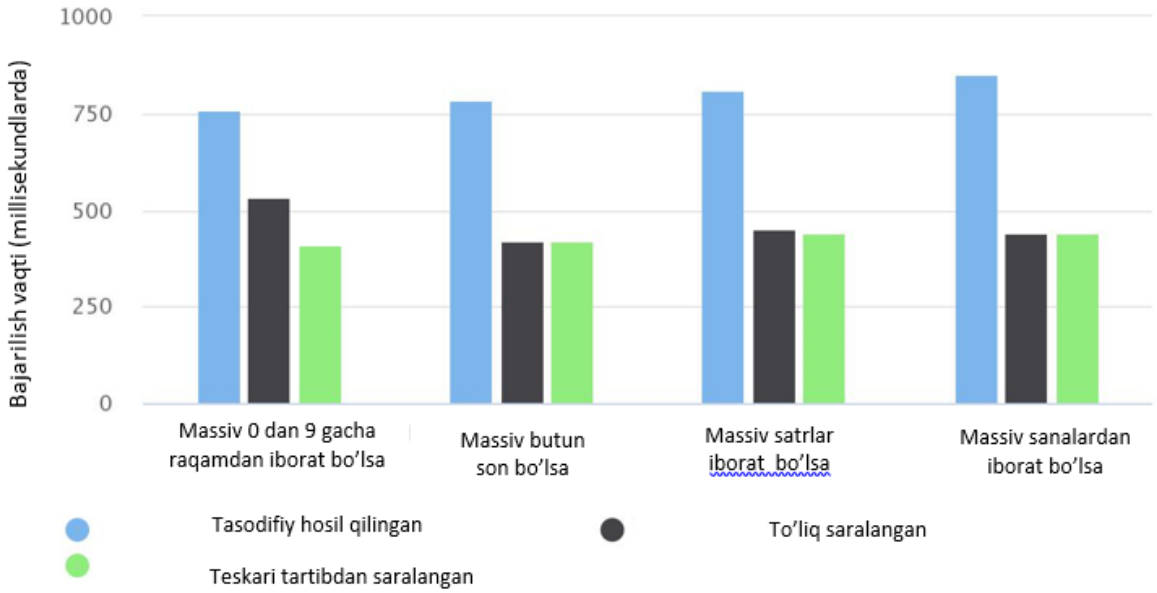
```
        // Ikkinchi kichik massivning boshlang‘ich ko‘rsatkichi
        int j = 0;

        // Birlashtirilgan ichki massivning dastlabki ko‘rsatkichi
        int k = l;

        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                Array[k] = L[i];
                i++;
            }
            else {
                Array[k] = R[j];
                j++;
            }
            k++;
        }

        // L [] ning qolgan elementlarini nusxalash,
        //agar mavjud bo‘lsa
        while (i < n1) {
            Array[k] = L[i];
            i++;
            k++;
        }

        // Agar mavjud bo‘lsa, R [] ning
        //qolgan elementlarini nusxalash
        while (j < n2) {
            Array[k] = R[j];
            j++;
            k++;
        }
}

// l chap indeks uchun,
//r esa tartiblangan ichki massivning o‘ng indeksidir
void mergeSort(int Array[],int l,int r){
```

```
        if(l>=r){
                return;//rekursiv ravishda qaytadi
        }
        int m =l+ (r-l)/2;
        mergeSort(Array,l,m);
        mergeSort(Array,m+1,r);
        merge(Array,l,m,r);
}

// Massivni chop etish funksiyasi
void printArrayay(int A[], int size)
{
        for (int i = 0; i < size; i++)
                cout << A[i] << " ";
}
int main()
{
        int Array[] = { 12, 11, 13, 5, 6, 7 };
        int Array_size = sizeof(Array) / sizeof(Array[0]);

        cout << "Berilgan massiv \n";
        printArrayay(Array, Array_size);

        mergeSort(Array, 0, Array_size - 1);

        cout << "\n Tartiblangan massiv \n";
        printArrayay(Array, Array_size);
        return 0;
}
```



14-rasm. Merge Sort algoritmi dastur natijasi

Birlashtirib saralash algortimini baholash. Algoritmning murakkabligini taxmin qilaylik. Har qanday rekursiv funksiya chaqiruvi daraxtga o‘xshaydi (Izoh: “Daraxtlar” haqida keyingi ma’ruzalarda to‘xtalib o‘tiladi). Bunday daraxtni rekursion daraxt deb ham atashadi. Daraxtning har bir darajasi bir yoki bir nechta funksiya chaqiruvlarini aks ettiradi. Shoxlari bo‘lmagan daraxt tugunlari rekursiyani tugatadigan funksiya chaqiruvlarini anglatadi. Birlashtirish tartibida daraxtning balandligi $\log_2 n$ ga teng, chunki har bir qadamda boshlang‘ich massiv n/2 uzunlikdagi ikkita ichki massivga bo‘linadi. Ajratishdan so‘ng, birlashtirish operatsiyasi amalga oshiriladi. Birlashtirish jarayoni n taqqoslashni, navbati bilan *log*n marta, ya’ni daraxtning har bir darajasi uchun 1 marta takrorlashni talab qiladi. Keyin algortim asimptotikasi O (nlogn) bo‘ladi.



15-rasm. Merge Sort algoritmining turli xil tiplar uchun ishlash vaqti (elementlar soni 50000 ta)

1. Merge sort algoritmining murakkabliklarini baholang
2. Merge sort algoritmida “Bo‘lib tashlash”da nimalarga e’tibor berish kerak

3. Birlashtirish bosqichi qanday amalga oshiriladi?

5-§. Tez saralash algoritmi

Tezkor saralash (Quick Sort). Saralashning eng yaxshi algoritmlari o‘nligi tuzilganda, ko‘plab dasturchilar ro‘yxati orasida Tezkor saralash (Quick Sort) algoritmini ko‘rishimiz mumkin. O‘tgan mavzuda saralash algoritmlarining eng yaxshilaridan biri sifatida birlashtirib saralash (Merge Sort) algoritmni ko‘rib chiqqandik. Shuning uchun Quick Sort algoritmining qanday afzalliklari mavjud, degan tabiiy savol paydo bo‘ladi?

Amaliy nuqtai nazardan Quicksort algoritmi raqobatbardosh bo‘lib, ko‘pincha MergeSort algoritmidan ustun turadi va shu sababli bu ko‘plab dasturlash kutubxonalarida standart tartiblash usuli hisoblanadi.

Quicksort algoritmining MergeSort algoritmidan katta ustunligi shundaki, u bir joyda ishlaydi - u kirish massivi bilan faqat elementlarning juft to‘g‘ridan-to‘g‘ri almashinuvini takrorlash orqali ishlaydi va shu sababli oraliq uchun faqat ozgina qo‘shimcha tezkor xotira kerak bo‘ladi.

Tezkor saralash (Quick sort – Xoara metodi) ko‘pincha qsort deb nomlanadi (uning nomi C standart kutubxonasida) - bu ingliz kompyuter olimi Toni Xoara tomonidan 1960-yilda Moskva davlat universitetida ishlab yurgan paytlarida yaratilgan saralash algoritmi hisoblanadi.

Massivlarni saralash bo‘yicha eng tez ma’lum bo‘lgan universal algoritmlardan biri: n elementni saralashda o‘rtacha O (nlogn) almashinuv bo‘ladi. Bir qator kamchiliklar mavjudligi sababli amalda odatda ba’zi bir o‘zgartirishlar bilan qo‘llaniladi.

Umumiy tavsif. QuickSort - bu to‘g‘ridan-to‘g‘ri almashinuvni saralash algoritmining (Bubble Sort va Shaker Sort algoritmlari) sezilarli darajada takomillashtirilgan variant bo‘lib, u past samaradorligi bilan ham tanilgan. Asosiy farq shundaki, birinchi navbatda, almashtirishlar mumkin bo‘lgan masofada amalga oshiriladi va har bir o‘tishdan keyin elementlar ikkita mustaqil guruhga bo‘linadi. (Shunday qilib, eng samarasiz to‘g‘ridan-to‘g‘ri saralash usulini takomillashtirish eng samarali takomillashtirilgan usullardan birini beradi.)

Algoritmning umumiy g‘oyasi quyidagicha:

1) Massivdan “tayanch” elementni tanlang. Bu massivdagi har qanday element bo‘lishi mumkin. Algoritmning to‘g‘riligi “tayanch” elementini tanlashga bog‘liq emas, lekin ba’zi hollarda uning samaradorligi kuchli bog‘liq bo‘lishi mumkin (pastga qarang).
2) Qolgan barcha elementlarni “tayanch” elementi bilan taqqoslang va ularni massiv ichida tartiblang, shunday qilib massivni ketma-ket uchta doimiy segmentga bo‘ling: "tayanch elementdan *kichikroq* elementlar, "tayanch elementga teng elementlar" va "tayanch elementdan katta elementlar".
3) "Kichik" va "katta" qiymatlar segmentlari uchun segmentning uzunligi birdan katta bo‘lsa, bir xil amallar ketma-ketligini bajaring.

Amalda massiv odatda uchga emas, balki ikki qismga bo‘linadi: masalan, "tayanch elementdan kichikroq" va "tayanch elementga teng va katta". Bu yondashuv odatda yanada samaraliroq bo‘ladi, chunki bu qismlarni ajratish algoritmini soddalashtiradi (pastga qarang).

Xoara bu usulni mashinada tarjima qilish uchun ishlab chiqdi; lug‘at magnit lentada saqlangan va qayta ishlangan matn so‘zlarini saralash, ularni tarjima qilmasdan, lentaning bir qatorida olish imkoniyatini yaratdi.

Algoritmni Xoara Sovet Ittifoqida bo‘lganida ixtiro qilgan, u yerda u Moskva universitetida kompyuter tarjimasini o‘rgangan va ruscha-inglizcha so‘zlashuv kitobini ishlab chiqishda ishlagan.

Saralashning umumiy mexanizmi. Quicksort – bu ham “bo‘lib tashla va hukmronlik qil” prinsipiga asoslanuvchi algoritmdir.

Eng umumiy ko‘rinishida psevdokod algoritmi quyida berilgan. (bu yerda A - saralanadigan massiv, low va high esa, mos ravishda, ushbu massivning saralangan qismining pastki va yuqori chegaralari)

Psevdokod nima? **Psevdokod** - bu imperativ dasturlash tillarining kalit so‘zlaridan foydalanadigan algoritmlarni tavsiflash uchun ixcham, ko‘pincha norasmiy til, ammo algoritmni tushunish uchun zarur bo‘lmagan tafsilotlar va o‘ziga xos sintaksisni chiqarib tashlaydi.

Algoritmni kompyuterga tarqatish va dasturni keyinchalik bajarish uchun emas, balki odamga taqdim etish uchun mo‘ljallangan.

Rekursiv QuickSort funksiyasi uchun psevdokod:

```
/* low  --> boshlang‘ich index,  high  --> yuqori index */
quickSort(arr[], low, high)
{
   if (low < high)
   {
       / * pi - bu qismlarni ajratish ko‘rsatkichi, arr [pi] endi kerakli joyda *
       /
pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Pi oldin
       quickSort(arr, pi + 1, high); // pi keyin
   }
}
```

“Bo‘lib tashlash” algoritmi. “Bo‘lib tashlash”ni amalga oshirishning ko‘plab usullari bo‘lishi mumkin, psevdokoddan so‘ng quyidagi algoritm qo‘llaniladi. Mantiqan sodda, biz eng chap elementdan boshlaymiz va kichik (yoki teng) elementlarning indeksini i sifatida kuzatamiz. Tekshirish paytida kichik element topsak, joriy elementni arr [i] bilan almashtiramiz. Aks holda biz joriy elementni e’tiborsiz qoldiramiz.

```
quickSort(arr[], low, high)
{
   if (low < high)
   {

      pi = partition(arr, low, high);

      quickSort(arr, low, pi - 1);  // Before pi
      quickSort(arr, pi + 1, high); // After pi
   }
```

}

“Bo‘lib tashlash” algoritmning psevdokodi. Ushbu funksiya so‘nggi elementni “tayanch” sifatida qabul qiladi, “tayanch” elementni tartiblangan qatorga to‘g‘ri holatiga qo‘yadi va kichikroq (burilishdan kichikroq) burilishning chap tomoniga va barcha katta elementlarni “tayanch element” ning o‘ng tomoniga joylashtiradi

```
partition (arr[], low, high)
{
   // pivot (Element to‘g‘ri joyga joylashtiriladi)
   pivot = arr[high];

   i = (low - 1)  // Kichikroq element ko‘rsatkichi va tayanch
                  //to‘g‘ri holatini ko‘rsatadi
  for (j = low; j <= high- 1; j++)
   {
      // Agar joriy element “tayanch” elementdan kichikroq bo‘lsa
      if (arr[j] < pivot)
      {
         i++;    // kichik elementning o‘sish ko‘rsatkichi
         swap arr[i] and arr[j]
      }
   }
   swap arr[i + 1] and arr[high])
   return (i + 1)
}
```

“Bo‘lib tashlash” algoritmining ishlashini quyidagi misolda qarab chiqish mumkin:

arr[] = {10, 80, 30, 90, 40, 50, 70}
Indekslar: 0   1   2   3   4   5   6

low = 0, high = 6, pivot = arr[h] = 70
Kichik element indeksini initsializatsiya qilish, **i = -1**

j = low to high-1
j = 0 : arr[j] <= pivot, shart bajarilsa, i++ va swap(arr[i], arr[j])
i = 0
arr[] = {10, 80, 30, 90, 40, 50, 70} //Massivda o‘zgarish bo‘lmaydi

j = 1 : arr[j] > pivot, bajarilsa, hech nima o‘zgarmaydi
// i va arr [] da o‘zgarish yo‘q

j = 2 : arr[j] <= pivot, shart bajarilsa i++ va swap(arr[i], arr[j])
i = 1
arr[] = {10, *30*, *80*, 90, 40, 50, 70} // 80 va 30 almashdi

j = 3 : arr[j] > pivot, shart bajarilsa, hech nima o‘zgarmaydi
// No change in i and arr[]

j = 4 : arr[j] <= pivot, shart bajarilsa i++ va swap(arr[i], arr[j])

i = 2
arr[] = {10, 30, 40, 90, 80, 50, 70} // 80 va 40 almashadi
j = 5 : arr[j] <= pivot, shart bajarilsa i++ va swap(arr[i], arr[j])
i = 3
arr[] = {10, 30, 40, 50, 80, 90, 70} // 90 va 50 almashadi

So‘nggi natija
arr[i+1] va arr[high]
arr[] = {10, 30, 40, 50, 70, 90, 80} // 80 va 70 almashtiriladi

“Tayanch element” hisoblanuvchi 70 ham o‘z o‘rnida. Undan kichik elementdan boshida, kattalari esa undan tepada

Quick sort algoritmning umumiy dasturi (C++)

```
#include <bits/stdc++.h>
using namespace std;
```

```
// Ikki elementni almashtirish uchun yordamchi funksiya
void swap(int* a, int* b)
{
	int t = *a;
	*a = *b;
	*b = t;
}

/*Ushbu funksiya
so‘nggi elementni “tayanch” sifatida qabul qiladi,
“tayanch” elementni tartiblangan qatorga to‘g‘ri holatiga qo‘yadi
va kichikroq (burilishdan kichikroq) burilishning chap tomoniga
va barcha katta elementlarni “tayanch element” ning o‘ng tomoniga joylashtiradi
*/

int partition (int arr[], int low, int high)
{
	int pivot = arr[high]; // tayanch element
	int i = (low - 1); // Kichikroq element ko‘rsatkichi va tayanch
		//to‘g‘ri holatini ko‘rsatadi

	for (int j = low; j <= high - 1; j++)
	{
		//Agar joriy element tayanchdan kichik bo‘lsa
		if (arr[j] < pivot)
		{
			i++;
			swap(&arr[i], &arr[j]);
		}
	}
	swap(&arr[i + 1], &arr[high]);
	return (i + 1);
}

void quickSort(int arr[], int low, int high)
{
	if (low < high)
```

```
        {

                int pi = partition(arr, low, high);


                quickSort(arr, low, pi - 1);
                quickSort(arr, pi + 1, high);
        }
}

void printArray(int arr[], int size)
{
        int i;
        for (i = 0; i < size; i++)
                cout << arr[i] << " ";
        cout << endl;
}


int main()
{
        int arr[] = {10, 7, 8, 9, 1, 5};
        int n = sizeof(arr) / sizeof(arr[0]);
        quickSort(arr, 0, n - 1);
        cout << "Saralangan massiv: \n";
        printArray(arr, n);
        return 0;
}
```

QuickSort algoritmi tahlili. Massivni „tayanch“ elementiga nisbatan ikki qismga bo‘lish jarayoni $O(\log_2 n)$ vaqtni oladi. Bir xil rekursiya darajasi bajariladigan barcha bo‘linish amallari hajmi doimiy bo‘lgan boshlang‘ich massivning turli qismlarini qayta ishlagani uchun, har bir rekursiya darajasida jami O (n) amallar ham talab qilinadi. Shuning uchun algoritmning umumiy murakkabligi faqat bo‘linishlar soni, ya’ni rekursiya darajasi bilan belgilanadi. Rekursiyaning darajsi,

o‘z navbatida, kirishlarning kombinatsiyasiga va “tayanch element” qanday aniqlanishiga bog‘liq.

Eng yaxshi holat. Eng yaxshi holatda har bir bo‘linish paytida massiv ikkita bir xil (+/- bitta element) qismlarga bo‘linadi, shuning uchun qayta ishlangan ichki massivlarning o‘lchamlari 1 ga yetadigan maksimal rekursiya darajasi $\log_2 n$ bo‘ladi. Natijada, quicksort tomonidan taqqoslash soni $O(n\log_2(n))$ algoritmining umumiy murakkabligini beradigan $C_n = 2C_{n/2} + n$ rekursiv ifodasining qiymatiga teng bo‘ladi.

O‘rtacha holat. Kirish ma’lumotlarini tasodifiy taqsimlash uchun o‘rtacha murakkablik faqat ehtimollik bilan baholanishi mumkin.

Avvalo shuni ta’kidlash kerakki, aslida, “tayanch” elementi har safar massivni ikkita teng qismga ajratishi shart emas. Masalan, agar har bir bosqichda dastlabki massivning 75% va 25% uzunlikdagi massivlarga bo‘linish bo‘lsa, rekursiya darajasi $log_{\frac{4}{3}} n$ ga teng bo‘ladi va O (nlogn) murakkablikni beradi. Umuman olganda, “tayanch” elementning chap va o‘ng tomonlari orasidagi har qanday aniq nisbatlar uchun algoritmning murakkabligi bir xil bo‘ladi, faqat har xil konstantalar mavjud.

Yomon holat. Eng yomon holatda har bir “tayanch” 1 va n-1 o‘lchamdagi ikkita kichik massivni beradi, ya’ni har bir rekursiv chaqiriq uchun kattaroq massiv oldingi vaqtga nisbatan 1 ta qisqa bo‘ladi. Agar har bir ishlov berilgan elementlarning eng kichigi yoki eng kattasi mos yozuvlar sifatida tanlansa, bu sodir bo‘lishi mumkin.

Bunday holda, n-1 “bo‘linish” amallar talab qilinadi va umumiy ishlash muddati $\sum_{i=0}^{n}(n-1) = O(n^2)$ ta operatsiyani tashkil qiladi, ya’ni saralash kvadratik vaqt ichida amalga oshiriladi. Ammo almashtirishlar soni va shunga ko‘ra ish vaqti uning eng katta kamchiligi emas. Bundan ham yomoni, bu holda algoritmni bajarish paytida rekursiya darajasi n ga yetadi.

Mavzu yuzasidan savollar:

1. Saralash algoritmlari va ularning tahlili haqida gapiring

2. Eng sodda algoritmlar va ularning murakkabligi

3. QuickSort va Merge Sort algoritmlarining biri-biridan farqli jihatlari.

4. Eng sodda algoritmlarning eng yaxshi va eng yomon holatdagi ishlash vaqtlarini tahlil qilish

5. Quick Sort algoritmining eng yaxshi va eng yomon holatdagi bahosini tahlil qiling.

Mustaqil ishlash uchun masalalar:

1. Alisher 5-“B” sinf o‘quvchilariga dars beradi. Bu sinfda 30 ta o‘quvchi o‘qiydi. Alisher Jismoniy tarbiya fani o‘qituvchisi. 5-“B” sinf o‘quvchilari orasida eng ezun bo‘yga ega bo‘lgan uchta o‘quvchini bo‘ylari yig‘indisi, eng pastga bo‘yga ega bo‘lgan uchta o‘quvchining bo‘ylari yig‘indisidan necha barobar katta ekanligini aniqlang.

2. Sizga bir o‘lchamli butun sonlardan iborat massiv berilgan. Sizning vazifangiz bu massiv elementlarini modullari jihatdan kamaymaslik tartibida saralaydigan dastur tuzish. Agar modul jihatdan teng musbat va manfiy sonlar mavjud bo‘lsa manfiy son oldinroq joylashtirilsin:

Masalan:

9 8 -9 2 -4 3

2 3 -4 8 -9 9

3. Buxgalter. Buxgalterda xodimlarning maoshlari miqdori ma’lum. Buxgalter eng yuqori miqdorda maosh oluvchi xodimlar nechta ekanligini bilmoqchi. Buxgalter eng yuqori maosh miqdori qancha ekanligini bilmaydi.

4. Sportchilar natijasi. Og‘ir atletika bo‘yicha musobaqa o‘tkazilmoqda. Ushbu musobaqada 10 ta sportchi kurash olib bormoqda. Ular uchta urinishni amalga oshirishadi. Natija sifatida esa urinishlarning o‘rta arifmetigi yoziladi. “Oltin”, “Kumush”, “Bronza” medal sohiblari qanday natija ko‘rsatganini aniqlang. Agar natija qiymatlari bir xil bo‘lsa, shuncha miqdorda “Oltin”, “Kumush”, “Bronza” medal beriladi deb hisoblansin.

6-§. Graflar nazariyasi elementlari va o'tish algoritmlari

Graflarni o‘rganish bilan shug‘ullanadigan diskret matematikaning bo‘limi “Graflar nazariyasi” deb ataladi. Graflar nazariyasida ushbu matematik obyektlarning asosiy tushunchalari, xususiyatlari, tasvirlash usullari va qo‘llanilish sohalari batafsil ko‘rib chiqilgan. Bizni faqat dasturlashda muhim bo‘lgan jihatlari qiziqtiradi.

Graflar - bu chiziqlar bilan bog‘langan nuqtalar to‘plami. Nuqtalar **uchlar** (tugunlar) chiziqlar esa **qirralar** (yoylar) deb nomlanadi.

Grafning ifodalanishi.

Grafning to‘plam nazariya bo‘yicha ta’rifi. Bizga V- bo‘sh bo‘lmagan to‘plam berilgan, masalan $\{v_1, v_2, v_3, v_4, v_5\}$.

Uning barcha ikki elementli $V^{(2)}$ ichki to‘plamlari to‘plamini yozamiz. Bizning misolimiz uchun ushbu to‘plam quyidagicha bo‘ladi:

$$V^{(2)} = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_1, v_5\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_4, v_5\}\}.$$

Ixtiyor ravishda ba’zi bir $E \subseteq V^{(2)}$ ni olamiz, masalan:

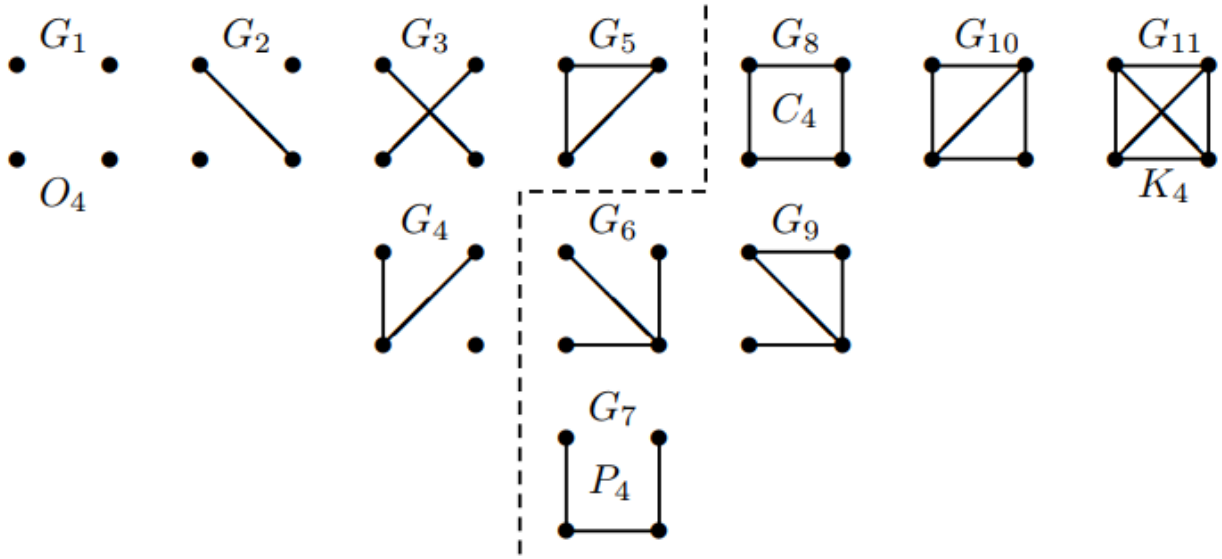
$$E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_4, v_5\}\}.$$

$\langle V, E\rangle$ juftligi yo‘naltirilmagan G grafi deb nomlanadi, unda V - uchlar to‘plami, E - qirralarning to‘plami, $V^{(2)}$ to‘plamining ichki to‘plami hisoblanadi.

Yuqoridagilardan kelib chiqib, ushbu ta’rif odatda quyidagicha shakllantiriladi: $\langle V, E\rangle$ oriyentirlanmagan graflar juftligi deb ataladi, agar V uchlar deb ataladigan bo‘sh bo‘lmagan elementlar to‘plami bo‘lsa va $E - V$ dagi qirralar deb ataluvchi turli elementlarning tartibsiz juftlari to‘plami bo‘lsa.

Graflar nazariyasida turli xil munosabatlarni yozishda VG yoki V(G) yozuvlari, G graf qirralarining to‘plami uchun EG yoki E(G) yozuvlari ishlatiladi.

Grafni namoyish qilishning vizual usuli - bu chizmalar (diagramma), unda uchlar nuqta, doiralar yoki boshqa figuralar bilan tasvirlangan va qirralar juft uchlari tasvirlarini bir-biriga bog‘laydigan chiziqlar bilan tasvirlangan. Yuqorida tavsiflangan grafni bunday tasvirlash uchun quyidagi variantlar berilgan.



16-rasm. Graf turlari

Graf - bu abstrakt obyekt bo‘lib, uchlar to‘plami (tugunlar) va qirralarning to‘plami - uchlar juftliklari orasidagi bog‘lanishlardan tashkil topadi (ulanishlar). Graf mavzusi juda keng. Graflar diskret matematikaning o‘rganish mavzusidir (bu yerda graf tushunchasining aniqroq ta’rifi berilgan). Graf murakkab tuzilgan ma’lumotni tavsiflash uchun ishlatiladi va shuning uchun katta amaliy ahamiyatga ega. Matematikada graflar paydo bo‘lishiga Eyler asarlari yordam berdi.

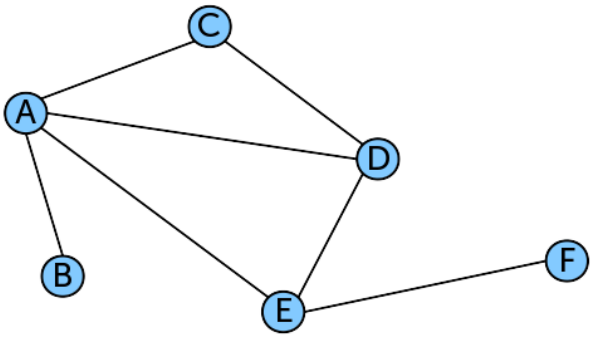
Graflar bilan qayerda uchrashamiz? Ehtimol, ular bilan qayerda uchrashmasligimizni aytish osonroq. Ya’ni biz graflarda juda ko‘p holatda uchratamiz. Misol qilib quyidagilarni keltirishimiz mumkin:

- Lokal yoki global tarmoq modeli
- Algoritmlarning blok-sxemasi
- Elektr sxemalar

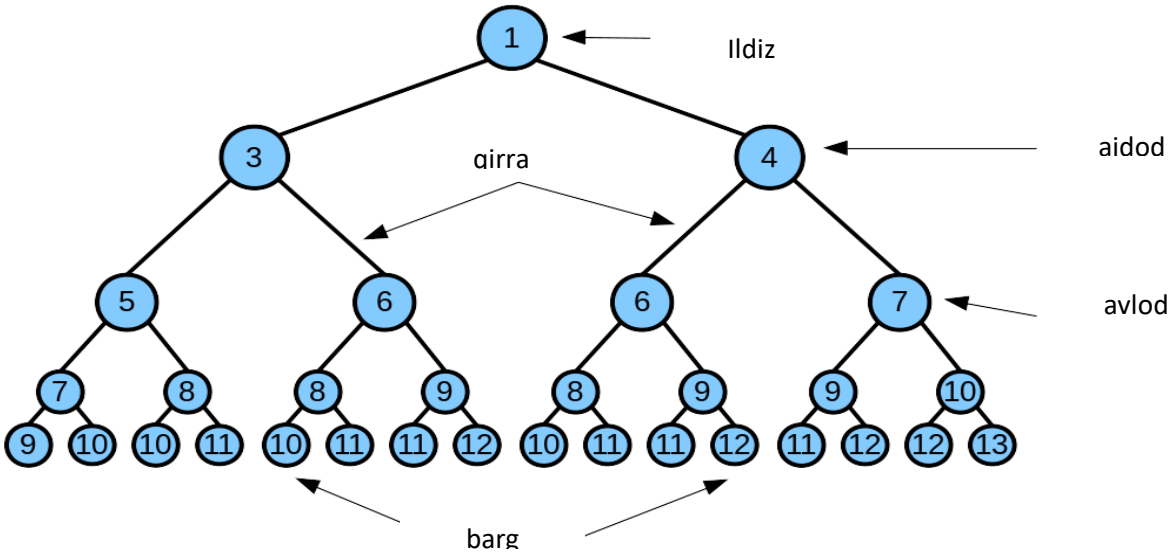
- Oila daraxti (Shajara)
- Metro xaritasi
- Ma’lumotlar bazasi modeli
- Aqlli xaritalar

va boshqa ko‘plab sohalarda qo‘llanilib kelmoqda. Ushbu darsda butun graflar nazariyasini olish mumkin emas. Shuning uchun qisqacha ma’lumotlarni keltirib o‘tamiz.

G graf - G: = (V, E) tartiblangan juftlik, bu yerda V - uchlarning (yoki tugunlarning) bo‘sh bo‘lmagan to‘plami, E esa qirralar deb nomlangan uchlarning juftlari to‘plamidir. Grafning uchlari va qirralari (ular graf elementlari deb ataladi), grafdagi uchlar soni | V | - graf tartibi, qirralarning soni | E | - graf hajmi deb ataladi.



17-rasm. Yo‘naltirilmagan graf



18-rasm. Daraxt - bu bog‘langan asiklik graf

Graflar nazariyasining asosiy atamalari. Bu yerda graflar nazariyasidan (diskret matematikaning bir bo‘limi) atamalarning kichik tanlovini qildik, ammo bu atamalar boshqa adabiyotlarda boshqacha berilgan bo‘lishi mumkin.

4-jadval.

Graflar nazariyasining boshlang‘ich terminologiyasi

O‘zbek	Рус	En	Tavsif
Uch	Вершина	vortex	Grafning elementi
Tugun	Узел	node	Uch tushunchasi bilan bir xil
Qirra	Ребро	edge	Ikki qo‘shni uchlarning bog‘lanishi
Yoy	Дуга	arc	Qirra bilan bir xil, lekin orgrafda emas
Aloqa, bog‘lanish, munosabat	Связь	link	Graf elementi (qirra yoki yoy)
Qo‘shnilik	Смежность	adjacent	Ikkita uch o‘rtasida aloqa mavjud bo‘lganini bildiruvchi atama
Insidentlik	Инцидентность	incident on	Uchga nisbatan qirra haqida
Daraja	Степень	degree	Uchga tutashgan qirralarning soni

6.1. Graflar nazariyasining asosiy tushunchalari

Grafdagi marshrut - bu har bir uch (oxirgisidan tashqari) ketma-ketlikdagi keyingi uchga qirra bilan bog‘langan uchlarning cheklangan ketma-ketligi.

Yo‘l - bu qirralarning takrorlanmagan yo‘lidir. **Oddiy zanjir** - bu uchlarni takrorlamaydigan marshrut (bu oddiy zanjirda takrorlanadigan qirralarning yo‘qligini anglatadi)

Orgrafdagi **yo‘naltirilgan marshrut** (yoki yo‘l) - bu har bir element oldingi va keyingi qismga tushadigan uchlar va yoylarning cheklangan ketma-ketligi.

Birinchi va oxirgi uchlar bir-biriga to‘g‘ri keladigan zanjirlar **sikl** deb ataladi (1-rasmda ACD va ACDE sikllar)

Yo‘lning (yoki siklning) uzunligi uni tashkil etuvchi qirralarning soni deyiladi

Agar uning qirralari takrorlanmasa, **yo‘l (yoki sikl) oddiy** deb nomlanadi; agar u sodda bo‘lsa va undagi tepaliklar takrorlanmasa u **elementar** deb nomlanadi.

Graf turlari. Yo‘naltirilgan graf - (qisqacha orgraf) - qirralari yo‘naltirilgan graf (4-rasm pastga qarang).

Yo‘naltirilmagan graf - uchlar juftligi tartiblanmagan graf (22-rasm)

Bog‘langan graf - bu har qanday uch juftligi o‘rtasida kamida bitta yo‘l mavjud bo‘lgan graf.

Daraxt - bu bog‘langan asiklik grafik, ya’ni sikllar yo‘q va tepalik juftligi orasida bitta yo‘l bor (18-rasm). Kirishning nol darajasiga ega bo‘lgan uch **daraxtning** ildizi, chiqish nol darajaga ega tugunlar esa **barglar** deb nomlanadi.

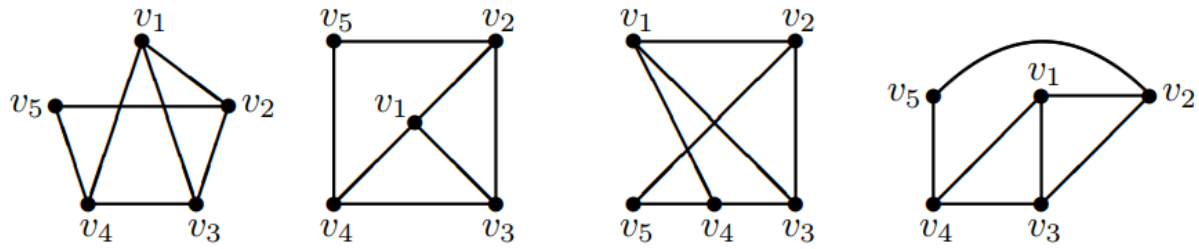
Bog‘langan va bog‘lanmagan graflar. 16-rasmda ko‘rsatilganlar graflarni ikki guruhga bo‘lish mumkin (kesilgan chiziq bilan ajratilgan): bog‘lanmagan ($G_1 - G_5$) va bog‘langan ($G_6 - G_{11}$).

Bog‘lanmagan graflarda qirralar bilan ulanmagan ikki yoki undan ortiq qismlar mavjud bo‘ladi. Ushbu qismlar bog‘lanish komponentlari deb ataladi.

Yuqorida keltirilgan graflarda G_1da to‘rtta komponent, G_2da uchta komponent, $G_3, G_4 \; va \; G_5$da 2 ta komponent mavjud, qolganlarida esa bittadan komponent mavjud. G_6 va G_{11} graflari o‘rtasida G_{11} grafini alohida ajratib ko‘rsatish mumkin, chunki to‘rtinchi darajali graf uchun maksimal sondagi qirralarga ega.

Daraxtlar va zanjirlar. Bog‘langan graflarda minimal miqdordagi qirralar mavjud bo‘lsa, ($|EG| = n - 1$) ular daraxtlar sinfini tashkil etadi. Yuqorida rasmda bu G_6va G_7 daraxtga to‘g‘ri keladi. Daraxtlar haqida keyingi mavzuda batafsil to‘xtalamiz. Bu yerda biz 16-rasmda P4 sifatida belgilangan G7 grafini qayd etamiz, bu daraxtning alohida holati va oddiy zanjir deb ataladi.

Umuman olganda zanjir – uchlar va qirralarning $(v_0, e_1, v_1, e_2, v_2, \dots, v_{k-1}, e_k, v_k)$ o‘zgaruvchan ketma-ketligi. Bu yerda v_{i-1} va v_i e_i qirraning oxirlari hisoblanadi. Ushbu yozuvni qisqacha shaklda quyidagicha yozishimiz mumkin: $(v_0, v_1, \dots, v_{k-1}, v_k)$ yoki $(e_1, e_2, \dots, e_{k-1}, e_k)$. Umumiy turdagi oddiy zanjirdan farqli o‘laroq, u takrorlanadigan uchlarni o‘z ichiga olishi mumkin. Masalan, quyida keltirilgan graflarda $(v_1, v_2, v_5, v_4, v_3)$ – oddiy zanjir, $(v_1, v_2, v_5, v_4, v_1, v_3)$ – zanjir.



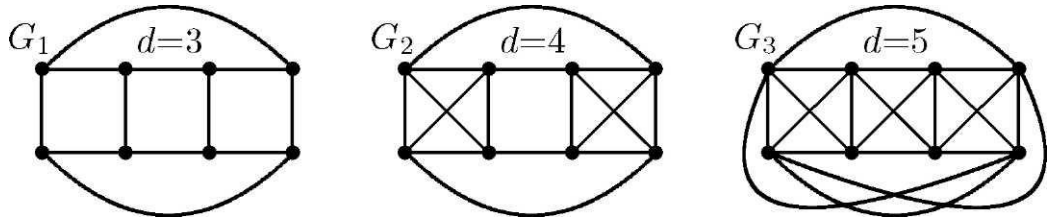
19-rasm. Graflarning vizual namoyish qilinishi

Odatda zanjir mustaqil graf sifatida emas, balki ba’zi bir graflarning bir qismi sifatida qaraladi. Zanjirning uzunligi - uni tashkil etuvchi qirralarning soni. Oddiy zanjirning uzunligi o‘z ichiga olgan graf uchlari sonidan, umumiy zanjirning uzunligi esa ushbu graf qirralarining sonidan oshmasligi aniq.

Graflar nazariyasida **zanjir** tushunchasi keng qo‘llaniladi. Masalan, bog‘langan grafni har qanday uchlar juftligi kamida bitta zanjir bilan bog‘langan graf sifatida belgilash mumkin.

Sikllar. Sikl (oddiy sikl) - bu yopiq zanjir (oddiy zanjir). Oddiy siklga G_8 grafi misol bo‘la oladi. Oddiy siklni ifodalovchi graf C_n bilan belgilanadi. Zanjirlarda bo‘lgani kabi, sikllarni ba’zi bir graflarning qismlari sifatida ko‘rib chiqish qiziq.

Regulyar graflar. 16-rasmdagi G_1, G_3, G_8, G_{11} graflari ularning har birida barcha uchlar bir xil darajaga ega ekanligi bilan ajralib turadi. Bunday graflar regulyar yoki bir jinsli deb nomlanadi. 20-rasmda uchinchi, to‘rtinchi va beshinchi darajadagi muntazam sakkizta uchli graflar ko‘rsatilgan.



20-rasm. Regulyar graflar

Uchinchi darajali graflar kubik deb nomlanganiga e’tibor bering. – rasmdagi G_{11} va –rasmdagi G_1. Shubhasiz, d darajali regulyar grafdagi qirralarning soni $m = \frac{1}{2}nd$ ga teng. Bundan kelib chiqadiki, toq sonli uchlar uchun regulyar graf faqat juft darajaga, toq daraja uchun esa faqat uchlar soni bo‘lishi mumkin. Shuning uchun har qanday kubik graf uchlarning juft soniga ega.

6.2. Grafni tasvirlash usullari

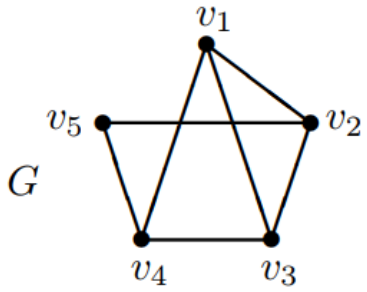
Grafni tasvirlash uchun bir nechta usullardan foydalaniladi. Graflardan o‘tish uchun siz o‘zingizning muammoingizni eng samarali hal qiladiganlardan foydalanishingiz kerak. Ko‘pincha, tanlov qo‘shnilik matritsasi va qo‘shnilik ro‘yxati o‘rtasida bo‘ladi (quyidagi jadval ikkala yondashuvning samaradorligini taqqoslaydi). Shu bilan birga, o‘rnatilgan C-massivga tayanib, o‘zingizning ma’lumotlar tuzilmalaringizni modellashtirishingiz va STD-da mavjud bo‘lgan turli xil konteynerlardan foydalanishingiz mumkin.

Qo‘shnilik matritsasi. Qo‘shnilik matritsasini n-tartibli $A = [a_i, a_j]$ nosimmetrik kvadrat matritsa sifatida aniqlaymiz, unda $a_{i,j}$ elementlar 1 ga teng, agar grafda $\{v_i, V_j\}$ qirrasi bo‘lsa, ya’ni v_i va v_j qo‘shni bo‘lsa, 0 ga teng, agar bunday qirra mavjud bo‘lmasa.

Ta’rifdan kelib chiqadiki, har qanday i uchun $\sum_{j=1}^{n} a_{i,j} = d(v_i)$, har qanday j uchun $\sum_{i=1}^{n} a_{i,j} = d(v_j)$ va $\sum_{i=1}^{n}\sum_{j=1}^{n} a_{i,j} = 2m$, ya’ni qo‘shnilik matritsasining har qanday qatori yoki ustunidagi birlar soni

grafning tegishli vertikal darajasiga teng va ularning umumiy soni uning qirralarining ikki baravariga teng.

Misol sifatida –rasmda berilgan G grafning A qo‘shnilik matritsasini $degv_i$ darajalar ketma-ketligini yozamiz.



$$\mathbf{A} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{c} \deg v_i \\ 3 \\ 3 \\ 3 \\ 4 \\ 1 \end{array}$$

21-rasm. Grafni qo‘shnilik matritsasi orqali tasvirlash

Graflarning ayrim turlarining qo‘shnilik matritsalarini qarab chiqaylik. Bo‘sh graf O_n qo‘shnilik matritsasi faqat nollardan iborat, ya’ni A (O_n) = 0_n.

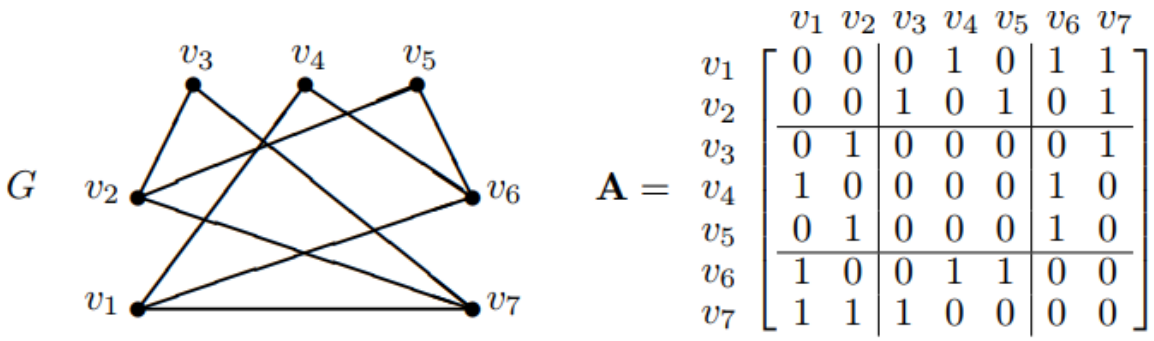
K_n to‘liq grafning qo‘shnilik matritsasi faqat dioganal elementlari birlardan iborat, qolgan elementlari nolga teng bo‘ladi. Buni $A(K_n) = 1_n - I_n$ deb yozamiz. Agar graf bog‘lanmagan bo‘lsa s komponentlarga ega bo‘lsa, unda satrlar va ustunlarni qayta tartibga solish orqali uning qo‘shnilik matritsasisini blok-diagonal shaklga keltirilishi mumkin:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{A}_{ss} \end{bmatrix},$$

Bu yerda har bir A_{ii} diagonal bloki s_i komponentining qo‘shnilik matritsasi. k-qismli graf holatida qo‘shnilik matritsasini blokli shaklga keltirish mumkin, agar asosiy diagonal bo‘ylab faqat "nol" bloklar kelsa:

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{A}_{12} & \dots & \mathbf{A}_{1k} \\ \mathbf{A}_{21} & \mathbf{0} & \dots & \mathbf{A}_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{k1} & \mathbf{A}_{k2} & \dots & \mathbf{0} \end{bmatrix}.$$

Masalan, 22-rasmda $\{v1, v2\}$, $\{v3, v4, v5\}$, $\{v6, v7\}$ bo‘laklar va unga qo‘shnilik matritsasi A bo‘lgan uch qismli G graf ko‘rsatilgan.



22-rasm. Grafni qo‘shnilik matritsasi orqali tasvirlash

Qo‘shnilik matritsasining bir xil analogi Kirxgof matritsasi $K = [k_{i,j}]$, n tartibli kvadrat matritsa sifatida aniqlangan, uning elementlari

$$k_{i,j} = \begin{cases} -1, & agar\ v_i\ va\ v_j\ \ qo'shnilar\ bo'lsa, \\ 0, & agar\ v_i\ va\ v_j\ \ qo'shnilar\ bo'lmasa, \\ deg v_i & agar\ i = j. \end{cases}$$

Qo‘shnilik matritsasi A va K Kirxgof matritsasi o‘rtasidagi bog‘liqlik, $K = D - A$ shaklga ega, bu yerda $D = diag\ (deg\ v_1, deg\ v_2, \dots, deg\ v_n)$, ya’ni bu diagonali elementlari mos keladigan uchlar darajalariga teng bo‘lgan matritsa. Kirxgof matritsasining muhim xususiyati (har qanday satr va har bir ustun elementlari yig‘indisi nolga teng bo‘lgan boshqa har qanday matritsa kabi) matritsaning barcha elementlarining algebraik qo‘shimchalari bir-biriga teng bo‘lishidir.

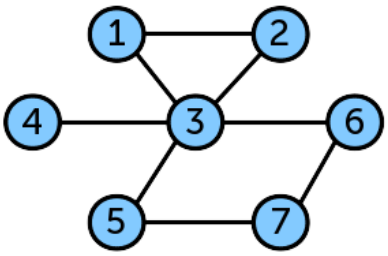
Izomorf graflar faqat mos mavhum graf uchlarini belgilashda (raqamlashda) farq qilganligi sababli, ularning qo‘shnilik matritsalarini (Kirxgof matritsalari) bir-biridan satrlar va ustunlarni biroz almashtirish orqali olish mumkinligi aniq bo‘ladi.

1 dan n gacha raqamlangan G grafning qo‘shnilik matritsasi nxn kvadrat kattalikdagi A matritsasi bo‘lib, unda a [i][j] elementining qiymati 1 ga teng bo‘lsa, grafning i- va j- uchlari qo‘shni bo‘ladi, aks holda qiymati nolga teng bo‘ladi. Bunday matritsa **binar matritsa** deb

ham ataladi. Oddiy graf uchun asosiy diagonal elementlari 0 ga teng bo‘ladi.

Qo‘shnilik matritsasi orgrafni tavsiflash uchun ham, yo‘naltirilmagan grafni tasvirlash uchun ham mos keladi. Yo‘naltirilmagan graf uchun elementlarning qiymatlari asosiy diagonalga nisbatan nosimmetrikdir.

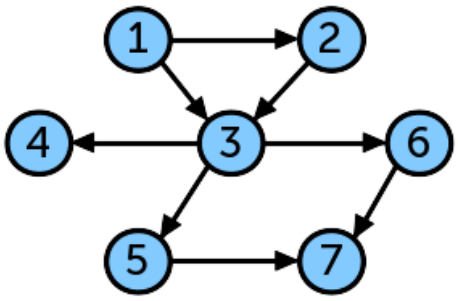
Qo‘shnilik matritsadan foydalanish faqat qirralari ko‘p bo‘lmagan hamda murakkab bo‘lmagan graflar uchun afzalroqdir, chunki u har bir element uchun bitta bit saqlashni talab qiladi. Agar graf murakkab bo‘lsa, unda xotiraning katta qismi nollarni saqlashga sarflanadi, ammo murakkab graflarda qo‘shnilik matritsasi grafni xotirada ixchamroq ifodalaydi va taxminan n^2 bit xotiradan foydalanadi. Ushbu kattalik qo‘shnilik ro‘yxatlariga qaraganda yaxshiroq (pastga qarang).



$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

22-rasm. Yo‘naltirilmagan grafda qo‘shnilik matritsasi

Qo‘shnilik matritsasini amalga oshirish uchun massivlar massivi qo‘llaniladi: vektorlar vektori (vector<vektor<bool>>) yoki kaliti uchlar soni, qiymati esa <bool> vektori. Agar grafni kengaytirish kerak bo‘lmasa, u holda vektorni **array (massiv)** bilan almashtirish kerak.

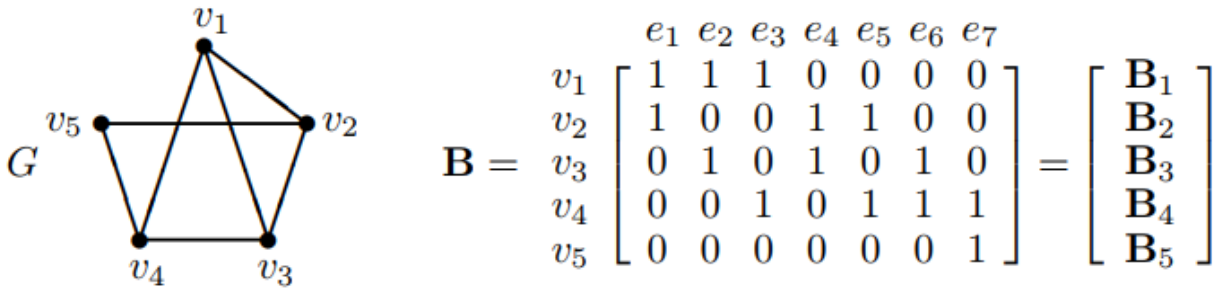


$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

23-rasm. Yo‘naltirilgan grafda qo‘shnilik matritsasi

Insidentlik matritsasi. Insidentlik matritsasi - bu grafning elementlari (qirra - uch) orasidagi bog‘lanishlar ko‘rsatiladigan grafni tasvirlash shakli. Matritsaning ustunlari qirralarga, satrlar uchlarga to‘g‘ri keladi. Demak, matritsa kvadrat bo‘lmaydi. Matritsa yacheykasidagi nolga teng bo‘lmagan qiymat uch va qirra (ularning insidentligi) o‘rtasidagi munosabatni bildiradi.

Graflar insidentlik matritsasini nxm o‘lchamdagi $B = [b_{i,j}]$ to‘rtburchaklar matritsasi sifatida aniqlaylik, bunda $b_{i,j}$ elementi 1 ga teng, agar e_j insident qirraning v_i uchi bo‘lsa, aks holda 0 bo‘ladi. B matritsasining satrlari insidentlik vektorlari deb nomlanadi va B_i bilan belgilanadi. 23-rasmda 21-rasmdagi kabi bir xil G graf ko‘rsatilgan va uning B insidentlik matritsasi ko‘rsatilgan.



24-rasm. Grafda insidentlik matritsasi

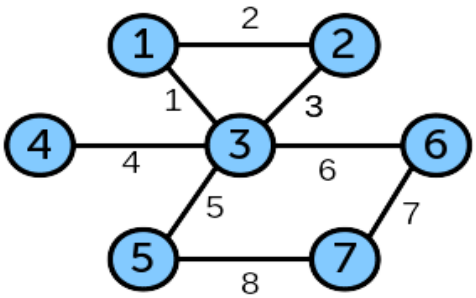
Ta’rifdan ko‘rinib turibdiki, insidentlik matritsasidagi birlarning umumiy soni graf qirralarining ikki baravariga teng, har qanday satrdagi birlar miqdori mos uchlar darajalariga teng, ustunlar esa ikkita birdan iborat.

Shuning uchun matritsa satrlari orasida oddiy munosabat mavjud: har qanday satr elementlarini ikki modulli qo‘shish orqali qolgan satrlarning bir xil elementlarining qo‘shnilarini olish mumkin. Insidentlik vektori tushunchasidan foydalangan holda, $B_i = \left(\sum B_j\right)$ (mod 2), bu yerda $1 \leq j \leq n$ va $j \neq i$. Shunday qilib, yuqoridagi matritsa uchun bizda: $B_1 = B_2 \oplus B_3 \oplus B_4 \oplus B_5 = [1,0,0,1,1,0,0] \oplus [0,1,0,1,0,1,0] \oplus [0,0,1,0,1,1,1] \oplus [0,0,0,0,0,0,1] = [1,1,1,0,0,00]$.

Bog‘lanmagan grafning insidentlik matritsasi, xuddi qo‘shnilik matritsasi singari, blok-diagonali ko‘rinishga keltirilishi mumkin, bu yerda har bir diogonal blok ba’zi bir bog‘langan komponentlarning insidentlik matritsasi hisoblanadi.

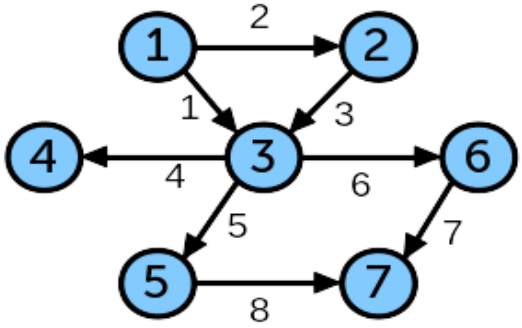
Grafda parallel qirralar bo‘lmaganligi sababli, agar v_i va v_j uchlar qo‘shni bo‘lsa, har qanday B_iB_j insidentlik vektorlari jufligi skalar ko‘paytmasi birga teng bo‘ladi va agar bu uchlar qo‘shni bo‘lmasa, skalar ko‘paytma nolga teng bo‘ladi. Binobarin, BB^T ko‘paytma graflarning qo‘shnilik matritsasiga to‘g‘ri keladigan mos darajalariga teng bo‘lgan diagonal elementlar bundan mustasno.

Yo‘naltirilgan graf holatida tegishli ustundagi har bir uch chiquvchi x vertikal qatorida "−1" va kiruvchi y uch qatorida "1" qiymatiga ega; agar uch va qirra o‘rtasida hech qanday bog‘liqlik bo‘lmasa, unda mos keladigan katak "0" qiymatiga ega bo‘ladi.



$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

25-rasm. Yo‘naltirilmagan grafda insidentlik matritsasi



$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \end{pmatrix}$$

26-rasm. Orgrafda insidentlik matritsasi

Qo‘shnilik ro‘yxati. Qo‘shnilik ro‘yxati - bu grafni uchlar ro‘yxati ("ro‘yxatlar ro‘yxati") to‘plami sifatida ko‘rsatish usuli - grafning har bir uchi qo‘shni uchlar ro‘yxatiga to‘g‘ri keladi. Masalan, 1-rasmni biz quyidagi qo‘shnilik ro‘yxati bilan tavsiflashimiz mumkin:

a: {b, c, d, e}

b: {a}

c: {a, d}

d: {a, c, e}

e: {a, f}

f: {e}

Bu sodda graflarni aks ettirish uchun ham, grafni kenglik yoki chuqurlikda bosib o‘tish uchun asosiy algoritmlarni amalga oshirish uchun ham eng qulay usuldir, bu yerda siz hozirda ko‘rib chiqilgan uchning "qo‘shnilarini" tezda olishingiz kerak.

Ko‘pgina masalalarni yechishda matritsalar bilan bir qatorda graflarni aks ettirish uchun qirralar ro‘yxati (insidentlik ro‘yxati) va uchlar ro‘yxati (qo‘shnilik ro‘yxat) ishlatiladi. Shunday qilib, -rasmda ushbu ro‘yxatlar berilgan:

5-jadval.

Qirralar ro‘yxati

e_1	e_2	e_3	e_4	e_5	e_6	e_7
v_1 v_2	v_1 v_3	v_1 v_4	v_2 v_3	v_2 v_4	v_3 v_4	v_4 v_5

6-jadval.

Uchlar ro‘yxati

v_1	v_2, v_3, v_4
v_2	v_1, v_3, v_4
v_3	v_1, v_2, v_4
v_4	v_1, v_2, v_3, v_5
v_5	v_2

Qirralarning ro‘yxatida har bir oxirgi uch juft uchlar bilan ifodalanadi, qo‘shnilik ro‘yxatida esa har bir uch uchun unga qo‘shni bo‘lgan barcha uchlar ko‘rsatiladi. Agar har bir ustunda ikkala birlikni tegishli uchlar (qatorlar) belgisi bilan almashtirsak va nollarni olib tashlasak qirralarning ro‘yxati insidentlik matritsasining ixcham yozuvi deb taxmin qilishimiz mumkin bo‘ladi. Xuddi shunday agar har bir satrda bo‘lgan birlar mos keladigan uchlar (ustun) belgisi bilan almashtirilsa va nollar olib tashlansa, qo‘shnilik matritsasidan uchlar ro‘yxatini olish mumkin.

7-jadval.

Qo‘shnilik matritsasi va ro‘yxatining algoritmlarda bajarilishini taqqoslash

Amal	Qo‘shnilik ro‘yxati	Qo‘shnilik matritsasi
(x,y) qirraning mavjudligini tekshirish	O(|V|+|E|)	O(1)
Qirraning darajasini hisoblash	O(1)	O(|V|)
Sodda grafilar uchun xotiradan foydalanish	O(|V|+|E|)	$O(|V|^2)$
Graflarda o‘tish	O(|V|+|E|)	$O(|V|^2)$

Mavzu yuzasidan savollar:

1. Graflarning umumiy ta’rifini bering.

2. Grafning eng asosiy tushunchalariga to‘xtalib o‘ting

3. Qo‘shnilik ro‘yxati va qo‘shnilik matritsasini hosil qilish jarayonini tushuntirib bering

4. Insidentlik matritsasi qanday hosil qilinadi?

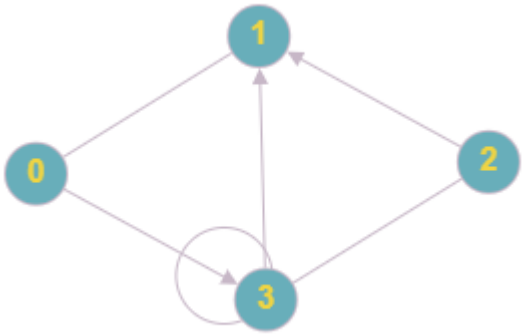
5. Graflar qo‘llaniladigan sohalarni sanang.

Mustaqil ishlash uchun masalalar:

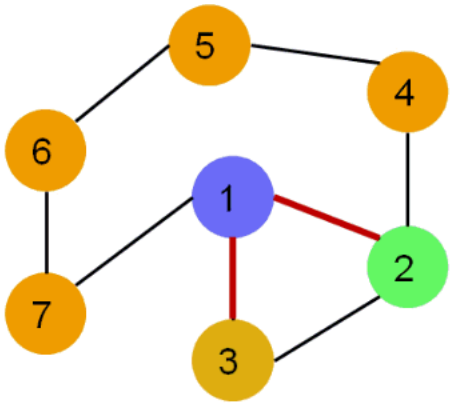
Quyida berilgan graflar bo‘yicha berilgan vazifalarni hal qiling:

1) Berilgan grafning barcha tushunchalarini keltiring (uchlar, qirralar, grafning yo‘nalishga ega yoki yo‘qligi bo‘yicha aniqlanishi, yakkalangan uch mavjudligi, regular yoki regular emasligi va hokazo).
2) Grafni mashina xotirasida tasvirlash
3) Grafni uchlar qo‘shniligi matritsasi orqali tasvirlang
4) Grafni qo‘shnilik ro‘yxati orqali tasvirlash
5) Grafning insidentlik matritsasi orqali tasvirlash

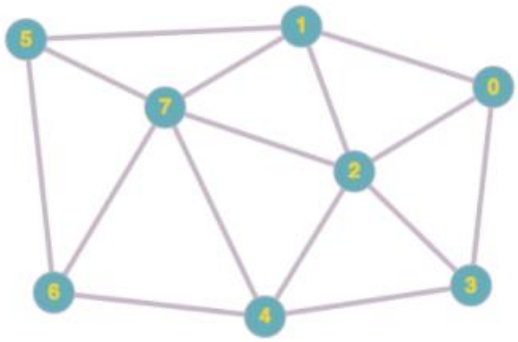
a)



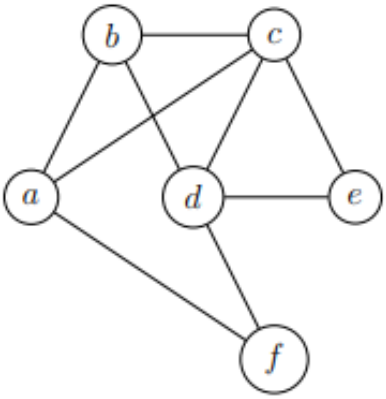
b)



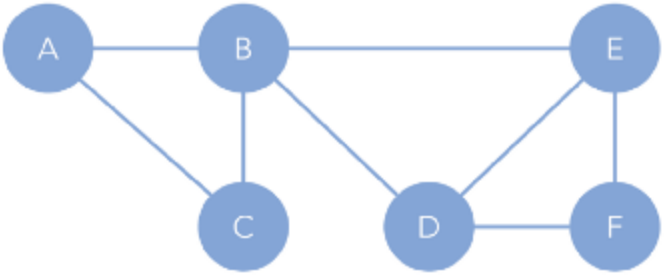
c)



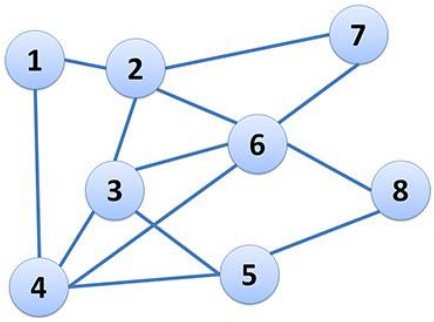
d)



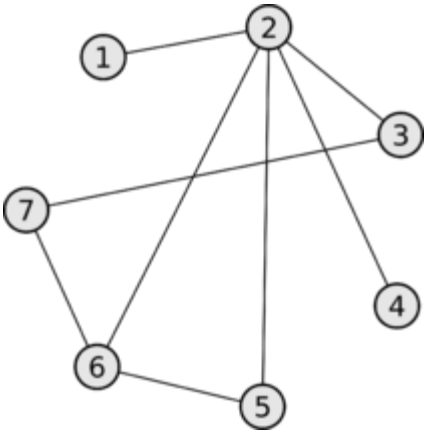
e)



f)



g)



7-§. Grafda o‘tish algoritmlari

Graflar bilan ishlashda barcha asosiy amallar (masalan, grafni bitta ko‘rinishda ikkinchisiga o‘tkazish, bosib chiqarish yoki grafni chizish) uning tizimli o‘tishini, ya’ni grafning har bir uchiga va har bir qirrasiga tashrif buyurishni nazarda tutadi. Agar labirintni graf ko‘rinishida namoyish etsak, u yerda qirralar o‘tish joylari, uchlar esa qirralarning kesishish nuqtalari bo‘lsa, u holda grafni bosib o‘tish uchun har qanday to‘g‘ri algoritm ixtiyoriy ravishda labirintidan chiqish yo‘lini topishi kerak. Ushbu algoritmlarning eng mashhurlari grafda o‘tish bo‘yi bo‘yicha qidiruv (DFS) va o‘tish eni bo‘yicha qidiruv (BFS) algoritmlari bo‘lib, ular qo‘llaniladigan muammolarni hal qilish uchun ko‘plab boshqa algoritmlar uchun asos bo‘lib xizmat qiladi.

Graflar bo‘ylab harakatlanishning asosiy g‘oyasi shundaki, har bir uchni birinchi marta tashrif buyurganingizda belgilang va barcha qirralari ko‘rib chiqilmagan uchlar haqidagi ma’lumotlarni saqlang. Qadimgi Yunoniston afsonalarida Tesey labirint atrofida yurish uchun Ariadna bergan ipdan foydalangan, qarichlab tosh yoki maydalangan toshlar bilan bosib o‘tgan yo‘lini belgilab qo‘ygan, sanab o‘tilgan turlar grafni bosib o‘tish uchun ishlatilgan. Grafni bosib o‘tish jarayonida har bir uch uchta holatdan birida bo‘ladi:

1) ochilmagan - uchning dastlabki holati;

2) ochiq - uch topilgan, ammo unga tushgan qirralar ko‘rib chiqilmagan;

3) ishlov berilgan (belgilangan) - ushbu uchga tushgan barcha qirralarga tashrif buyuriladi.

Grafning har bir uchi ketma-ket ushbu holatlarning barchasini qabul qilishi aniq. Dastlab, faqat bitta uch ochiq bo‘ladi, ya’ni grafni bosib o‘tish ushbu uchdan boshlanadi.

7.1. Grafda o‘tish eni bo‘yicha qidiruv- BFS algoritmi

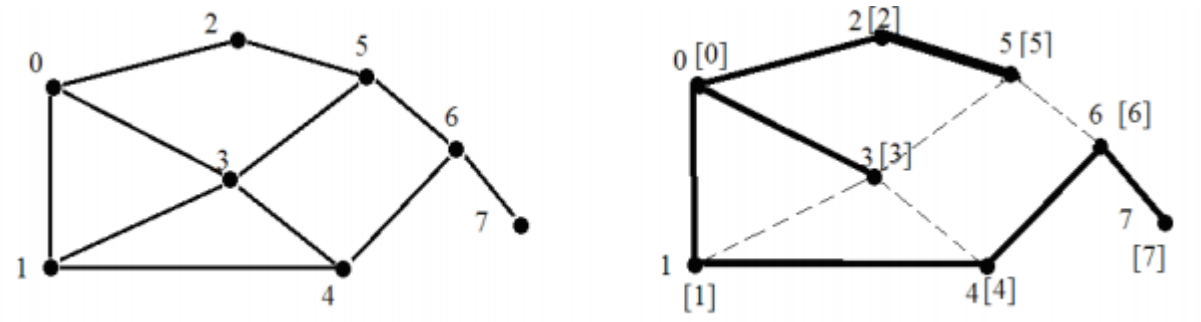
G = (V, E) grafi berilgan bo‘lsin va boshlang‘ich uchi v tanlansin. Birinchi kenglik bo‘yicha qidirish algoritmi v uchga yetib boruvchi barcha uchlarni "ochish" uchun G grafning barcha qirralarini muntazam ravishda kesib o‘tadi. O‘tish jarayonida barcha uchlarni o‘z ichiga olgan dastlabki uchda joylashgan kenglik bo‘yicha qidiruv daraxtini yaratadi. E’tibor bering, ildiz uchidan ushbu daraxtning istalgan uchiga masofa (qirralarning soni) eng qisqa bo‘ladi.

Kenglik bo‘yicha birinchi qidiruv ushbu nomga ega, chunki grafni bosib o‘tish jarayonida k+1 masofadagi har qanday uchni qayta ishlashdan oldin k masofadagi barcha uchlar belgilanadi.

Algoritm ham yo‘naltirilgan, ham yo‘naltirilmagan graflar uchun ishlaydi. Algoritm g‘oyasi: birinchisiga tutash bo‘lgan barcha uchlar ochiladi, ya’ni ular ro‘yxatga joylashtiriladi va bitta belgini oladi. Shundan so‘ng, dastlabki uch to‘liq qayta ishlanadi va 2 bilan belgilanadi.

Ro‘yxatning birinchi yuqori qismi keyingi uchga aylanadi. Amaldagi bilan qo‘shni bo‘lgan avval belgilanmagan barcha uchning ro‘yxatning oxiriga qo‘shiladi (ochiladi). Joriy uch ro‘yxatdan o‘chirilib, 2 raqami bilan belgilanadi. Jarayon uchlar ro‘yxati bo‘sh bo‘lguncha davom etadi. Ma’lumotlar ro‘yxatining ushbu ko‘rinishi navbat deyiladi.

Quyidagi misolni ko‘rib chiqamiz (27-rasm). 27-a) rasmda grafning dastlabki ko‘rinishi berilgan. 27-b) rasmda, uchlar yonida, graf uchlarini ko‘rish tartibi qavsda ko‘rsatilgan. Breadth First Search (BFS) daraxtini hosil qiladigan qirralar qalin rangda berilgan.



a) **b)**

27-rasm. BFS algoritmi jarayonida graf uchlarini ko‘rish

BFS algoritmi. Tasvirdan ko‘rinib turibdiki, algoritmning o‘zi juda ahamiyatsiz. Tashrif uchun uchlar navbati saqlanib qoladi. Keyingi uchga tashrif buyurganida, hali tashrif buyurmagan va hali navbatda bo‘lmagan barcha qo‘shnilari navbatga qo‘shiladi. Uchga allaqachon tashrif buyurilganligini tekshirish uchun bir qator yorliqlardan foydalaniladi. Dastlab, boshlang‘ich uchdan tashqari barcha i uchun visited[i] = false qiymatini qabul qiladi. i uch visited[i] navbatiga qo‘shilganda, true qiymati tayinlanadi.

```
#include <iostream>
using namespace std;

vector<int> graph[100000];
bool used[100000];

int main() {
   //Grafni kiritish
  // ... Bu qismda graf matritsa ko‘rinishida kiritiladi
   queue<int> q;
   q.push(0);
   used[0] = true;

   while (!q.empty()) {
      int cur = q.front();
      q.pop();

      cout << "BFS : " << cur + 1 << endl;
```

```
        for (int k: graph[cur]) {
            if (!used[k]) {
                q.push(k);
                used[k] = true;
            }
        }
    }
}
```

Ushbu algoritmning murakkabligi $O(n^2)$, bu yerda n - grafdagi uchlar soni. Darhaqiqat, har bir uch ochilib, navbatga bir martagina joylashtirilgan, shuning uchun navbatning uchlari orasidagi sikl n martadan koʻp boʻlmagan vaqtda bajariladi. while sikli grafning barcha uchlari siklni oʻz ichiga oladi va u n marta bajariladi. Agar biz graf tasvirni qoʻshnilik roʻyxati shaklida ishlatsak, unda murakkablik O(n+m) boʻladi, bu yerda m - qirralarning soni.

7.2. Grafda oʻtish boʻyi boʻyicha qidiruv (DFS) algoritmi

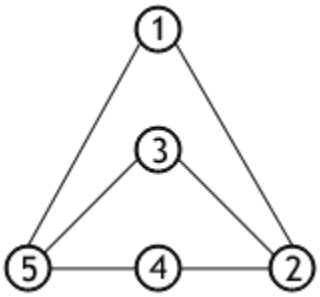
Grafda oʻtish boʻyi boʻyicha qidiruv (DFS) - bu graf uchlaridan oʻtishning rekursiv algoritmi. Agar boʻyi boʻyicha birinchi qidirish usuli nosimmetrik tarzda bajarilgan boʻlsa (grafning uchlari darajalar boʻyicha koʻrib chiqilgan boʻlsa), unda bu usul iloji boricha chuqurroq harakat qilishni oʻz ichiga oladi. Keyingi rivojlanishning mumkin emasligi shundan iboratki, keyingi qadam harakatlanishning bir nechta variantiga ega boʻlgan (ulardan biri toʻliq oʻrganib chiqilgan), ilgari tashrif buyurgan uch oxirgisiga oʻtish boʻladi.

Algoritm qanday ishlashini aniq bir misol yordamida koʻrib chiqamiz. Quyidagi yoʻnaltirilmagan bogʻlangan grafda jami 5 ta uch mavjud. Avval siz boshlangʻich uchni tanlashingiz kerak. Qaysi uch tanlangan boʻlsa ham, har qanday holatda ham graf toʻliq oʻrganib chiqiladi, chunki yuqorida aytib oʻtilganidek, bu bitta yoʻnaltirilmagan bogʻlangan graf. Oʻtish 1 tugundan boshlasin, u holda qarab chiqilgan tugunlar ketma-ketligi tartibi quyidagicha boʻladi: 1 2 3 5 4. Agar ijro,

masalan, 3 tugundan boshlangan bo‘lsa, u holda o‘tish tartibi boshqacha bo‘ladi: 3 2 1 5 4.

DFS algoritmi rekursiyaga asoslangan, ya’ni o‘tish funksiyasi o‘zini bajarilayotganda chaqiradi, bu esa kodni umuman ixcham qiladi.

Algoritmning psevdokodi quyidagicha



28-rasm. BFS algoritmi jarayonida graf uchlarini ko‘rish

DFS funksiya sarlavhasi (st)
Chiqish (st)
visited[st] ← tashrif buyurgan;
r = 1 uchun n gacha
Agar (graph[st, r] ≠ 0) va (visited[r] tashrif buyurilmagan) bo‘lsa, u holda DFS (r)

Bu yerda DFS (deep-first search) - bu funksiya nomi. Uning yagona parametri st - dasturning asosiy qismidan argument sifatida uzatiladigan boshlang‘ich uchdir. Mantiqiy qiymatlarni qabul qiladigan massivning har bir elementiga oldindan false (yolg‘on, 0) qiymat beriladi, ya’ni uchlarning har biri dastlab tashrif buyurilmagan deb yoziladi.

Ikki o‘lchovli **graph** massivi grafning qo‘shnilik matritsasi. Natijalar oxirgi satrda to‘planishi kerak. Agar qo‘shnilik matritsasining elementi, qandaydir bosqichda, 1 ga teng bo‘lsa (0 emas) va matritsaning tekshirilgan ustuni bilan bir xil songa ega bo‘lgan uchga tashrif buyurilmagan bo‘lsa, unda funksiya rekursiv ravishda takrorlanadi. Aks holda, funksiya rekursiyaning oldingi bosqichiga qaytadi.

```
#include <iostream>
using namespace std;
const int n=5;
int i, j;
bool *visited=new bool[n];
//qo‘shnilik grafi
int graph[n][n] =
{
   {0, 1, 0, 0, 1},
   {1, 0, 1, 1, 0},
   {0, 1, 0, 0, 1},
   {0, 1, 0, 0, 1},
   {1, 0, 1, 1, 0}
};
//bo‘yi bo‘yicha izlash
void DFS(int st)
{
   int r;
   cout<<st+1<<" ";
   visited[st]=true;
   for (r=0; r<=n; r++)
      if ((graph[st][r]!=0) && (!visited[r]))
         DFS(r);
}
int main()
{
   int start;
   cout<<"Qo‘shnilik matritsasi: "<<endl;
   for (i=0; i<n; i++)
   {
   visited[i]=false;
   for (j=0; j<n; j++)
   cout<<" "<<graph[i][j];
   cout<<endl;
   }
   cout<<"Boshlang‘ich uchni kiriting: >> "; cin>>start;
   // tashrif buyurilgan uchlar massivi
   bool *vis=new bool[n];
   cout<<" O‘tish tartibi: ";
```

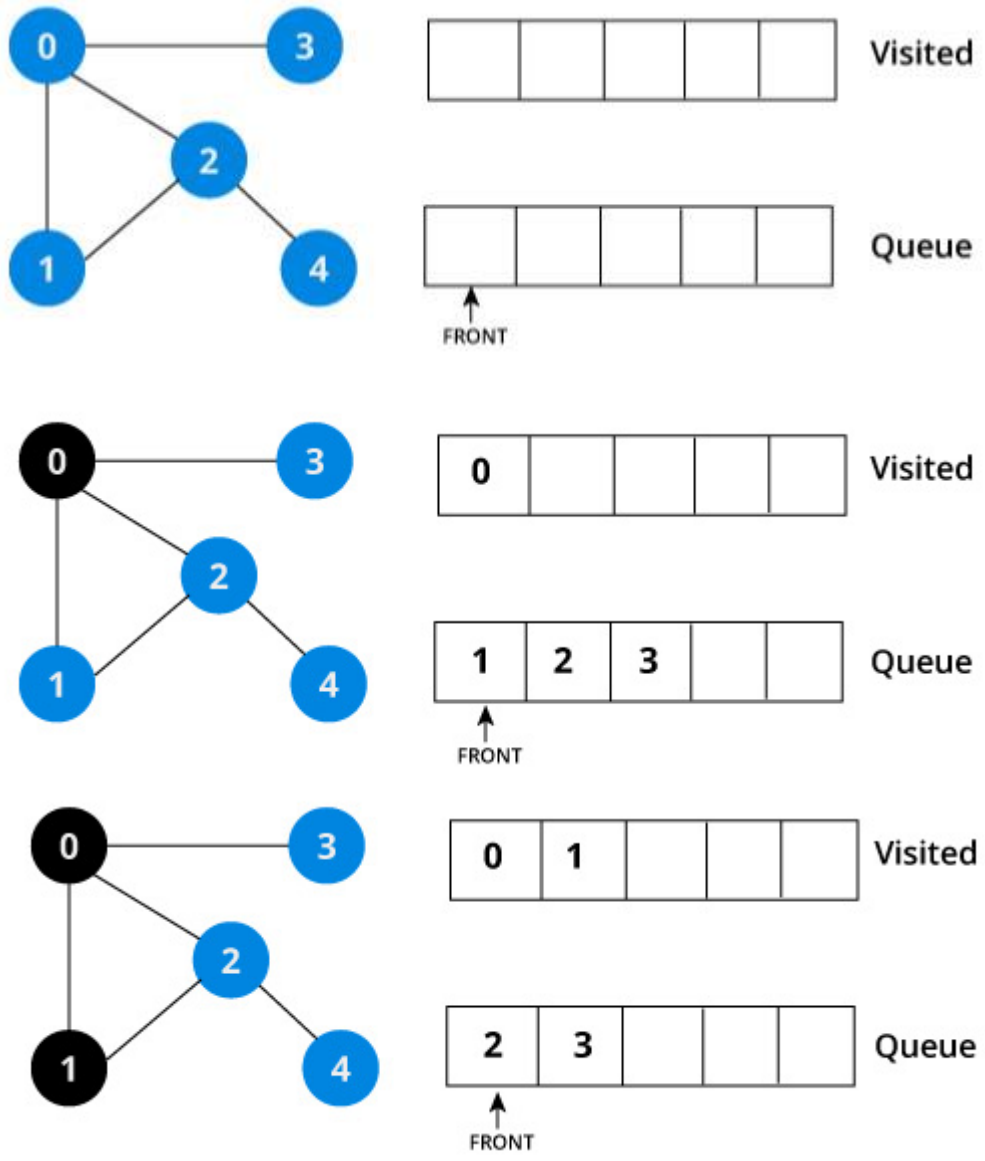
```
    DFS(start-1);
    delete []visited;
    system("pause>>void");
    return 0;
}
```

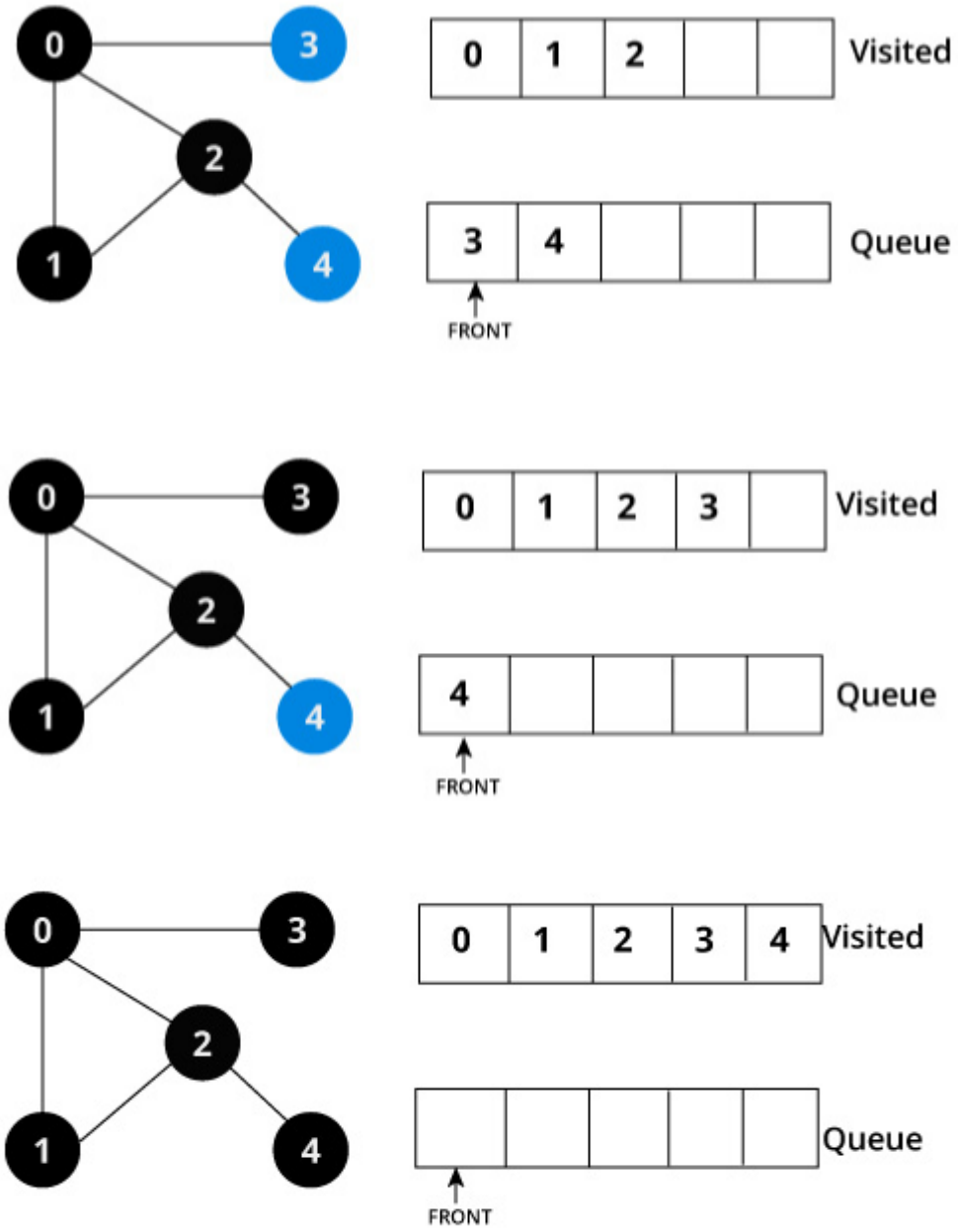
Misollar.

1-misol

BFS algoritmiga misol. Biz ikkita jadvalga ma’lumotlarni joylashtirib boramiz

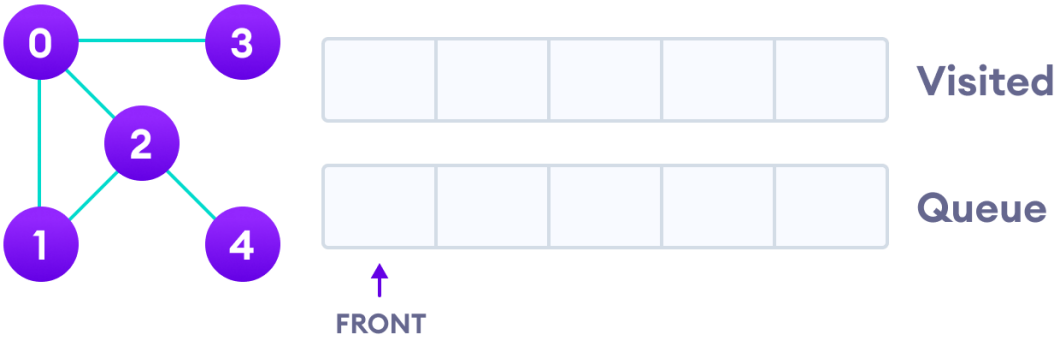
Tashrif buyurilgan uchlar – Visited jadvaliga. Navbatda turgan uchlar esa – Queue jadvaliga



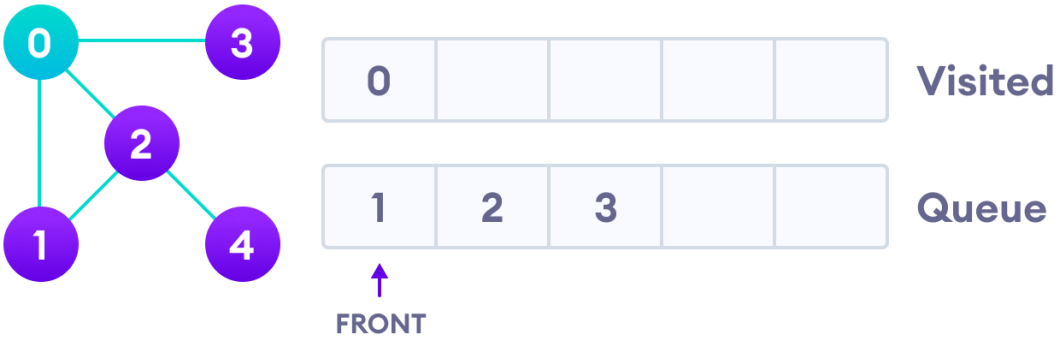


2-misol. Kenglik bo'yicha birinchi qidiruv (BFS)-bu graflar bilan ishlashning ko'plab muhim algoritmlari uchun asos bo'lgan, eng oddiy grafni o'tish algoritmlaridan biri.

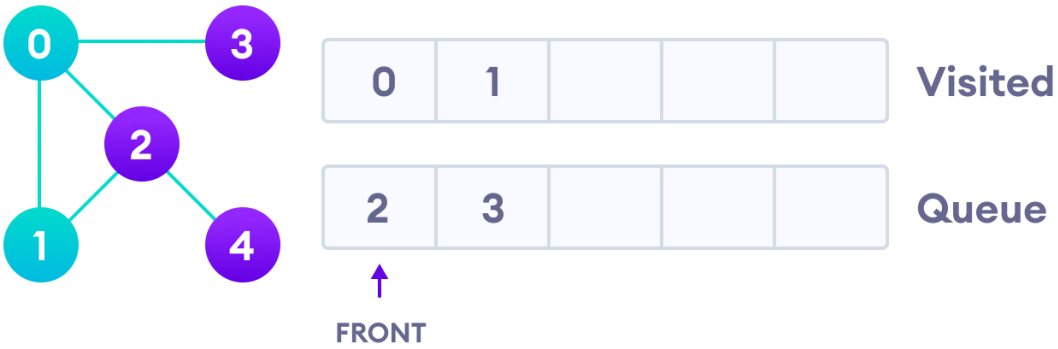
Keling, "Eni bo‘yicha qidirish" algoritmi qanday ishlashini misol bilan ko'rib chiqaylik. Biz 5 ta uchga ega bo'lgan yo'naltirilmagan grafdan foydalanamiz.



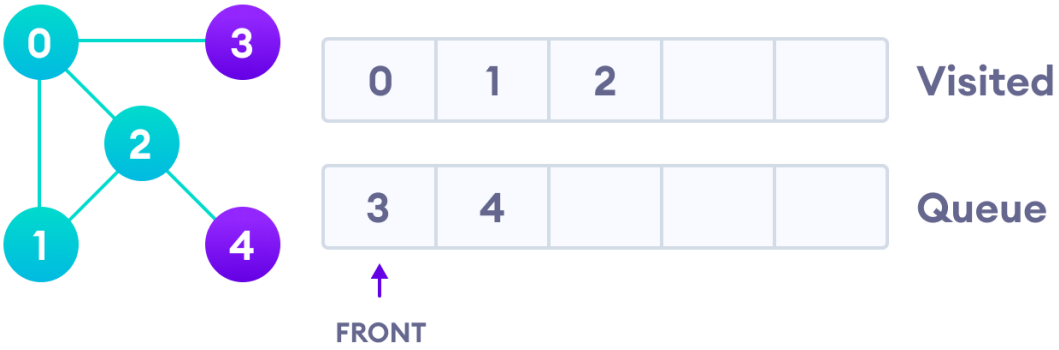
Biz 0 uchdan boshlaymiz, BFS algoritmi uni tashrif buyurilganlar ro'yxatiga qo'yib, uning yonidagi barcha uchlarni stekga qo'yishdan boshlanadi.

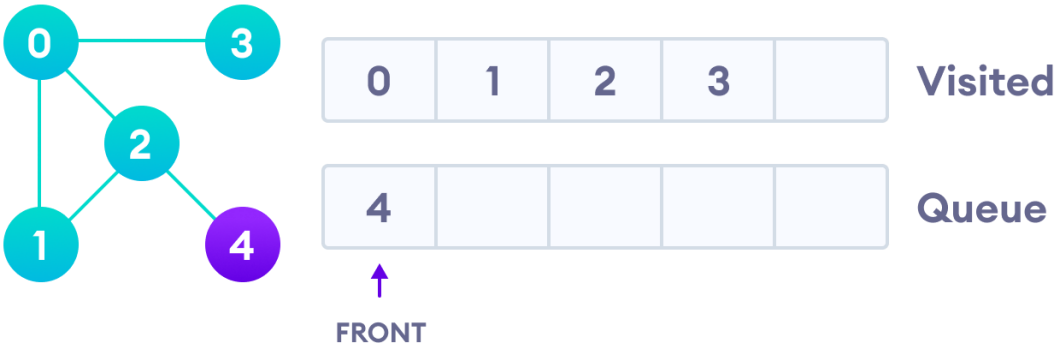


Keyinchalik, biz navbatning old qismidagi elementga tashrif buyuramiz, ya'ni 1 va uning yonidagi uchlarga o'tamiz. 0 tashrif buyurilgani uchun biz uning o'rniga 2 ga tashrif buyuramiz.

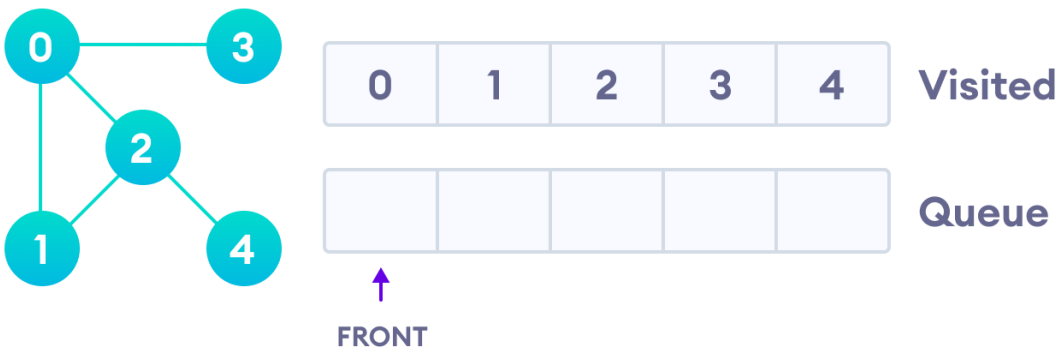


2-uchda ko'rilmagan qo'shni 4-uch bor, shuning uchun biz uni navbatning orqa qismiga qo'shamiz va navbatning oldida joylashgan 3 - ga tashrif buyuramiz.

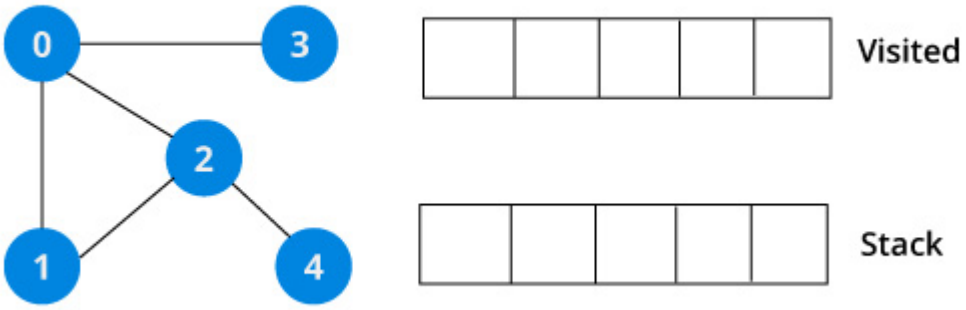




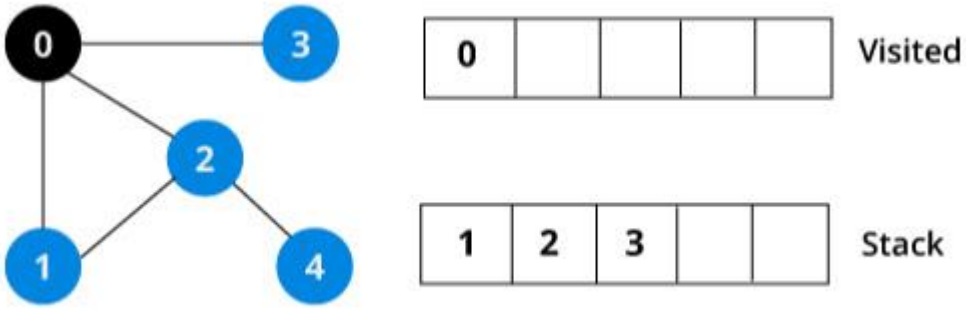
Navbatda faqat 4-uch qoladi, chunki qo'shni yagona 3-uch 3, ya'ni 0 ga tashrif buyurilgan. Biz unga tashrif buyuramiz.



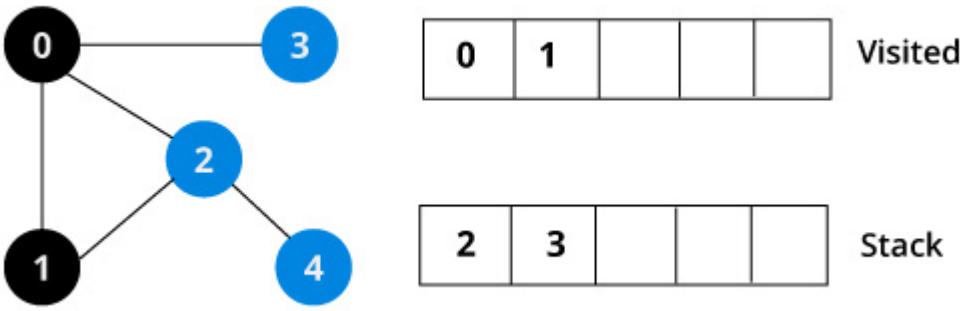
3-misol. DFS algoritmining ishlashi



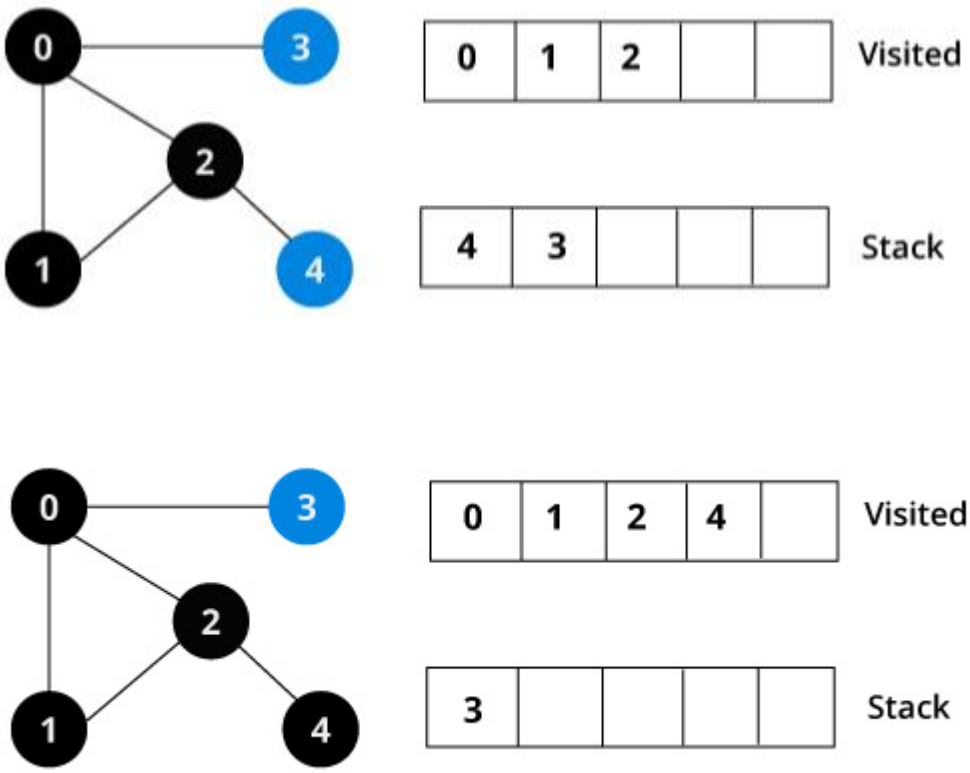
Biz 0 uchdan boshlaymiz, DFS algoritmi uni tashrif buyurilgan ro'yxatga qo'yishdan va barcha qo'shni uchlarni stekka joylashtirishdan boshlanadi.



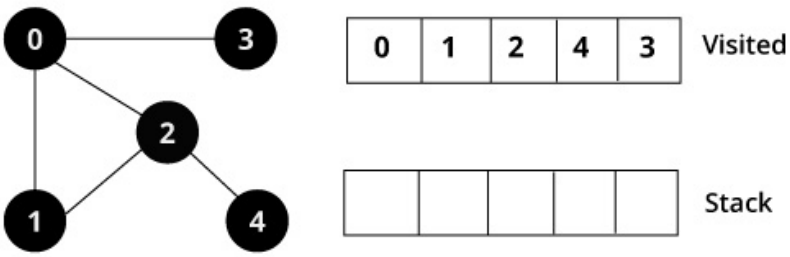
Keyin biz 1-uchning yuqori qismidagi elementga tashrif buyuramiz va qo'shni uchlarga o'tamiz. Biz allaqachon 0 ga tashrif buyurganimiz uchun, uning o'rniga 2 ga tashrif buyuramiz.



2-uchda ko'rilmagan qo'shni 4-uch bor, shuning uchun biz uni to'plamning yuqori qismiga qo'shamiz va tashrif buyuramiz.



Oxirgi 3-bandga tashrif buyurganimizdan so'ng, uning ko'zga ko'rinmas qo'shni uchlar yo'q. Bu grafni birinchi chuqurlik birinchi o'tishini yakunlaydi.



Mavzu yuzasidan savollar:

1. Graflarda o‘tish algoritmlari qanday masala hisoblanadi?
2. BFS algoritmining ishlash prinsipi qanday?
3. DFS algoritmining ishlash prinsipi qanday?

4. Graflarda yana qanday o‘tish algoritmlari mavjud?
5. Yuqorida keltirilgan algoritmlarning murakkabligini baholang

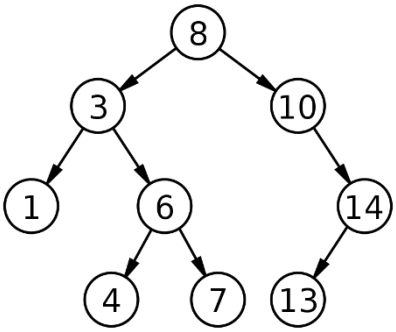
8-§. Daraxtlar grafning xususiy holati sifatida

Daraxt - bu bog‘langan asiklik graf, ya’ni sikllar yo‘q va uchlar juftligi orasida bitta yo‘l bor (29-rasm). Kirishning nol darajasiga ega bo‘lgan uch **daraxtning** ildizi, chiqish nol darajaga ega tugunlar esa **barglar** deb nomlanadi.

Ulanish har qanday uchlar juftligi o‘rtasida marshrut mavjudligini anglatadi, aylanuvchanlik sikllar yo‘qligini anglatadi. Demak, xususan, shundan kelib chiqadiki, daraxtdagi qirralarning soni uchlar sonidan bitta kamroq va har qanday uchlar juftlari orasida bitta va faqat bitta yo‘l bor.

O‘rmon – juda ko‘p daraxtlardir.

Yo‘naltirilgan (oriyentirlangan) daraxt - bu faqat bitta vertikal kirish nol darajasiga ega bo‘lgan (boshqa yoylar unga olib kelmaydigan), boshqa uchlarning kirish darajasi 1 bo‘lgan siklik orgraf (sikllarni o‘z ichiga olmaydigan yo‘naltirilgan graf).



29-rasm. Yo‘naltirilgan daraxt

Daraxtning asosiy tushunchalari

Ildiz tuguni - daraxtning eng yuqori tuguni (18-rasmdagi 8-tugun).

Ildiz – ixtiyoriy tanlab olingan uchlardan biri.

Barg yoki terminal tuguni – avlodi mavjud bo‘lmagan tugun (18-rasmdagi 1, 4, 7, 13 tugunlari).

Ichki tugun - bu daraxtga avlodi mavjud bo‘lgan har qanday tugun va shuning uchun barg tuguni emas (18-rasmda 3, 6, 10, 14).

Uchning darajasi - unga tushgan qirralarning soni.

Sentroid - uch, u olib tashlanganida hosil bo‘lgan ulanish komponentlarining o‘lchamlari $\frac{n}{2}$ dan oshmaydi (asl daraxtning yarmi kattaligi)

Tugun. Tugun - bu ba’zi bir qat’iy tabiat obyektiga mos keladigan ikki turdagi graf elementlaridan birining nusxasi. Tugun ma’lum bir ma’lumot strukturasining yoki daraxtning o‘zi qiymatini, holatini yoki ko‘rinishini o‘z ichiga olishi mumkin. Daraxtning har bir tugunida daraxt ostida joylashgan nol yoki undan ko‘p avlod tugunlari mavjud (odatda, daraxtlar haqiqiy daraxtlar singari yuqoriga emas, pastga qarab "o‘sadi"). Avlodga ega bo‘lgan tugun o‘z avlodiga nisbatan ajdod tugun deb nomlanadi (oldingi tugun yoki kattaroq tugun). Har bir tugunning ko‘pi bilan bitta ajdodi bor.

Tugunning balandligi - bu tugundan eng pastki tugunga (chekka tugunga) barg deb ataladigan pastga tushadigan yo‘lning maksimal uzunligi. Ildiz tugunining balandligi butun daraxtning balandligiga teng.

Ildiz tugunlari. Ajdodlari bo‘lmagan tugun (eng yuqorisi) ildiz tuguni deb ataladi. Bu daraxtdagi ko‘plab amallar boshlanadigan tugun (garchi ba’zi algoritmlar "barglar" dan boshlanib, ular ildizga yetguncha davom etadi). Boshqa barcha tugunlarga ildiz tugunidan qirralar (yoki bog‘lanishlar) bo‘ylab harakatlanish orqali erishish mumkin (rasmiy ta’rifga ko‘ra, har bir bunday yo‘l noyob bo‘lishi kerak). Diagrammalarda u odatda eng yuqori qismida tasvirlangan. Ba’zi daraxtlarda, masalan, uyumlarda, ildiz tuguni maxsus xususiyatlarga ega. Daraxtdagi har bir tugunni shu tugundan "o‘sayotgan" kichik daraxtning ildiz tuguni deb hisoblash mumkin.

Daraxt osti - bu alohida daraxt sifatida namoyish etilishi mumkin bo‘lgan daraxtga o‘xshash ma’lumotlar strukturasining bir qismidir. T daraxtining har qanday tuguni va uning barcha nasl tugunlari bilan birga T daraxtining pastki daraxti hisoblanadi. **Daraxt osti** har qanday tuguni uchun, yo ushbu kichik daraxtning ildiz tuguniga yo‘l bo‘lishi kerak, yo tugunning o‘zi ildiz bo‘lishi kerak. Ya’ni, kichik daraxt ildiz tuguniga butun daraxt bilan bog‘lanadi va boshqa barcha tugunlar bilan daraxt

osti aloqasi tegishli daraxt osti tushunchasi orqali aniqlanadi (“to‘plam osti" atamasi bilan o‘xshashlik bo‘yicha).

Daraxt strukturasi orasida tartiblangan daraxtlar eng keng tarqalgan. Binar (Ikkilik) qidiruv daraxti - tartiblangan daraxt turidir.

Daraxtlar ustida bajariladigan umumiy amallar:

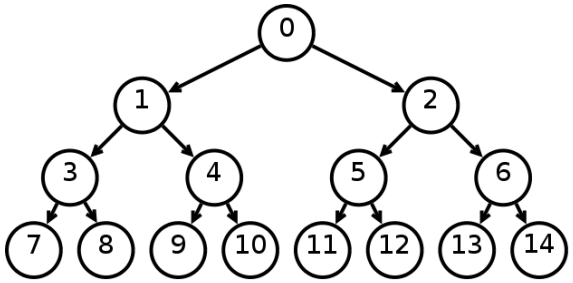
1) yangi elementni ma’lum bir joyga kiritish;
2) daraxt osti kiritish;
3) daraxt shoxini qo‘shish (payvandlash deb ataladi);
4) har qanday tugun uchun ildiz elementini topish;
5) ikkita uchning eng kichik umumiy ajdodini topish;
6) daraxtning barcha elementlarini sanab chiqish;
7) daraxt novdasi elementlarini sanab chiqish;
8) izomorfik daraxt osti qidirish;
9) elementni qidirish;
10) daraxt shoxini olib tashlash;
11) daraxt ostini olib tashlash;
12) elementni o‘chirish.

Daraxtlarning qo‘llanish sohalari:

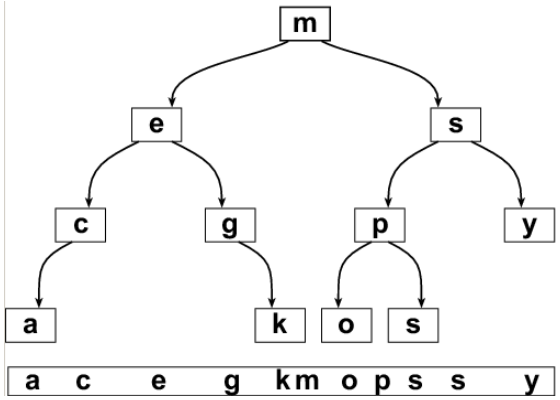
1) ma’lumotlar iyerarxiyasini boshqarish;
2) axborot olishni soddalashtirish
3) ma’lumotlarning saralangan ro‘yxatlarini boshqarish;
4) arifmetik ifodalarni tahlil qilish (inglizcha parsing), dasturni optimallashtirish;
5) turli xil vizual effektlarni olish uchun raqamli rasmlarni yaratish texnologiyasi sifatida;
6) ko‘p bosqichli qaror qabul qilish shakllarida (shaxmat).

8.1. Binar (ikkilik) daraxtlar

Ikkilik daraxt - bu har bir tugunda ko‘pi bilan ikkita avlod (bola) bo‘lgan ma’lumotlarning iyerarxik tuzilishi. Odatda, birinchisi ajdod tuguni, avlodlar esa chap va o‘ng merosxo‘rlar deb nomlanadi.



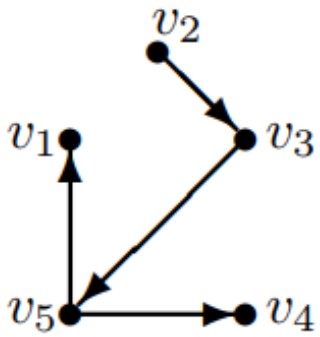
30-rasm. Binar daraxt



31-rasm. Kalitlari lotin alifbosi bo‘lgan ikkilik qidiruv daraxti, alfavit bo‘yicha tartiblangan.

8.2. Daraxtlarni mashinada tasvirlash usullari

Matritsali ko‘rinish. Daraxt, boshqa har qanday graf singari, matritsalar yordamida tasvirlanishi mumkin. Misol tariqasida quyida 32-rasmda ko‘rsatilgan tartiblangan daraxt uchun A – qo‘shnilik va B – insidentlik matritsalarini keltirilgan:



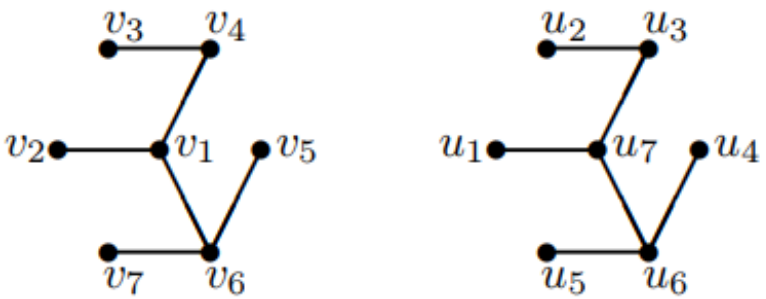
$$\mathbf{A} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \qquad \mathbf{B} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccc} a_1 & a_2 & a_3 & a_4 \end{array} \\ \begin{bmatrix} 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & -1 & 1 & 1 \end{bmatrix} \end{array}$$

32-rasm. Daraxtlarda qo‘shnilik (A) va insidentlik (B) matritsalari

Daraxtlar uchun bunday matritsalarning o‘ziga xos xususiyatlarini ta’kidlaylik. $\frac{n-1}{n}$ ga teng bo‘lgan daraxtning qirralari sonining nisbati bog‘langan graf uchun minimal, shuning uchun daraxtning qo‘shnilik matritsasi juda kam (ularning nisbati va undagi nollar $(n-1):(n^2-n+1) \approx 1/n$ yo‘naltirilgan daraxt uchun va $2(n-1):(n^2-2n+2) \approx \frac{2}{n}$ yo‘naltirilmagan uchun).

Daraxtning insidentlik matritsasi n x $(n-1)$ o‘lchamiga ega, ya’ni kvadratga yaqin, va aslida, agar biz uning ortiqcha ekanligini hisobga olsak. Darhaqiqat, har qanday qatorni olib tashlab, biz avvalgidek grafni to‘liq tavsiflaydigan kvadrat matritsani olamiz.

Quyida keltirilgan insident matritsasining yana bir xususiyati quyidagicha. Satr va ustunlarni qayta tartiblash orqali har qanday daraxtning insidentlik matritsasi i ustun birliklaridan biri i qatorda, ikkinchisi pastki qatorlardan birida bo‘lganda pastki trapetsiya matritsaga tushirilishi mumkin.



a) b)

$$B_a = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{array} \begin{array}{c} \begin{array}{cccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \qquad B_b = \begin{array}{c} \\ u_1(v_2) \\ u_2(v_3) \\ u_3(v_4) \\ u_4(v_5) \\ u_5(v_7) \\ u_6(v_6) \\ u_7(v_1) \end{array} \begin{array}{c} \begin{array}{cccccc} e_1 & e_4 & e_2 & e_5 & e_6 & e_3 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

33-rasm. Daraxtlarda insidentlik matritsalari

8.3. Pryufer Kodi

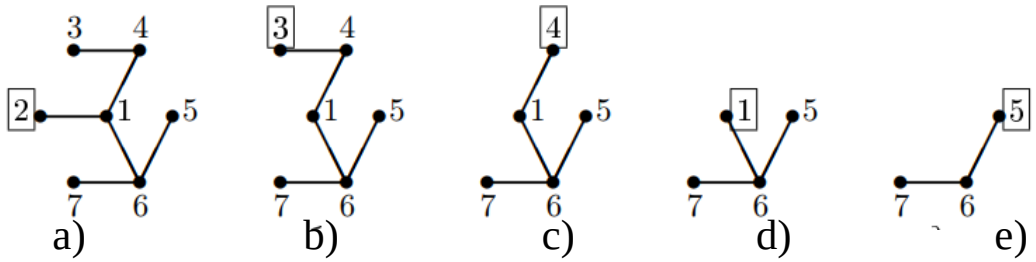
Pryufer kodi [1, n] kesmadagi n-2 butun sonlar ketma-ketligi yordamida n uchlari bilan belgilangan daraxtlarni birma-bir kodlash usuli. Ya'ni, Pryufer kodi - bu to'liq graf va raqamlar ketma-ketligining barcha daraxtlari orasidagi biyeksiyasidir.

Daraxtlarni kodlashning ushbu usuli nemis matematiki Xaynts Pryufer tomonidan 1918-yilda taklif qilingan. n ta uchlari bilan berilgan daraxt uchun Pryufer kodini qurish algoritmini ko'rib chiqaylik.

Kirishda qirralarning ro'yxati berilgan. Eng kichik raqamga ega bo'lgan daraxtning bargi tanlanadi, keyin u daraxtdan olib tashlanadi va bu barg bilan bog'langan uchlarning soni Pryufer kodiga qo'shiladi. Ushbu protsedura $n-2$ marta takrorlanadi. Oxir-oqibat, daraxtda faqat 2 ta uch qoladi va algoritm shu yerda tugaydi. Qolgan ikkita uchning raqamlari kodga yozilmaydi.

Shunday qilib, ma'lum bir daraxt uchun Pryufer kodi $n-2$ ta raqamlar ketma-ketligi bo'lib, bu yerda har bir raqam eng kichik barg bilan bog'langan uchning soni - bu segmentdagi raqam [1, n].

Pryufer kodini aniqlash.



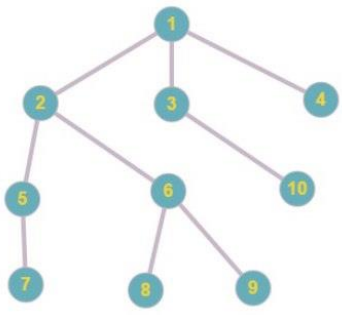
34-rasm. Daraxtlar uchun Pryufer kodini aniqlash bosqichlari

Kodni olish algoritmi quyidagicha. Daraxt tugunlari 1 dan n gacha bo‘lgan raqamlar bilan belgilangan (raqamlangan) bo‘lsin. Biz eng kichik sonli 1-darajali uchni topamiz va kodga unga qo‘shni bo‘lgan uchning sonini kiritamiz, shundan so‘ng topilgan uchni (qirrasi bilan birga) o‘chirib tashlaymiz. Olingan graf osti bilan biz xuddi shu amalni bajaramiz, uni faqat bitta qirra qolguncha takrorlaymiz. Kodni yaratish jarayoni 34-rasmda keltirilgan. Keyingi bosqichda o‘chirilgan uchning soni ramkaga kiritilgan. Berilgan grafda (34-rasm, a) birinchi darajali uchlar orasida minimal son 2-uchda joylashgan. U 1-uchga qo‘shni. Shuning uchun Pryufer kodining birinchi raqami 1. 2-uchni olib tashlash natijasida biz b-rasmda ko‘rsatilgan grafni olamiz. Ushbu grafda darajasi birga teng bo‘lgan uchlar orasidagi minimal son 3 ga teng, shuning uchun kodning ikkinchi raqami 4. Shaklda ko‘rsatilgan graflarga mos keladigan yana uchta takrorlashni bajargandan so‘ng, c, d, e-rasmlardagi bitta qirradan iborat daraxtni olamiz {7; 6}. Jarayon tugallandi.

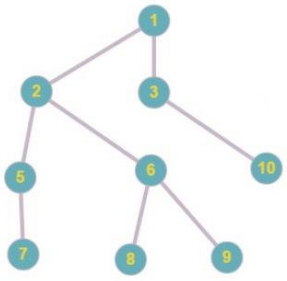
Qabul qilingan qadamlarning natijalari jadvalda keltirilgan. Oxirgi qatorida kerakli kod mavjud - 14166.

Qadam		1	2	3	4	5
34-rasm		a	b	c	d	e
Minimal raqam		2	3	4	1	5
O‘chirilgan qirra		{1;2}	{4;3}	{1;4}	{6;1}	{6;5}
Pryufer kodi		1	4	1	6	6

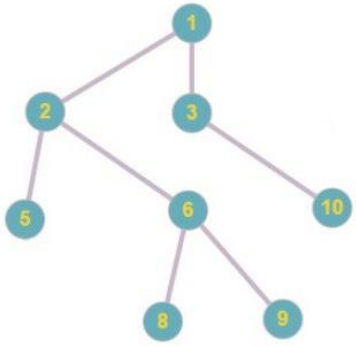
2-misol. 35-rasmda berilgan daraxtning Pryufer kodini topish qadamlari 35-a,b,c,d,e,f,g,h rasmlarda berilgan.



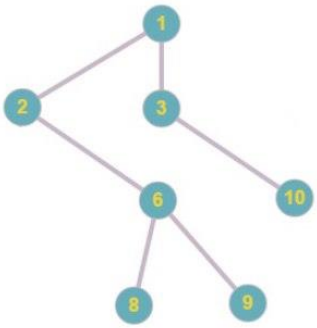
35-rasm. Daraxtning dastlabki berilishi



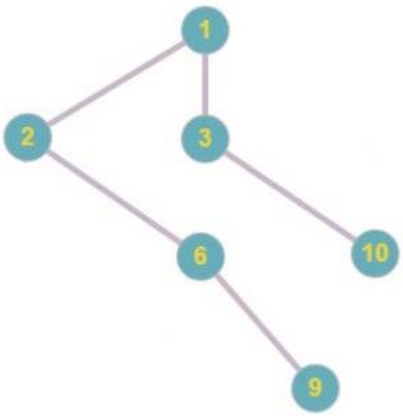
a) Pryufer kodi: 1



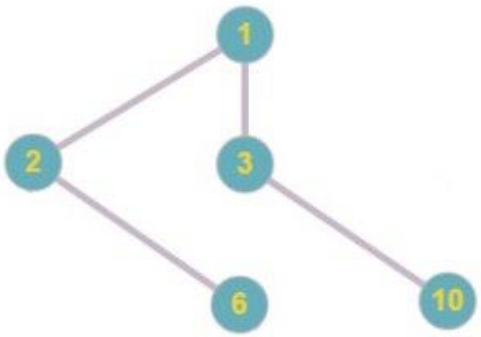
b) Pryufer kodi: 1 5



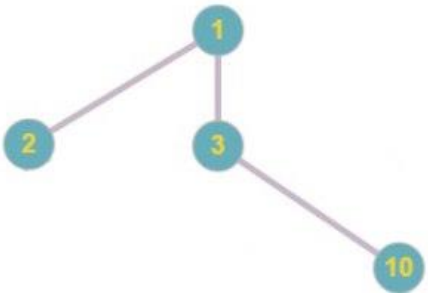
c) Pryufer kodi: 1 5 2



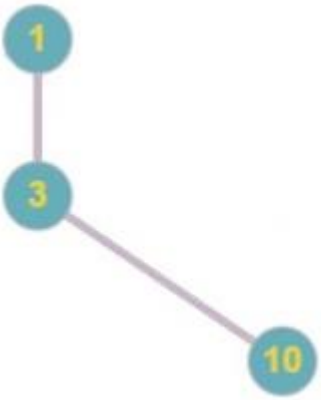
d) Pryufer kodi: 1 5 2 6



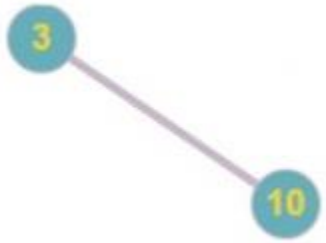
e) Pryufer kodi: 1 5 2 6 6



f) Pryufer kodi: 1 5 2 6 6 2



g) Pryufer kodi: 1 5 2 6 6 2 1



h) Pryufer kodi: 1 5 2 6 6 2 1 3

Pryufer kodi orqali daraxtni tiklash. Pryufer kodi bilan ifodalangan daraxtlarni hosil qilish algoritmi qirralarning tegishli roʻyxatini olishga imkon beradi.

Antikodni Pryufer kodiga kiritilmagan uchlar sonining ortib boruvchi ketma-ketligi deylik. Koʻrib chiqilgan misol uchun antikod 2357 ga teng.

Daraxt ketma-ket qirralarni qoʻshib quriladi. Keyingi qoʻshilgan chekka, birinchisidan boshlab, vertikal juftlik bilan hosil boʻladi, ularning raqamlari kod satrida va antikod satrida birinchi boʻladi. Shundan soʻng, ishlatilgan satr elementlari chiziladi. Agar kod satridan chiqib ketgan raqam undagi qolgan elementlar qatoriga kiritilmagan boʻlsa, uning tartibini buzmasdan antikod qatoriga qoʻshilishi kerak. Ta'riflangan harakatlar kod va antikod satrlarining "qoldiqlari" bilan ularning birinchisining barcha elementlari oʻchirilguncha takrorlanadi. Bunday holda, antikod chizigʻida hosil qilingan roʻyxatga qoʻshiladigan soʻnggi chekkani belgilaydigan ikkita element boʻladi, natijada biz Pryufer kodi bilan belgilangan daraxtga mos keladigan n - 1 qirralarning roʻyxatini olamiz.

3-misol. Masalan, 1-misolda berilgan 14166 kodi yordamida daraxtni tiklaylik. Yuqorida 1-misolda koʻrsatilgandek mos keladigan antikod 2357 ni tashkil qiladi. Shuning uchun daraxtning birinchi qirrasi {1; 2}. 1 va 2-ni kesib oʻtib, biz kod satrida 4166 va antikod satrida 357 olamiz. Keyingi takrorlashda {4; 3} juftligini kesib tashlaymiz va qatorga antikod 4 ni kiritamiz va hokazo. Takrorlashlar ketma-ketligi 8-jadvalda keltirilgan.

8-jadval.

Pryufer kodi orqali daraxtni tiklash ketma-ketligi

Qadam	1	2	3	4	5	6	
Kod satri	~~1~~	~~4~~	~~1~~	~~6~~	6		
Antikod satri	~~2~~	3 ~~3~~	5 5 ~~4~~	7 7 5 ~~1~~	 7 5 ~~5~~	 7 7 6	 7
Qirra qo‘shish	{1;2}	{4;3}	{1;4}	{6;1}	{6;5}	{6;7 }	

Qirralarning ro‘yxatini tahlil qilib, asl daraxt olinganligiga ishonch hosil qilamiz. E’tibor bering, qirralarning tartibi avvalgi jadvaldagi kabi.

4-misol. Pryufer kodini yaratish vazifasining oldida kodlangan daraxtni tiklash vazifasi ham mavjud. Daraxtlarni qayta qurish algoritmini quyidagi shartlar bilan ko‘rib chiqamiz: kirish sifatida Pryufer kodini ifodalovchi raqamlar (uchlar) ketma-ketligi, natijada daraxt qirralarining ro‘yxati bo‘ladi.

Kod hal qilish algoritmini batafsil ko‘rib chiqamiz. Koddan tashqari, bizga grafning barcha uchlari ro‘yxati kerak. Biz bilamizki, Pryufer kodi n-2 ta uchlardan iborat, bu yerda n - grafdagi uchlar soni. Ya’ni kodlangan daraxtdagi uchlar sonini kodning kattaligi bo‘yicha aniqlashimiz mumkin.

Natijada, algoritmning boshida bizda Pryuferning $n-2$ o‘lchamdagi kodlari va grafdagi barcha uchlar qatori mavjud: [1 ... n]. Keyin quyidagi protsedura $n-2$ marta takrorlanadi: Pryufer kodini o‘z ichiga olgan massivning birinchi elementi olinadi va kod bilan massivda bo‘lmagan eng kichik uchni qidirish daraxt uchlari bilan massivda amalga oshiriladi. Topilgan uch va Pryufer kodi bilan massivning joriy elementi daraxtning qirrasini tashkil qiladi. Ushbu uchlar tegishli massivlardan olib tashlanadi va yuqoridagi protsedura kodli qator elementlari tugamaguncha takrorlanadi. Algoritm oxirida graf uchlari

bilan massivda ikkita uch qoladi; ular daraxtning so‘nggi uchini tashkil qiladi. Natijada biz grafning kodlangan barcha qirralarining ro‘yxatini olamiz.

2-misolda olingan Pryufer kodi yordamida daraxtni tiklaylik.

Birinchi qadam
Pryufer kodi: 1 5 2 6 6 2 1 3
Daraxtlar uchlari massivi: 1 2 3 4 5 6 7 8 9 10
Pryufer kodida mavjud bo‘lmagan minimal uch 4 ga teng
Qirralar ro‘yxati: 1 4

Ikkinchi qadam
Pryufer kodi: 1 5 2 6 6 2 1 3
Daraxtlar uchlari massivi: 1 2 3 4 5 6 7 8 9 10
Pryufer kodida mavjud bo‘lmagan minimal uch 7 ga teng
Qirralarning ro‘yxati: 1 4, 5 7

Uchinchi qadam
Pryufer kodi: 1 5 2 6 6 2 1 3
Daraxtlar uchlari massivi: 1 2 3 4 5 6 7 8 9 10
Pryufer kodida mavjud bo‘lmagan minimal tepalik 5 ga teng
Qirralarning ro‘yxati: 1 4, 5 7, 2 5

To‘rtinchi qadam
Pryufer kodi: 1 5 2 6 6 2 1 3
Daraxtlar uchlari massivi: 1 2 3 4 56 7 8 9 10
Pryufer kodida mavjud bo‘lmagan minimal tepalik 8 ga teng
Qirralarning ro‘yxati: 1 4, 5 7, 2 5, 6 8

Beshinchi qadam
Pryufer kodi: 1 5 2 6 6 2 1 3
Daraxtlar uchlari massivi: 1 2 3 4 56 7 8 9 10
Pryufer kodida mavjud bo‘lmagan minimal vertex 9 ga teng
Qirralarning ro‘yxati: 1 4, 5 7, 2 5, 6 8, 6 9

Oltinchi qadam
Pryufer kodi: 1 5 2 6 6 2 1 3
Daraxtlar uchlari massivi: 1 2 3 4 56 7 8 9 10
Pryufer kodida mavjud bo‘lmagan minimal vertex 6 ga teng
Qirralarning ro‘yxati: 1 4, 5 7, 2 5, 6 8, 6 9, 2 6

Yettinchi qadam
Pryufer kodi: 1 5 2 6 6 2 1 3
Daraxtlar uchlari massivi: 1 2 3 4 5 6 7 8 9 10
Pryufer kodida mavjud bo‘lmagan minimal tepalik 2 ga teng
Qirralarning ro‘yxati: 1 4, 5 7, 2 5, 6 8, 6 9, 2 6, 1 2

Sakkizinchi qadam
Pryufer kodi: 1 5 2 6 6 2 1 3
Daraxtlar uchlari massivi: 1 2 3 4 5 6 7 8 9 10
Pryufer kodida mavjud bo‘lmagan minimal tepalik 1 ga teng
Qirralarning ro‘yxati: 1 4, 5 7, 2 5, 6 8, 6 9, 2 6, 1 2, 3 1

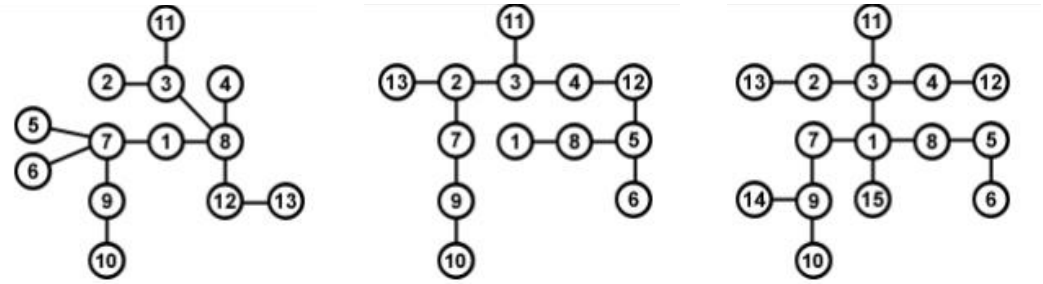
Algoritmni yakunlash
Pryufer kodi: 1 5 2 6 6 2 1 3
Daraxtlar uchlari massivi: 1 2 3 4 5 6 7 8 9 10
Pryufer kodida mavjud bo‘lmagan minimal tepalik 1 ga teng
Qirralarning ro‘yxati: 1 4, 5 7, 2 5, 6 8, 6 9, 2 6, 1 2, 3 1, 3 10

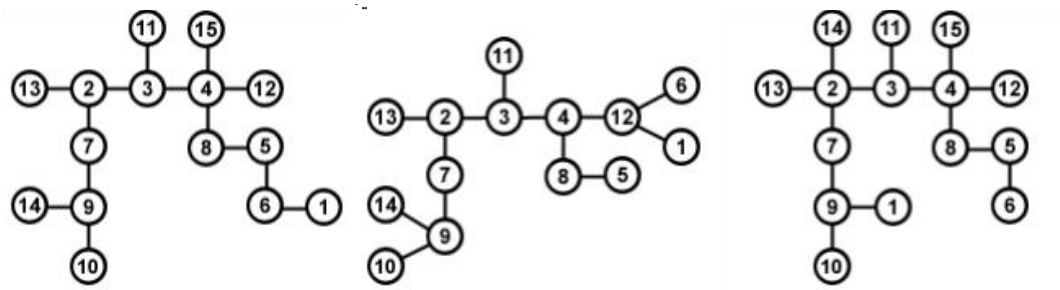
Mavzu yuzasidan savollar:

1. Daraxt ma’lumotlar strukturasiga ta’rif bering
2. Daraxtning eng asosiy tushunchalariga to‘xtalib o‘ting.
3. Pryufer kodini hosil qilish va qo‘lllanishi haqida gapiring
4. Pryufer kodi asosida daraxtni tiklash qanday amalga oshiriladi?
5. Daraxt ma’lumotlar strukturasi qo‘llaniladigan sohalarga qaysilar kiradi?

Mustaqil ishlash uchun masalalar:

1) Quyidagi daraxtlarning pryufer kodini toping.





2) Quyidagi Pryufer kodi berilgan. Ushbu kodga ko‘ra daraxtlarni hosil qiling.

(2, 2, 7, 2, 11, 11, 7, 7, 6, 9, 4, 5)

(1, 1, 7, 6, 13, 1, 7, 12, 6, 9, 4, 5, 3)

(1, 2, 8, 3, 1, 10, 1, 1, 6, 5, 3, 2, 9)

(2, 5, 7, 12, 10, 11, 7, 7, 6, 9, 4, 5)

(12, 2, 1, 1, 1, 1, 3, 3, 4, 1, 2, 3, 8, 9)

9-§. Tartiblangan va muvozanatlashgan daraxtlar

9.1. AVL daraxti

AVL daraxt. AVL-daraxt (inglizcha AVL-Tree) bu muvozanatlashgan ikkilik qidiruv daraxti bo‘lib, unda quyidagi xususiyat qo‘llab-quvvatlanadi: uning har bir uchlari uchun uning ikkita ostki daraxtining balandligi 1 dan ko‘p bo‘lmagan qiymatga farq qiladi.

AVL daraxtlari birinchi marta 1962-yilda AVL daraxtlaridan foydalanishni taklif qilgan G. M. Adelson-Velskiy va E. M. Landisning ismlarining birinchi harflari bilan nomlangan.

Uchlarni muvozanatlash - bu $|h(L) - h(R)| = 2$ chap va o‘ng pastki daraxtlari balandliklari farqi bo‘lgan taqdirda, $|h(L) - h(R)| \leq 2$ daraxtining xususiyati tiklanishi uchun ushbu uchlarning pastki daraxtidagi ajdod va avlod munosabatlarini o‘zgartiradigan amal, aks holda hech narsani o‘zgartirilmaydi. Muvozanatlash uchun biz har bir uch uchun uning chap va o‘ng $diff(i) = h(L) - h(R)$ pastki daraxtlari balandliklari orasidagi farqni saqlaymiz.

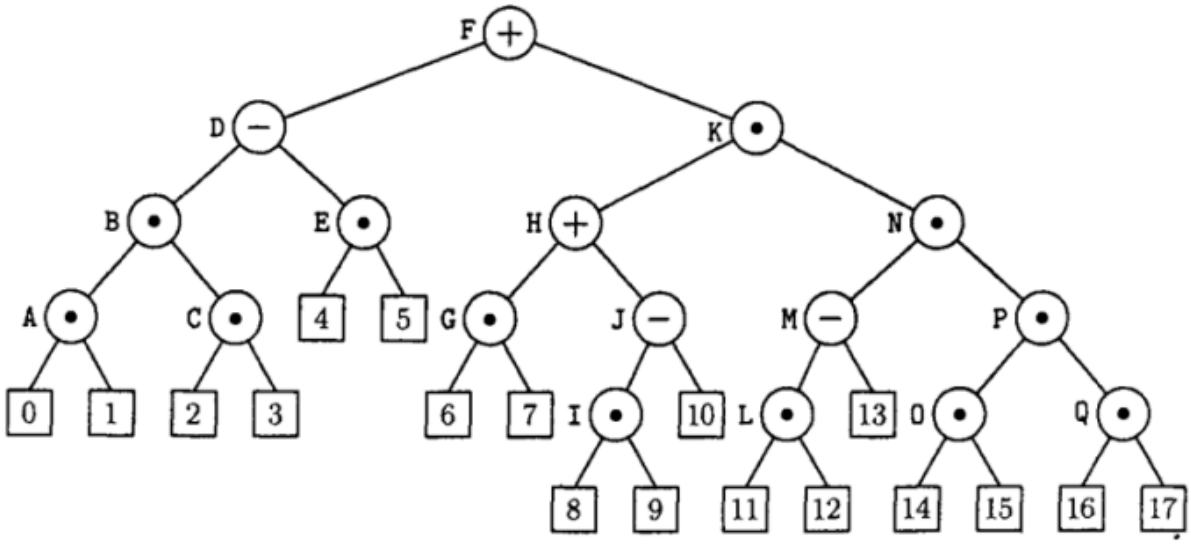
Daraxtning balandligi uning maksimal darajasi, ildizdan tashqi tugunga qadar eng uzun yo‘lning uzunligi sifatida aniqlanadi. Ikkilik qidiruv daraxti muvozanatli deyiladi, agar biron bir tugunning chap pastki daraxtining balandligi o‘ng pastki daraxtning balandligidan ± 1 dan oshmasa. Keyingi 36-rasmda ko‘rsatilgan 5 ta balandlikdagi 17 ta ichki tugunli muvozanatli daraxt; muvozanat koeffitsiyenti har bir tugun ichida belgilar bilan va o‘ng va chap pastki daraxtlar (+1, 0 yoki -1) balandliklari orasidagi farq kattaligiga muvofiq belgilanadi.

Muvozanatlashgan daraxtlar haqidagi teorema. Adelson-Velskiy va Landis quyidagi teoremani isbotladilar:

Teorema. n ichki tugunli muvozanatli daraxt balandligi $\lg(n + 1)$ va $1.4405 \lg(n + 2) - 0.3277$ qiymatlar bilan chegaralangan.

Shunday qilib, biz AVL-muvozanatlangan daraxtdagi qidirish yo‘li mukammal muvozanatlangan daraxtdagi qidirish yo‘lidan 45% dan oshmaydi degan xulosaga kelishimiz mumkin.

AVL daraxtiga yangi tugun kiritilganda paydo bo‘ladigan variantlarni ko‘rib chiqaylik:



36-rasm. AVL ikkilik daraxtiga ko‘ra muvozanatlash

• hl=hR. Yoqilgandan so‘ng, L va R har xil balandliklarga aylanadi, ammo muvozanat mezonlari buzilmaydi;

• hL<hR. Yoqilgandan so‘ng, L va R balandlikda teng bo‘ladi, ya’ni muvozanat mezonlari yanada yaxshilanadi;

• hL>hR. Yoqilgandan so‘ng muvozanat mezonlari buziladi va daraxtni qayta qurish kerak.

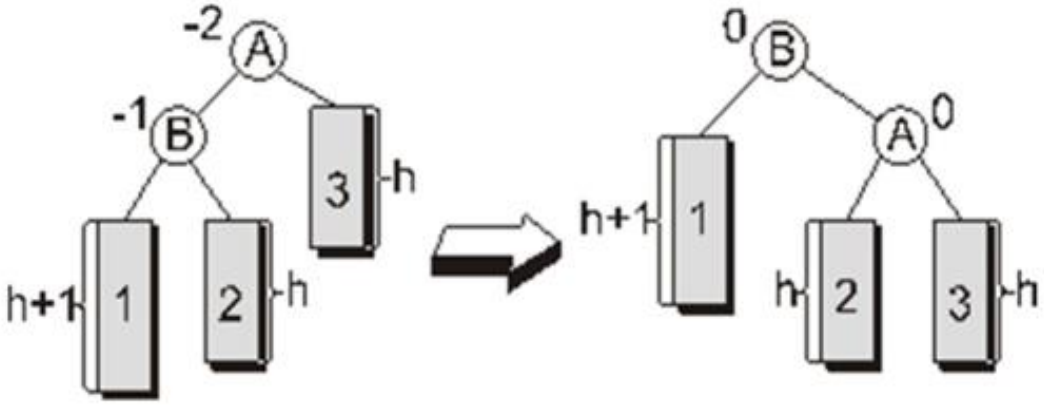
Shunday qilib, biz AVL daraxtiga yangi tugunni kiritish uchun umumiy algoritmni tuzamiz:

1. Kiritilgan daraxtning ichida emasligiga ishonch hosil qilish uchun daraxtni aylanib o‘tish;

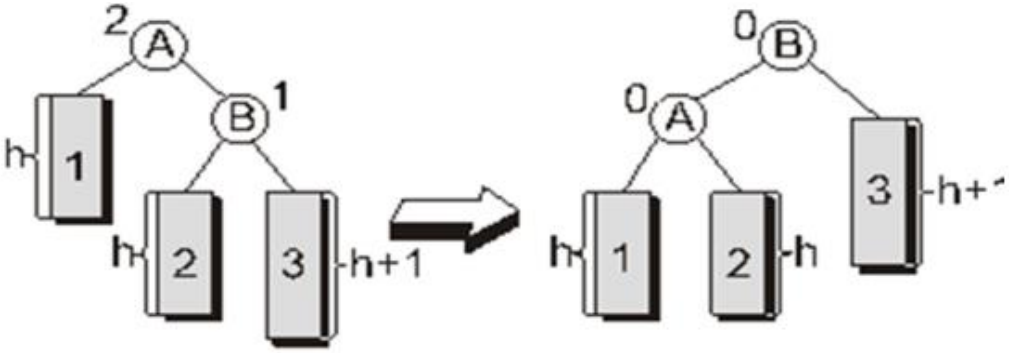
2. Yangi uchni kiritish va natijada balans ko‘rsatkichini aniqlash;

3. Qidiruv yo‘li bo‘ylab "orqaga chekinish" va har bir uchda balans ko‘rsatkichini tekshirish. Agar kerak bo‘lsa, muvozanatni saqlash.

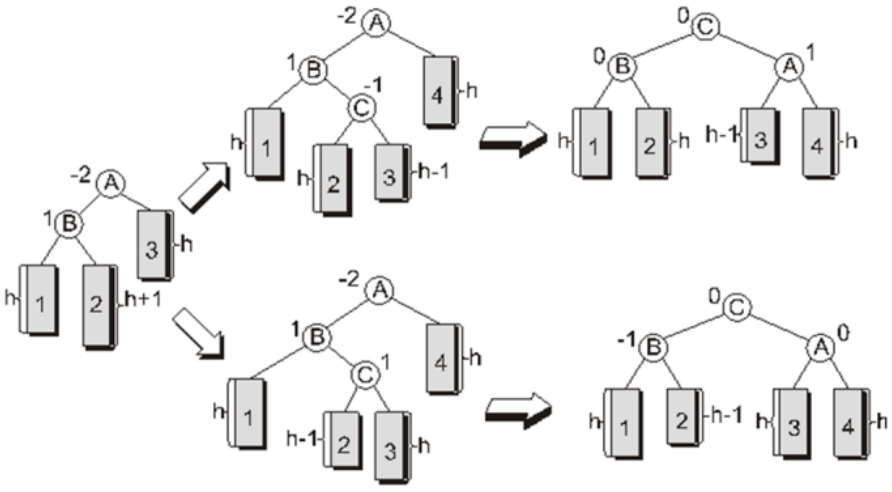
Amalda AVL balansini tiklashning 4 algoritmi qo‘llaniladi: muvozanat ko‘rsatkichlari qiymatiga qarab tanlangan kichik va katta chap burilish, kichik va katta o‘ng burilish (37-40-rasmlar)



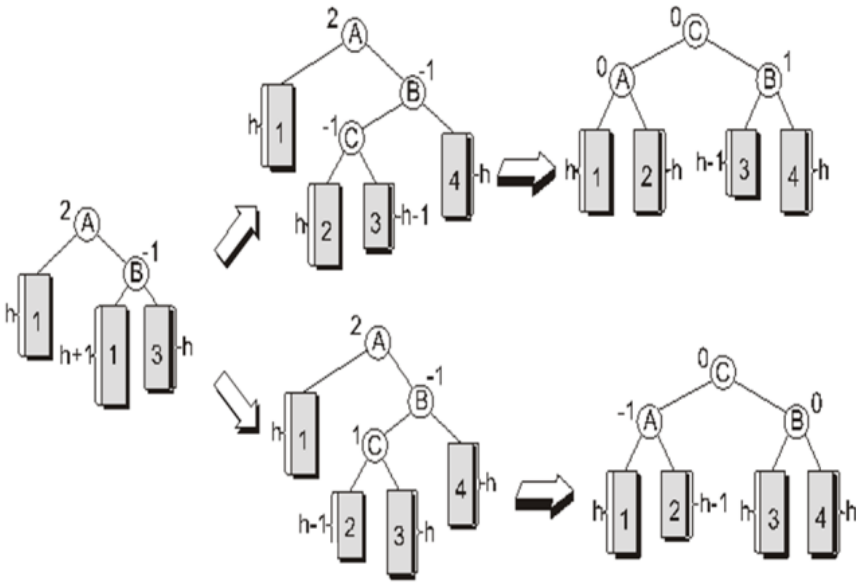
37-rasm. Kichik chap burilish algoritmi



38-rasm. Kichik o‘ng burilish algoritmi

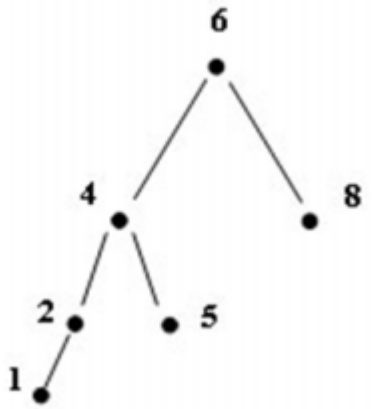


39-rasm. Katta chap burilish algoritmi



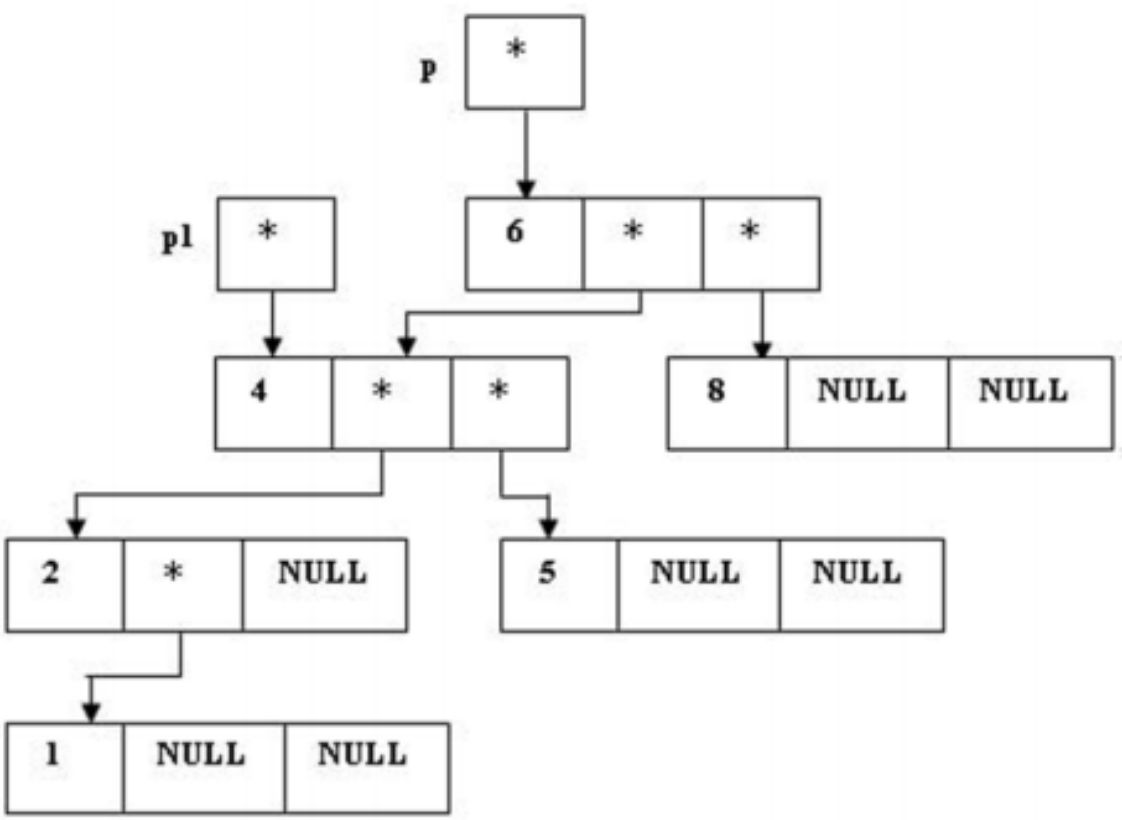
40-rasm. Katta o‘ng burilish algoritmi

Rasmlarda to‘rtburchaklar pastki daraxtlarni bildiradi, ichidagi raqamlar kichik daraxtlarning raqamlari, tugunlar yonidagi raqamlar balans ko‘rsatkichlari. Balanslash algoritmi chap tomonga burilishning quyidagi misolida keltirilgan.



41-rasm. Daraxtning dastlabki berilishi

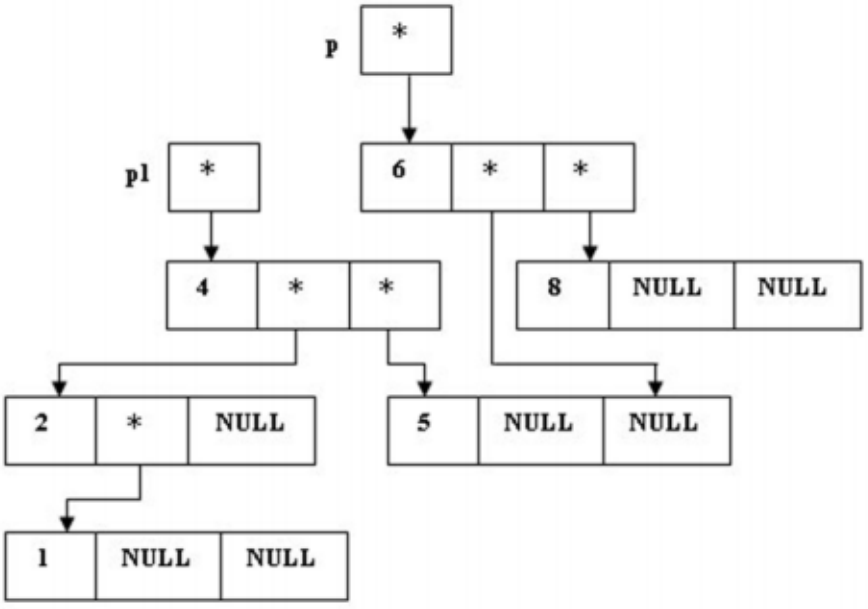
1. Daraxtning ildiziga aylanadigan uchining manzilini aniqlash:
2. P1=(*p).Left;



42-rasm. Yangi daraxt ildizining manzilini saqlash

3. "Yangi" ildizdan o‘ng pastki daraxtni qayta ulang, ushbu daraxtni "eski" ildizning chap pastki daraxtiga aylantiring:

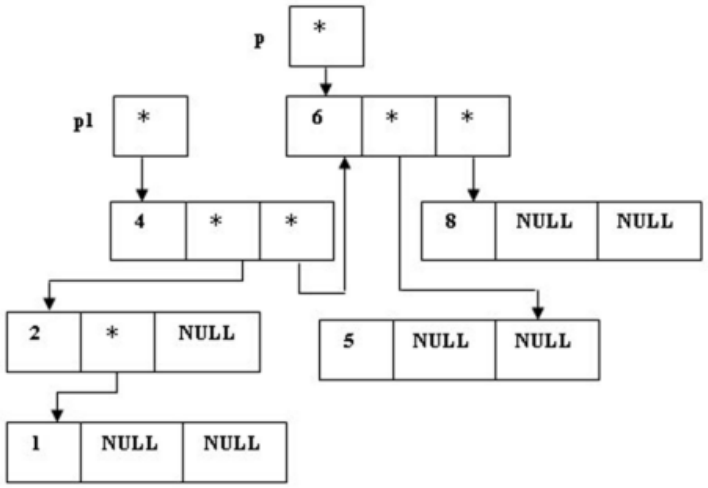
4. (*p).Left = (*p1).Right;



43-rasm. Qayta biriktirish

5. "Yangi" ildizning o‘ng pastki daraxtini "eski" ildizdan boshlanganligini aniqlash:

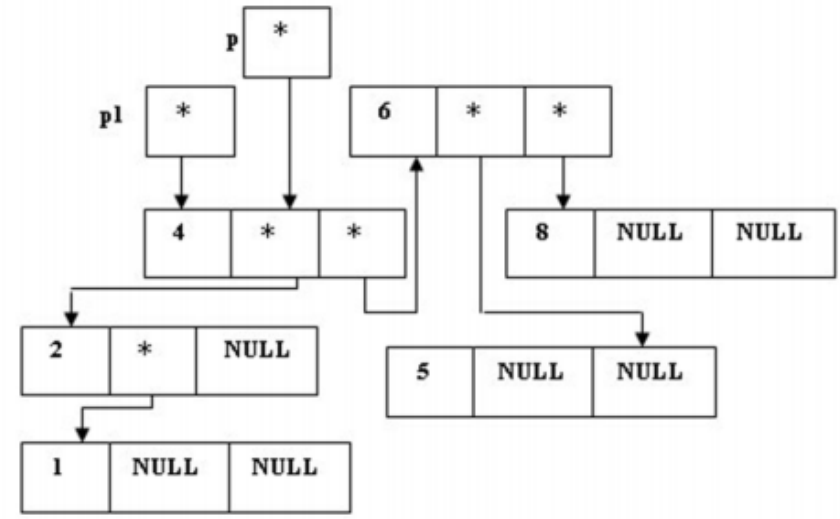
6. (*p1).Right = p;



44-rasm. “Yangi” ildizning o‘ng pastki daraxtini aniqlash

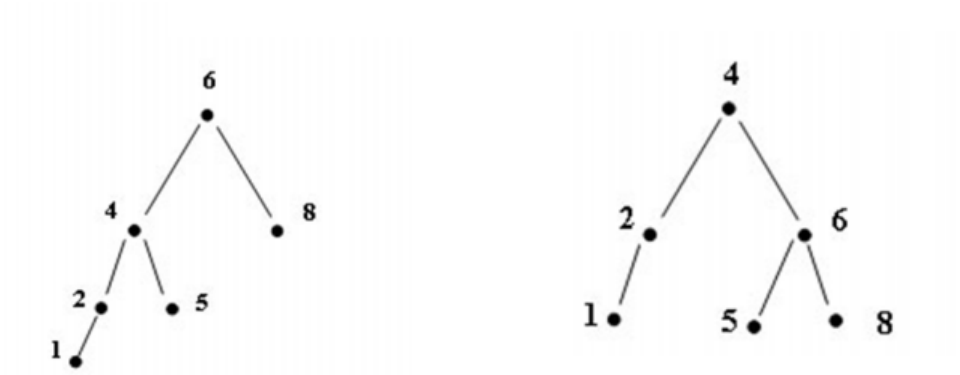
7. Ko‘rsatkichning qiymatini daraxtning ildiziga o‘zgartiring (p) va balans qiymatini tiklang:

8. (*p).bal=0; p=p1;



45-rasm. "Yangi" ildizning o‘ng pastki daraxtini aniqlash

Muvonazatlash algortmidan so‘ng, AVL bo‘yicha muvozanatlashgan quyidagi daraxt hosil bo‘ldi:



a) Dastlabki daraxt **b) Muvozanatlashgan daraxt**

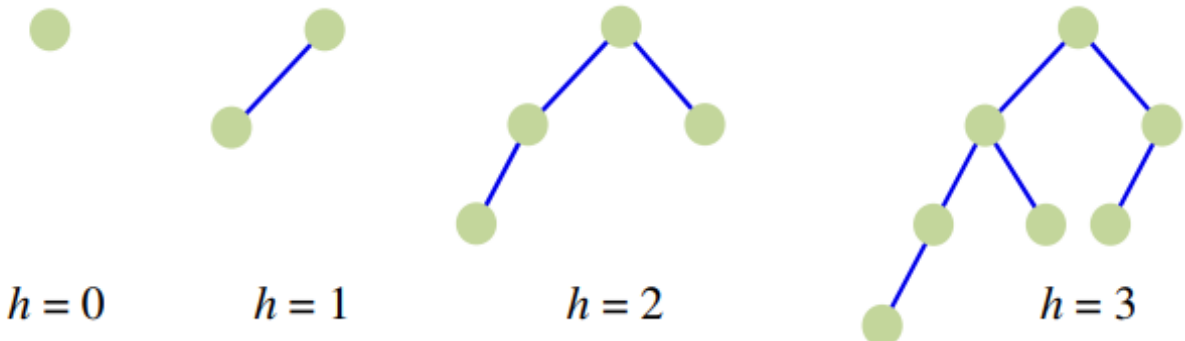
9.2. AVL daraxtlarining samaradorligini tahlil qilish

N elementni o‘z ichiga olgan AVL daraxtining balandligini yuqoridan baholaylik.

h balandlikdagi AVL daraxtini hosil qilish uchun zarur bo‘lgan minimal tugunlarni N (h) bilan belgilaymiz.

N(-1)=0, N(0)=1, N(1)=2, N(2)=4, N(3)=7, …

0,1, 2, 4, 7, 12, 20, 33, 54, …



$$N(h) = N(h-1) + N(h-2) + 1$$

- $N(h)$: 1, 2, 4, 7, 12, 20, 33, 54, …
- Fibonacci(h): 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

$$\boldsymbol{N(h) = F(h+3) - 1}, \text{ для } h \geq 0$$

Fibonachchi ketma-ketligining h –hadi uchun Binet formulasidan

$$F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}} = \frac{\varphi^n - (-\varphi)^{-n}}{\varphi - (-\varphi)^{-1}} = \frac{\varphi^n - (-\varphi)^{-n}}{2\varphi - 1}$$

$\varphi = \frac{1+\sqrt{5}}{2}$ – Oltin nisbat

$$F_n \sim \frac{\varphi^n}{\sqrt{5}}$$

AVL daraxtining h (n) balandligi uchun yuqori chegara:

$$\log(n+1) \leq h(n) \leq 1.4404 \cdot \log(n+2) - 0.328$$

AVL daraxtidan tugunlarni olib tashlash

```
#include <iostream>
using namespace std;
struct avltree
{

   int key;
   char *value;
   int height;
   struct avltree *left;
   struct avltree *rigth;
};
```

AVL daraxtidan barcha tugunlarni olib tashlash funksiyasi

```
void avltree_free (struct avltree *tree)
{
   if (tree == NULL)
      return;
   avltree_free(tree->left);
   avltree_free(tree->rigth);
   free(tree);
}
```

Tugundan kalit bo‘yicha izlash funksiyasi

```
struct avltree *avltree_lookup(struct avltree *tree, int key)
{
   while(tree !=NULL)
   {
      if(key == tree->key)
      {
         return tree;
      }
      else
      if(key<tree->key)
      {
         tree = tree->left;
      }
      else
      {
        tree = tree->rigth;
      }
   }
};
```

Tugun hosil qilish funksiyasi

```
struct avltree *avltree_create(int key,char *value)
{
   struct avltree *node;
   node = malloc(sizeof(*node));
   if (node != NULL)
   {
      node->key = key;
      node->value = value;
      node->left = NULL;
      node->right = NULL;
      node->height = 0;
   }
   return node;
}
```

Daraxtni ekranda chiqarish funksiyasi

```
void avltree_print_dfs(struct avltree *tree, int level)
{
   int i;
   if (tree == NULL)
      return;
   for (i = 0; i < level; i++)
      printf("  ");
   printf("%d\n", tree->key);
   avltree_print_dfs(tree->left, level + 1);
   avltree_print_dfs(tree->right, level + 1);
}
int main()
{
   struct avltree *tree = NULL;
   tree = avltree_add(tree, 5,    "5");
   tree = avltree_add(tree, 3,    "3");
   /* Code */
   avltree_free(tree);
   return 0;
}
```

Mavzu yuzasidan savollar:

1. AVL daraxti nima?
2. Uchlarni muvozanatlash deganda nimani tushunasiz.
3. Tugundan kalit bo‘yicha izlash funksiyasini tushuntirib bering.
4. Muvozanatlashgan daraxt tushunchasi nima?
5. Daraxt ma’lumotlar strukturasi qo‘llaniladigan sohalarga qaysilar kiradi?

Mustaqil ishlash uchun masalalar:

1. AVL daraxtida tugun olib tashlash funksiyasini yozing va uni daraxtda qo‘llang.
2. Kalit bo‘yicha izlash funksiyasini optimallashtiring.

10-§. B daraxtlar

10.1. B daraxt ta’rifi

B daraxti (inglizcha *B-tree*) – izlash, qo'shish va o'chirish imkonini beradigan, juda ko‘pshoxli muvozanatlashgan qidiruv daraxti. Tugunlari *n* bo'lgan B daraxti O(logn) balandlikka ega bo‘ladi. Tugun shoxlari soni bittadan bir necha minggacha bo'lishi mumkin (odatda, B-daraxtining shoxlanish darajasi daraxt ishlaydigan qurilma (disklar) xususiyatlari bilan belgilanadi). B-daraxtlar O(logn) ko'p dinamik to'plam amallarini o'z vaqtida bajarish uchun ham ishlatilishi mumkin.

B daraxti birinchi marta 1970-yilda R. Bayer va E. Makkreyt tomonidan taklif qilingan.

B daraxt strukturasi. B daraxti mukammal muvozanatlashgan, ya'ni uning barcha barglarining chuqurligi bir xil. B daraxti quyidagi xususiyatlarga ega (t – bu daraxt parametrlari, B daraxtining *minimal darajasi* deyiladi, 2 dan *kam* emas):

- Ildizdan tashqari har bir tugun hech bo'lmaganda *t-1* kalitni o'z ichiga oladi va har bir ichki tugun kamida avlodli *t* tugunlarga ega. Agar daraxt bo'sh bo'lmasa, ildizda kamida bitta kalit bo'lishi kerak.
- Har bir tugun, ildizdan tashqari, ichki tugunlarda ko'pi bilan *2t - 1* kalitni va ko'pi bilan *2t* avlodni o'z ichiga oladi
- Ildizda daraxt bo'sh bo'lmasa bittadan *2t-1* gacha kalit va balandligi 0 dan katta bo'lsa 2 dan 2t gacha avlodni o'z ichiga oladi.
- Daraxtning har bir tugunida k_1, ..., k_n kalitlari bo'lgan barglardan tashqari n + 1 avlodlari bor. i-avlodda $[k_{i-1}; k_i]$, $k_0 = -\infty$, $k_{n+1} = \infty$ kesmaning kalitlari mavjud.
- Har bir tugunning kalitlari kamaymaydigan tartibda tartiblangan.
- Barcha barglar bir xil darajada.

B daraxtlar disklarda (fayl tizimlarida) yoki boshqa to'g‘ridan-to'g‘ri kiruvchi bo'lmagan saqlash muhitlarida, shuningdek, ma'lumotlar bazalarida foydalanish uchun mo'ljallangan. B daraxtlar qizil-qora daraxtlarga o'xshaydi, lekin ular diskni o'qish/yozish amallarini minimallashtirishda yaxshiroq hisoblanadi.

Daraxtlar - bu dinamik to'plam amallarni bajaradigan ma'lumotlar tuzilmalari. Bunday amallar sifatida elementni qidirish, minimal (maksimal) elementni qidirish, kiritish, o'chirish, avlod-ajdodga o'tish, avlodga o'tish kabilarni keltirish mumkin. Shunday qilib, daraxt oddiy lug‘at sifatida ham, ustivor navbat sifatida ham ishlatilishi mumkin. Daraxtlardagi asosiy amallar uning balandligiga mutanosib vaqtda bajariladi. Muvozanatlashgan daraxtlar ularning balandligini kamaytiradi (masalan, tugunlari bo'lgan ikkilik muvozanatli daraxtning balandligi *logn*).

Bu standart qidiruv daraxtlarida qanday muammo bor? Oldingi mavzularda aytib o'tilgan daraxtlardan biri tasvirlangan ulkan ma'lumotlar bazasini ko'rib chiqaylik. Shubhasiz, bu daraxtlarning barchasini tezkor xotirada saqlay olmaymiz – ma'lumotlarning faqat bir qismini saqlaymiz, qolganlari uchinchi tomon vositasida saqlanadi (masalan, kirish tezligi ancha past bo'lgan qattiq diskda). Qizil-qora yoki Dekart kabi daraxtlar uchinchi tomon vositalariga kirishni talab qiladi. Katta *n* lar uchun bu juda ko'p. Aynan mana shu muammo B daraxtlar hal qilish imkonini beradi.

B daraxtlari ham muvozanatlashgan daraxtlardir, shuning uchun standart amallarni bajarish uchun vaqt balandlikka mutanosib. Ammo, boshqa daraxtlardan farqli o'laroq, ular maxsus disk xotirasi bilan ishlash uchun yaratilgan (oldingi misolda - uchinchi tomon tarqatuvchi), aniqrog‘i ular kirish -chiqish turidagi murojaatlarni kamaytiradi.

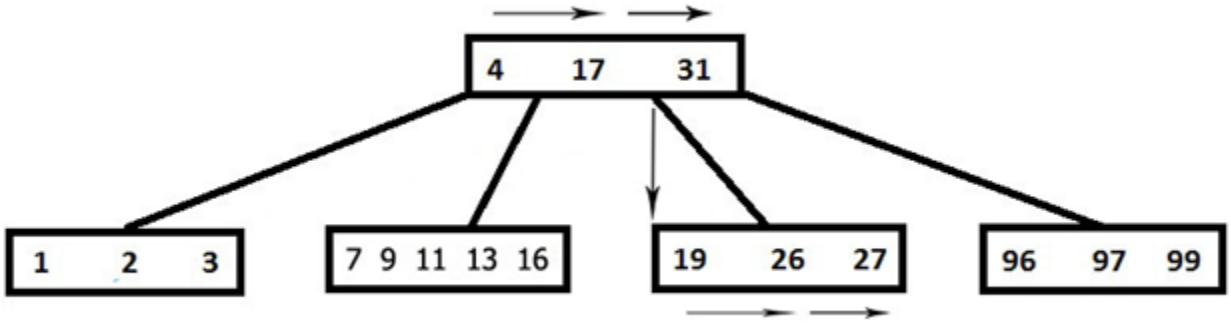
10.2. B daraxtda amallar

B daraxtda izlash algoritmi. Yuqorida aytib o'tilganidek, B daraxti barcha standart qidirish, qo'shish, o'chirish va hokazo amallarni bajaradi.

B daraxtida izlash binar daraxtni qidirishga juda o'xshaydi, faqat bu yerda biz avlodga yo'lni 2 variantdan emas, balki bir nechta variantdan tanlashimiz kerak. Aks holda, farqi bo'lmay qoladi. Quyidagi 46-rasmda 27-kalitni qidirish ko'rsatilgan. Tasvirni ko‘rib chiqaylik (va shunga mos ravishda standart qidirish algoritmi):

- Biz ildiz kalitlarini kerak bo'lguncha o'tamiz. Bu holda 31 ga yetdik.

- Bu kalitning chap tomonidagi avlodga tushamiz.
- 27 dan kichik bo‘lgunga qadar yangi tugunni kalit bo‘yicha izlaymiz. Bunday holda, biz 27 ni topdik va to'xtadik.

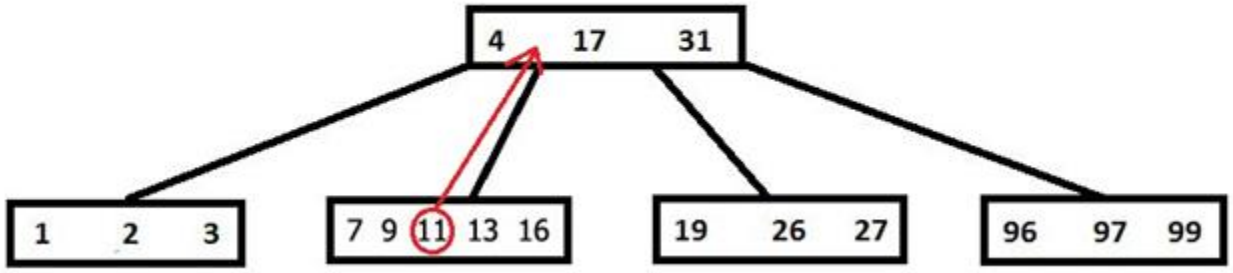


46-rasm. a) B daraxtda izlash algoritmining bajarilishi

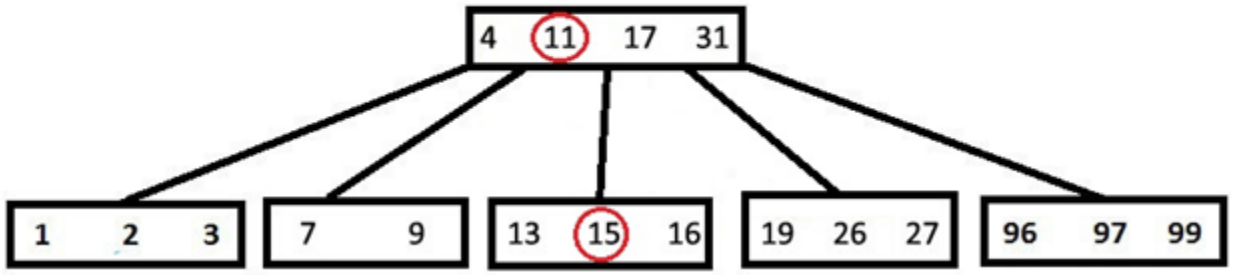
Izlash amali O(t*logt n) vaqtida bajariladi, bu yerda *t* - minimal daraja. Bu yerda disk amallarini faqat O (logt n) da bajarishimiz muhim qismidir.

B daraxtlarda element qo‘shish. Izlashdan farqli o'laroq, qo'shish usuli ikkilik daraxtga qaraganda ancha murakkab, chunki yangi barg yaratish va unga kalit qo'yish mumkin emas, chunki B daraxtining xususiyatlari buziladi. Kalitni allaqachon to'ldirilgan bargga kiritish mumkin emas. Tugunni ikkiga bo'lish amali kerak, agar barg to'ldirilgan bo'lsa, unda *2t-1* kalit bor edi. 2 ga *t-1* ga bo'lamiz undan kam kalitlar va oxirgi t-1 ajdod tuguniga o'tkaziladi. Shunga ko'ra, agar avlod-ajdod tuguni ham to'lgan bo'lsa, yana bo'lishimiz kerak va shunga o'xshash ildizgacha (agar ildiz ajralgan bo'lsa, unda yangi ildiz paydo bo'ladi va daraxt chuqurligi oshadi). Oddiy ikkilik daraxtlar singari, joylashtirish ham ildizdan barggacha bir o'tishda amalga oshiriladi.Har bir iteratsiyada (yangi kalit uchun pozitsiyani qidirishda - ildizdan barggacha) o'tadigan barcha to'ldirilgan tugunlarni ajratamiz (bargni ham o'z ichiga olgan holda). Shunday qilib, agar natijada tugunni ajratish zarur bo'lsa, uning avlod-ajdodi to'ldirilmaganligiga ko‘rishimiz mumkin.

Quyidagi 47-rasmda xuddi o'sha daraxt qidirilmoqda (t = 3). Faqat hozir biz "15" kalitini qo'shamiz. Yangi kalitning o'rnini qidirib,



a)



b)

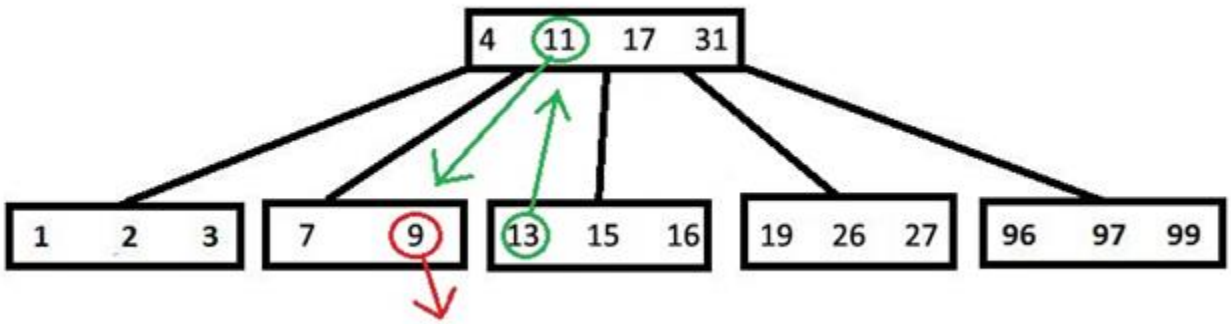
47-rasm. B daraxtlarda element qo‘shish algoritmining bajarilishi

tugallangan tugunga duch kelamiz (7, 9, 11, 13, 16). Algoritmga amal qilib, uni ajratdik - bu holda "11" avlod-ajdod tuguniga o'tadi va manba 2 ga bo'linadi. Keyin "15" tugmasi ikkinchi "ajratish" tuguniga kiritiladi. Bu holda B daraxtining barcha xususiyatlari saqlanib qolgan.

Element qo'shish amali ham O (t logt n) vaqtda bajariladi. Shunga qaramay, biz diskdagi amallarni faqat O (h) da bajaramiz, bu yerda h - daraxt balandligi.

Element o‘chirish. Kalitni B daraxtidan olib tashlash, unga element qo‘shishdan ham murakkab hisoblanadi. Buning sababi shundaki, ichki tugunni olib tashlash daraxtni umuman qayta tiklashni talab qiladi. Element qo‘shishga o'xshab, biz B daraxtning xususiyatlarini saqlaganligimizni tekshirishimiz kerak, faqat bu holda biz kalitlarning t-1 bo'lishini kuzatishimiz kerak (ya'ni, agar bu tugundan kalit o'chirilsa, tugun mavjud bo'la olmaydi). O'chirish algoritmini ko'rib chiqamiz:

1) Agar bargdan o'chirish sodir bo'lsa, unda qancha kalit borligini tekshirish kerak. Agar t-1 dan ko'p bo'lsa, biz shunchaki o'chirib tashlaymiz va boshqa hech narsa qilishimiz shart emas. Aks holda, agarda t-1 dan ortiq kalitlarni o'z ichiga oluvchi qo'shni barg bo'lsa (uning yonida joylashgan va avlod-ajdodi bir xil bo'lsa), biz qo'shni tugunning qolgan kalitlari orasidagi ajratuvchi bo'lgan bu qo'shnidan kalitni tanlaymiz. Bu kalit k_1 bo'lsin. Avval tugunni va uning qo'shnisini ajratuvchi asosiy tugundan k_2 kalitini tanlaymiz. Kerakli kalitni manba tugundan olib tashlaylik (o'chirilishi kerak edi), bu tugunga k_2 ni tushiring va ajdod tugunidagi k_2 o'rniga k_1 qo'ying. Tushunarli bo'lishi uchun quyida "9" tugmasi o'chirilgan rasm (48 -rasm) ko'rsatilgan. Agar bizning tugunning barcha qo'shnilarida t-1 kalitlari bo'lsa. Keyin biz uni qo'shnisi bilan birlashtiramiz, kerakli kalitni o'chirib tashlaymiz va bu ikkita "sobiq" qo'shnilar uchun ajratuvchi bo'lgan avlod-ajdod tugunining kaliti, yangi tashkil etilgan tugunimizga o'tamiz (bu, albatta, undagi mediana bo'ladi).

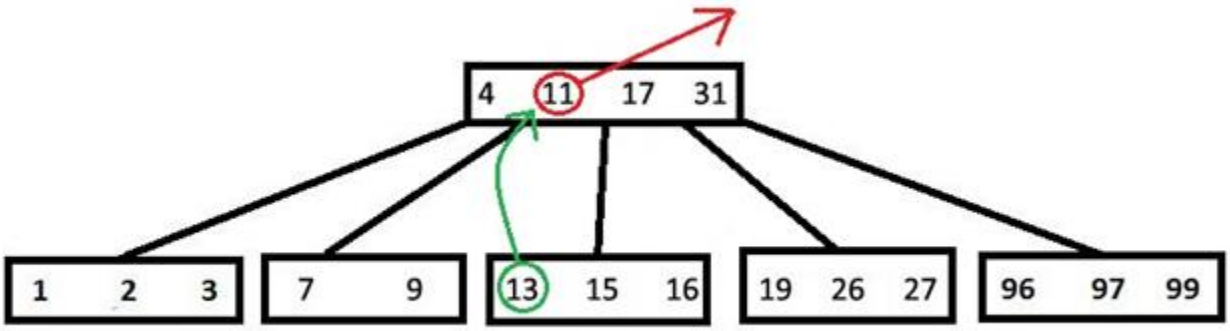


48-rasm. B daraxtda element o‘chirish algoritmining bajarilishi

2) Endi x ichki tugunidan k kalitini olib tashlashni ko'rib chiqaylik. Agar k kalitidan oldingi avlod tugunida t -1 dan ortiq kalitlar bo'lsa, biz k_1 ni topamiz - bu tugunning pastki daraxtida k. Uni o'chirib tashlaymiz (Algoritmni rekursiv tarzda ishga tushiramiz). Manba tugundagi k ni k_1 bilan almashtiramiz. Agar k kalitidan keyingi avlod tugunida t-1 dan ortiq kalit bo'lsa, biz ham xuddi shunday ishni bajaramiz. Agar ikkalasida ham (keyingi va oldingi avlod tugunlarida) t-1 kalitlari bo'lsa, biz bu avlodlarni birlashtiramiz, k-ni ularga o'tkazamiz, so'ngra k-ni yangi tugundan olib tashlaymiz (biz algoritmimizni rekursiv tarzda ishga

tushiramiz). Agar ildizning oxirgi 2 avlodi birlashsa, ular ildizga aylanadi va oldingi ildiz olib tashlanadi. Rasm quyida ko‘rsatilgan (49-rasm), bu yerda "11" ildizdan chiqariladi (keyingi tugunda t-1 avlodlari ko'p bo'lgan holat).

O'chirish jarayoni O (t logt n) qo'shilishi bilan bir xil vaqtni oladi va disk operatsiyalari faqat O (h) talab qilinadi, bu yerda h - daraxt balandligi.



49-rasm. B daraxtda element o‘chirish algoritmining bajarilishi

10.3. B-daraxtni realizatsiya qilish

```
/* B-daraxtning minimum darajasi*/
#define T 2        /* 2-3-4 B-tree */
struct btree {
int leaf;
int nkeys;
int *key;
int *value;
struct btree **childj
};
```

B-daraxtni hosil qilish

```
struct btree *btree_create()
{
struct btree *node;
node = malloc(sizeof(*node));
node->leaf = 1;
node->nkeys = 0;
node->key = malloc(sizeof(*node->key) *
```

```
        2 * T - 1);
node->value = malloc(sizeof(*node->value)
        2 * T - 1);
node->child = malloc(sizeof(*node->child)
2 * T);

        return node;
        }
```

B daraxtda element izlash

```
        void btree_lookup(struct btree *tree, int key,
        struct btree **node, int *index)
        {
        int i;
        for (i = 0; i < tree->nkeys && key > tree->key[i]; ) {
i++;
        }
        if (i < tree->nkeys && key == tree->key[i]) {
        *node = tree;
        *index = i;
return;
        }
        if (!tree->leaf) {
        /* Disk read tree->child[i] */
btree_lookup(tree, key, node, index);
        } else {
        *node = NULL;
        }
```

B daraxtda element qo‘shish

```
        struct btree *btree_insert(struct btree *tree,
        int key, int value)
        {
        struct btree *newroot;
        if (tree == NULL) {
        tree = btree_create();
tree->nkeys = 1;
tree->key[0] = key;
```

```
tree->value[0] = value;
return tree;
        }
        if (tree->nkeys == 2 * T - 1) {
        newroot = btree_create(); /* Create empty root */
        newroot->leaf = 0;
        newroot->child[0] = tree;
        btree_split_node(tree, newroot, 0);
        return btree_insert_nonfull(newroot, key, value);
        }
        return btree_insert_nonfull(tree, key, value);
        }

        B-daraxtda tugunni ajratish

        void btree_split_node(struct btree *node,
        struct btree *parent, int index)
        {
        struct btree *z;
int i;
        z = btree_create();
z->leaf = node->leaf;
z->nkeys = T - 1;
for (i = 0; i < T - 1; i++) {
        z->key[i] = node->key[T + i];
z->value[i] = node->value[T + i];
        }
        if (!node->leaf) {
        for (i=0; i<T; i++)
        z->child[i] = node->child[i + T]j
        }
        node->nkeys = T - 1;
        /* Ajdod tugunga mediana kalitni kiritish */
for (i = parent->nkeys; i >= 0 && i <= index + lj i--)
parent->child[i + 1] = parent->child[i];
parent->child[index + 1] = z;
        for (i = parent->nkeys - 1; i >= 0 && i <= index; i--)
parent->key[i + 1] = parent->key[i];
parent->value[i + 1] = parent->value[i];
```

```
        }
        parent->key[index] = node->key[T - 1];
parent->value[index] = node->value[T - 1];
parent->nkeys++;

        struct btree *btree_insert_nonfull(
        struct btree *node, int key, int value)
        {
        int i;
        i = node->nkeys;
if (node->leaf) {
        for (i = node->nkeys - 1; i > 0 &&
key < node->key[i]; i--)
        {
        node->key[i + 1] = node->key[i]
        }
        node->key[i + 1] = key;
node->nkeys++;
        } else {
        for (i = node->nkeys - 1; i > 0 &&
key < node->key[i]; )
        {
        i--;
        }
        i++;
        if (node->child[i]->nkeys == 2 * T - 1) {
        btree_split_node(node->child[i], node, i);
if (key > node->key[i])
i++J
        }
        node = btree_insert_nonfull(node->child[i],
key, value);
        }
        return node;
        }
        int main()
        {
        struct btree *tree;
        tree = btree_insert(NULL,       3, 0);
```

```
    tree = btree_insert(tree,  12, 0);
    tree = btree_insert(tree,  9, 0);
    tree = btree_insert(tree,  18, 0);
    return 0;
}
```

Mavzu yuzasidan savollar:

1. B daraxt nima
2. B daraxtda izlash qanday amalga oshiriladi?
3. B daraxtda element o‘chirish qanday amalga oshiriladi?
4. B daraxtda element qo‘shish qanday amalga oshiriladi?
5. B-daraxt ma’lumotlar strukturasi qo‘llaniladigan sohalarga qaysilar kiradi?

Mustaqil ishlash uchun masalalar:

1. B daraxtida element olib tashlash funksiyasini yozing va uni daraxtda qo‘llang
2. B-daraxtda element qo‘shish funksiyasini optimallashtiring

11-§. Ustivor navbatlar

Ko'pgina ilovalar kalitlarga ega bo'lgan elementlarni qayta ishlashni talab qiladi, lekin ular to'liq tartibda va birdaniga hammasi emas. Ko'pincha, biz bir qator narsalarni to'playmiz, so'ngra eng katta kalit bilan ishlov beramiz, keyin ko'proq narsalarni to'playmiz, so'ngra hozirgi eng katta kalit bilan ishlov beramiz va hokazo. Bunday muhitda tegishli ma'lumotlar turi ikkita amalni qo'llab-quvvatlaydi: maksimal miqdorni o‘chirish va joylashtirish. Bunday ma'lumotlar turi **ustivor navbat** deb nomlanadi.

Ustivor navbatlar odatdagi navbat yoki stek ma'lumotlar tuzilmasiga o'xshash abstrakt ma'lumotlar turi bo'lib, unda har bir element qo'shimcha ravishda bog‘liq bo'lgan "ustivorlikka" ega. Ustivor navbatda yuqori ustivor element past ustivor elementdan oldin xizmat qiladi.

Ustivor navbatlar ko'pincha uyum (kucha) bilan amalga oshirilsa-da, ular konseptual jihatdan uyumlardan farq qiladi. Ustivor navbat - bu "ro'yxat" yoki "karta" ga o'xshash narsa; Ro'yxat bog‘langan ro'yxat yoki massiv yordamida amalga oshirilishi mumkin bo'lganidek, ustivor navbat uyum yoki tartiblanmagan massiv kabi boshqa usullar yordamida amalga oshirilishi mumkin.

Ustivor navbat - bu yozuvlar bir-biri bilan chiziqli taqqoslanadigan kalitlarga (masalan, raqamlar) ega bo'lgan va ikkita amalni realizatsiya qiladigan axborot tizimidir. Bu ikki amal tizimga tasodifiy yozuvni kiritish va yozuv tizimidan eng kichigi bilan tanlov kalit.

Dasturiy ta'minot tizimlarida ustivor navbatlar juda keng tarqalgan va dasturlarning ishlashi to'g‘ridan-to'g‘ri ularni amalga oshirish samaradorligiga bog‘liq.

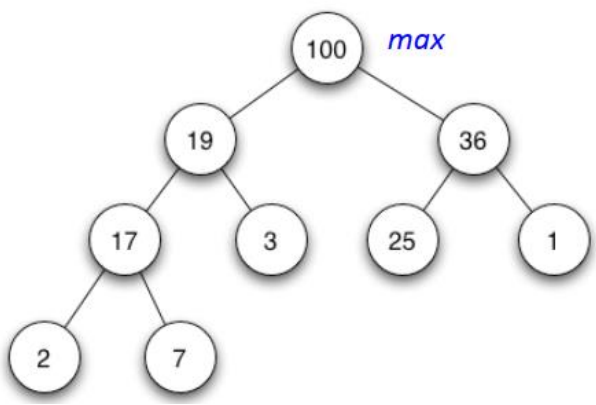
Ustivorda navbatda qo‘llab-quvvatlanadigan amallar quyidagilar hisoblanadi:

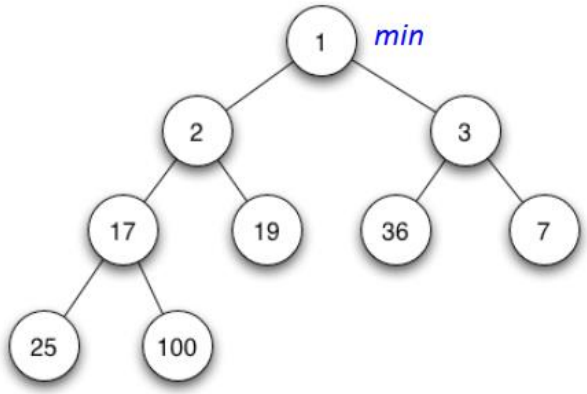
1) Insert - navbatga element qo'shish
2) Max - ustivorligi yuqori bo'lgan elementni qaytaradi
3) ExtractMax - navbatdagi eng ustivor elementni olib tashlaydi
4) IncreaseKey - berilgan elementning ustivor qiymatini o'zgartiradi
5) Merge - ikkita navbatni bittaga birlashtiradi

11.1. Binar uyum (kucha) - piramida (binary heap)

Binar uyum (binary heap) bu quyidagi shartlarni qanoatlantiradigan binar daraxtdir:

- Har qanday uchning ustivorligi, uning avlodlarining ustivorligidan kichik emas.
- Daraxt to'liq ikkilik daraxt bo‘lishi uchun (complete binary tree) - barcha darajalar chapdan o'ngga to'ldiriladi (oxirgisi bundan mustasno bo‘lishi mumkin).





O’smaydigan piramida

max-heap

Har qanday uchning ustuvorligi avlodlarning ustuvorligidan kichik emas

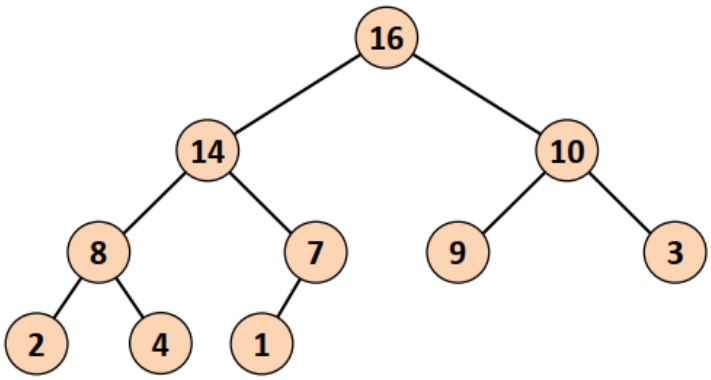
Kamaymaydigan piramida

min-heap

Har qanday uchning ustuvorligi avlodlarning ustuvorligidan katta emas

50-rasm. Binar uyum (kucha)

Massivlar orqali binar uyum (kucha) ni realizatsiya qilish

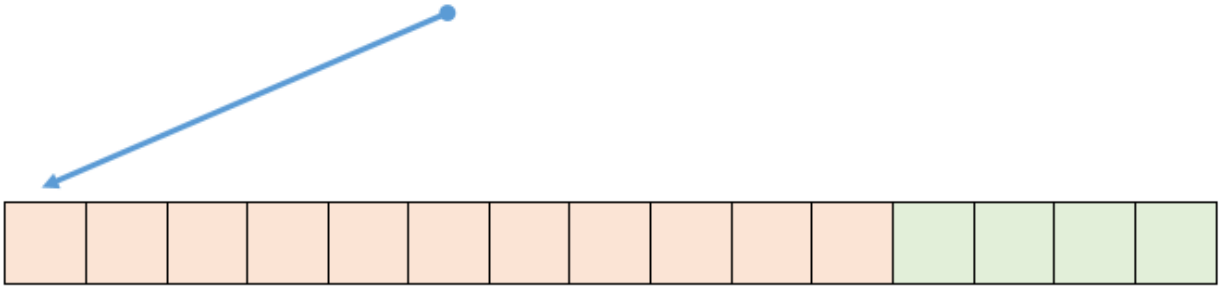


max-heap (10 ta element)

16	14	10	8	7	9	3	2	4	1				

H[1..10] ustuvorliklar (kalitlar) massivi

Daraxtning ildizi H [1] yacheykada saqlanadi - bu maksimal element;



i tugunning ajdod indeksi: $Parent(i)$= [i / 2];
Chap avlod tugun indeksi: $Left(i) = 2i$;
O‘ng avlod tugun indeksi: Right(i) = $2i + 1$;
$H[Parent(i)] \geq H[i]$.

```
struct heapnode {
    int key;          /* kalit */
    char *value;      /* qiymat*/
};


struct heap {
    int maxsize;     /* massiv o‘lchami */
    int nnodes;      /* Kalitlar soni */
    struct heapnode *nodes; /* Nodes: [0..maxsize] */
}
```

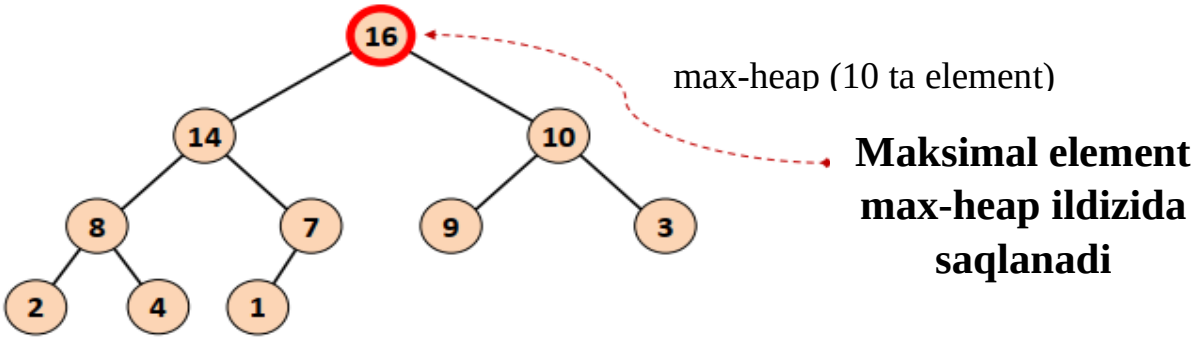
Bo‘sh uyum (kucha) hosil qilish

```
struct heap *heap_create(int maxsize)
{
        struct heap *h;
        h = malloc(sizeof(*h));
        if (h != NULL)
        {
        h->maxsize = maxsize;
        h->nnodes = 0;
    /* Heap nodes [0, 1, maxsize] */
        h->nodes = malloc(sizeof(*h->nodes) * (maxsize + 1));
        if (h->nodes == NULL)
```

```
        {
        free(h);
        return NULL;
        }
return h;
}
```

Uyumni o‘chirish

```
void heap_free(struct heap *h)
{
free(h->nodes);
free(h);
}
void heap_swap(struct heapnode *a, struct heapnode *b)
{
struct heapnode temp;
temp = *a;
*a = *b;
```

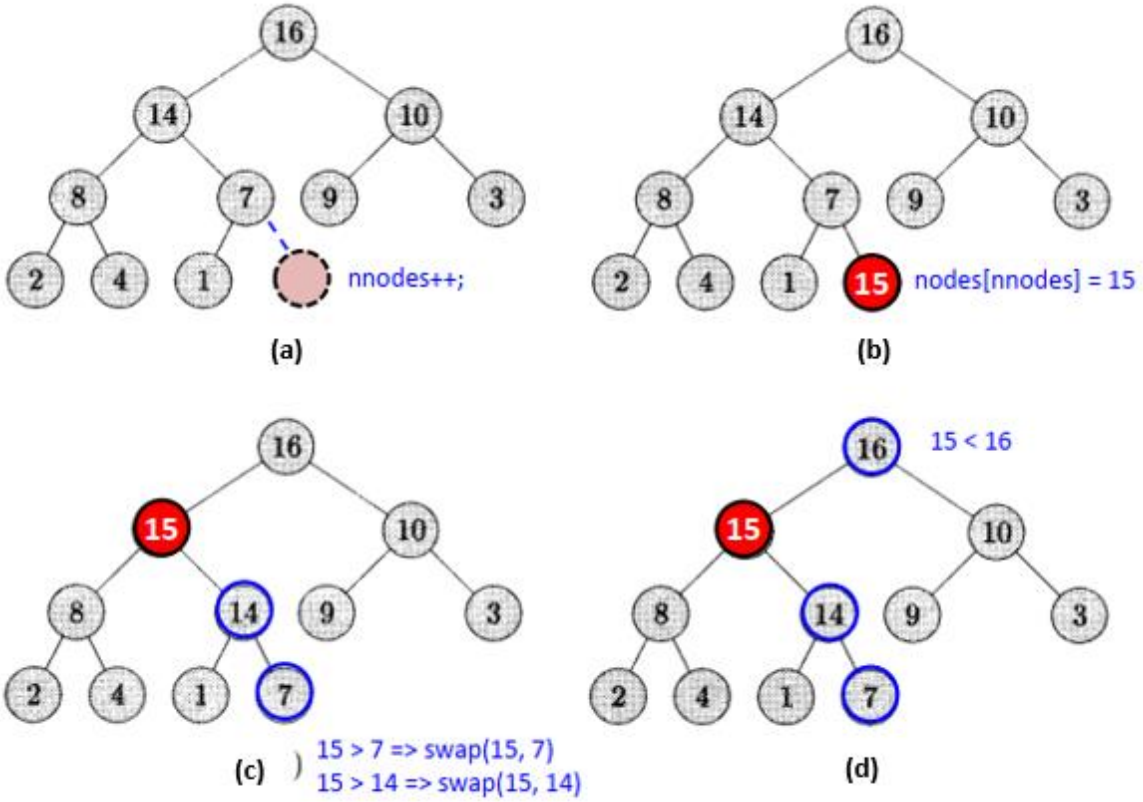


16	14	10	8	7	9	3	2	4	1				

51-rasm. Maksimal elementni izlash

```
struct heapnode *heap_max(struct heap *h
{
if (h->nnodes == 0)
   return NULL;
return &h->nodes[l];
}
```

Binar uyum (kucha) ga element qo‘shish. Ustuvorligi 15 ga teng bo‘lgan elementni joylashtirish



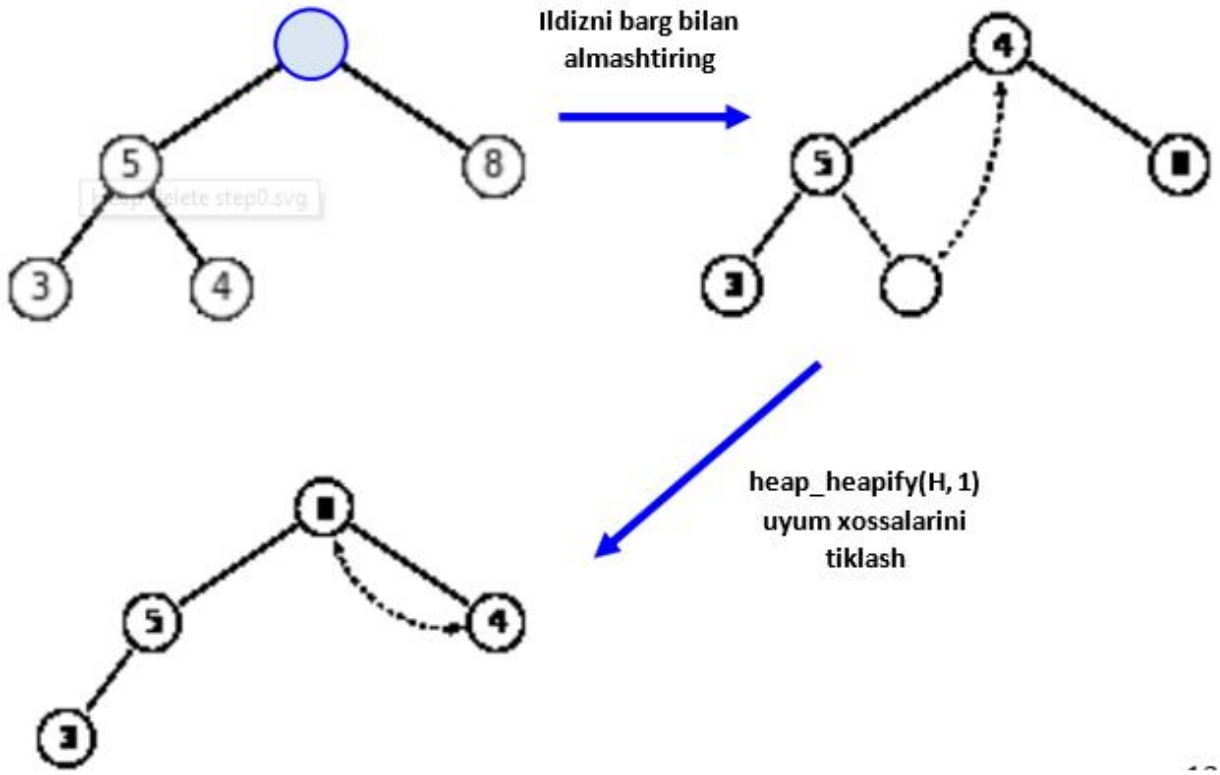
52-rasm. Binar uyum (kucha) ga element qo‘shish

Binar kuchaga element joylashtirish

```
int heap_insert(struct heap *h, int key, char *value)
{
        if (h->nnodes >= h->maxsize) {
return -1;
}
h->nnodes++;
h->nodes[h->nnodes].key = key;
h->nodes[h->nnodes].value = value;
for (int i = h->nnodes; i > 1 &&
        h->nodes[i].key > h->nodes[i / 2].key; i = i/2)
        {
                heap_swap(&h->nodes[i], &h->nodes[i / 2]);
        }
return 0;
```

}

Maksimal elementni o‘chirish



53-rasm. Maksimal elementni o‘chirish

```
struct heapnode heap_extract_max(struct heap
{
if (h->nnodes == 0)
return (struct heapnode){0, NULL};
struct heapnode maxnode = h->nodes[l];
h->nodes[l] = h->nodes[h->nnodes];
h->nnodes--;
heap_heapify(h, 1);
return maxnode;
}
```

Uyum xususiyatlarini tiklash

```
void heap_heapify(struct heap *h, int index)
{
    for (;;) {
```

```
int left = 2 * index;
int right = 2 * index + 1;
int largest = index;
if (left <= h->nnodes &&
h->nodes[left].key > h->nodes[index].key)
{ largest = left; }
if (right <= h->nnodes && h->nodes[right].key > h->nodes[largest].key)
    { largest = right; }
if (largest == index)
    break;
heap_swap(&h->nodes[index], &h->nodes[largest]);
index = largest;
    }
}
```

Kalit qiymatini oshirish

```
int heap_increase_key(struct heap *h, int index, int key)
{
if (h->nodes[index].key > key)
    return -l;
h->nodes[index].key = key;
for ( ; index > 1 && h->nodes[index].key > h->nodes[index / 2].key; index = index
/ 2)
{
        heap_swap(&h->nodes[index], &h->nodes[index / 2]);
}
return index;
}
```

Binar kucha bilan bilan ishlash

```
int main()
{
        struct heap *h;
        struct heapnode node;
        h - heap_create(100);
        heap_insert(h,      16, "16");
        heap_insert(h,      14, "14");
        heap_insert(h,      10, “10");
```

```
    heap_insert(h,      8, "8");
    heap_insert(h,      7, "7");
    heap_insert(h,      9, “9");
    heap_insert(h,      3, "3");
    heap_insert(h,      2, "2");
    heap_insert(h,      A, “4”);
    heap_insert(h,      1, "1");
    node = heap_extract_max(h);
    printf(“Item: %d\n", node.key);
    int i = heap_increase_key(h, 9, 100);
    heap_free(h);
    return 0;
}
```

11.2. Uyum (kucha)larni saralash (Heap-Sort)

Heapsort (Heapsort, "Heap sorting") - n elementlarni saralashda O(nlogn) amallarda eng yomon, o'rtacha va eng yaxshi (ya'ni kafolatlangan) holda ishlaydigan saralash algoritmi. Ishlatiladigan qo'shimcha xotira miqdori massiv kattaligiga bog‘liq emas (ya'ni O (1)).

Ushbu saralashni pufaksimon saralashning rivojlantirilgan ko‘rinishi deb qarash mumkin.

Eng yomon vaqt - $O(n\log(n))$

Eng yaxshi vaqt - $O(n\log(n))$

O‘rtacha vaqt - $O(n\log(n))$

Heap-Sort algoritmini realizatsiya qilish (C++)

```
#include <iostream>
#include <time.h>

using namespace std;

int main()
{
   srand(time(NULL));
   int const n = 100;
   int a[n];
```

```
for ( int i = 0; i < n; ++i )
   {
      a[i] = rand()%1000;
      cout << a[i] << " ";
   }
//massivni to'ldirish
//-----------Saralash------------//
//O'sish bo'yicha saralash.
int sh = 0; //смещение
bool b = false;
for(;;) //Sikl cheksiz davom etadi
{
b = false;
for ( int i = 0; i < n; i++ )
{
   if( i * 2 + 2 + sh < n )
   {
   if( ( a[i + sh] > /*<*/ a[i * 2 + 1 + sh] ) || ( a[i + sh] > /*<*/ a[i * 2 + 2 +
sh] ) )
   {
      if ( a[i * 2 + 1 + sh] < a[i * 2 + 2 + sh] )
      {
      swap( a[i + sh], a[i * 2 + 1 + sh] );
      b = true;
      }
      else if ( a[i * 2 + 2 + sh] < a[ i * 2 + 1 + sh])
          {
             swap( a[ i + sh], a[i * 2 + 2 + sh]);
             b = true;
          }
   }
   // oxirgi ikki element uchun qo'shimcha tekshirish
         // ushbu tekshiruv yordamida siz piramidani saralashingiz mumkin
         // faqat uchta elementdan iborat
      if( a[i*2 + 2 + sh] < /*>*/ a[i*2 + 1 + sh] )
      {
      swap( a[i*2+1+sh], a[i * 2 +2+ sh] );
               b = true;
      }
```

```
        }
        else if( i * 2 + 1 + sh < n )
            {
               if( a[i + sh] > /*<*/ a[ i * 2 + 1 + sh] )
               {
                  swap( a[i + sh], a[i * 2 + 1 + sh] );
                  b = true;
               }
            }
      }
      if (!b) sh++;
      if ( sh + 2 == n ) break;
      }  //Saralash tugatildi

      //Natijani chiqarish
      cout << endl << endl;
      for ( int i = 0; i < n; ++i ) cout << a[i] << " ";


     // getch();
      return 0;
   }
```

Mavzu yuzasidan savollar:

1. Ustivor navbat nima?
2. Heap-Sort algoritmi haqida gapiring
3. Uyum tushunchasi.
4. Binar kucha bilan ishlash?
5. Ustivor navbat ma’lumotlar strukturasi qo‘llaniladigan sohalarga qaysilar kiradi?

Mustaqil ishlash uchun masalalar:

1. Masalalarda Heap-Sort algoritmini qo‘llang.
2. Binar uyumga element qo‘shish dasturini yozing

12-§. Hisoblash geometriyasi algoritmlari

Hisoblash geometriyasi - geometrik masalalarni yechish algoritmlari bilan shug‘ullanadigan informatika bo'limi.

Bu uchburchak, qavariq sirtlarni qurish, bitta obyektning boshqasiga tegishliligini aniqlash, ularning kesishishini topish va boshqalar kabi vazifalar bilan shug‘ullanadi, ular geometrik obyektlar bilan ishlaydi: nuqta, segment, ko'pburchak, aylana va hokazolar.

Hisoblash geometriyasi kompyuter grafikalarida, muhandislik dizaynida va boshqa ko‘plab geometriya sohalarida qo'llaniladi.

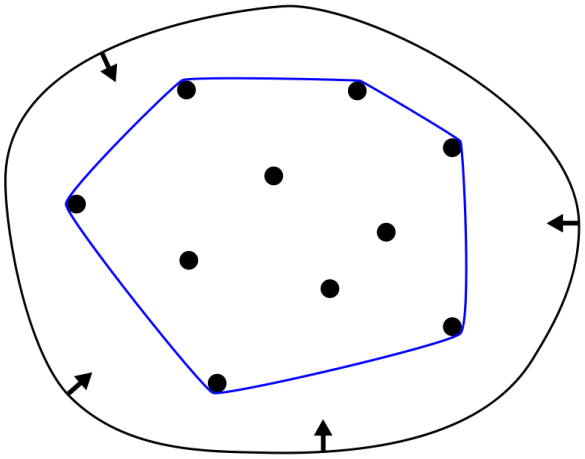
12.1. Qavariq qobiq muammolari

X to'plamining qavariq qobig‘i - X ni o'z ichiga olgan eng kichik qavariq to'plami. "Eng kichik to'plam" bu yerda to'plamlarni joylashtirishga nisbatan eng kichik elementni, ya'ni berilgan raqamni o'z ichiga olgan shunday qavariq to'plamni anglatadiki, u berilgan figurani o'z ichiga olgan boshqa har qanday qavariq to'plamda mavjud.

X to'plamining qavariq tanasi odatda ConvX bilan belgilanadi.

Misol. Ko'plab mixlar mixlangan taxtani tasavvur qiling. Arqonni oling, ustiga sirpanchiq ilmoq (lasso) bog‘lab, taxtaga tashlang va keyin mahkamlang. (54-rasm)

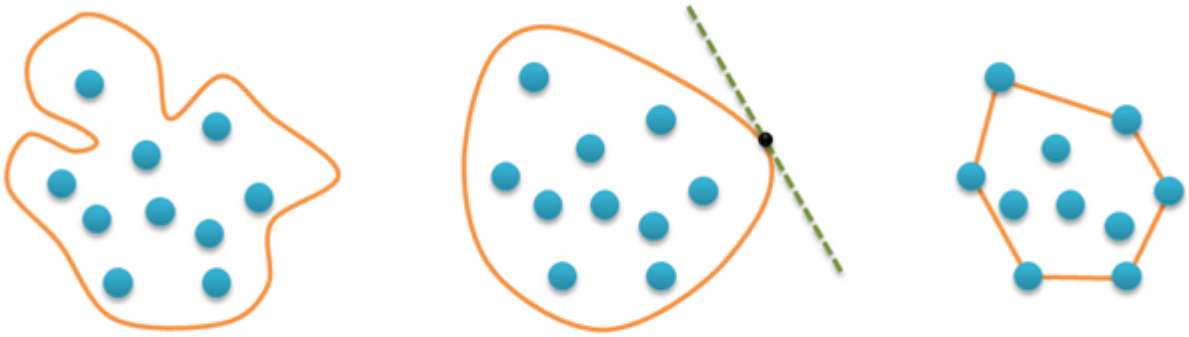
Arqon barcha mixlarni o'rab oladi, lekin u faqat eng tashqi qismlariga tegadi. U tegib turgan mixlar butun mixlar guruhi uchun qavariq qobiqni hosil qiladi.



54-rasm. Ko‘plab mixlar mixlangan taxta tasviri

Minimal qavariq qobiq tushunchasi. Tekislikda cheklangan A nuqtalar to'plami berilgan bo'lsin. Bu to'plamning konvertlari o'zaro kesishmalarsiz har qanday yopiq H chiziq bo'lib, A ning barcha nuqtalari shu egri chiziq ichida yotadi. Agar H egri chiziq qavariq bo'lsa (masalan, bu egri chiziqning har qanday urinish nuqtasi uni boshqa biron bir nuqtada kesib o'tmasa), u holda tegishli qobiq ham qavariq deb ataladi. Va nihoyat, minimal qavariq qobiq minimal uzunlikdagi (minimal perimetr) qavariq qobiq deb ataladi. Barcha kiritilgan tushunchalar quyidagi 55-rasmda keltirilgan.

A nuqtalar to'plamli minimal qavariq qobiqning asosiy xususiyati shundaki, bu tanasi qavariq ko'pburchak bo'lib, uning uchlari A dagi bir nechta nuqtadir, shuning uchun minimal qavariq qobiqni topish muammosi oxir-oqibat A dan kerakli nuqtalarni tanlash va tartiblashgacha kamayadi. Algoritm chiqishi ko'pburchak bo'lishi kerakligi sababli tartiblash ya’ni saralash zarur, ya'ni uchlar ketma-ketligi bo‘yicha. Uchlar tartibiga qo'shimcha ravishda shart qo'yamiz - ko'pburchakning o'tish yo'nalishi musbat bo'lishi kerak (soat strelkasi bo‘yicha).



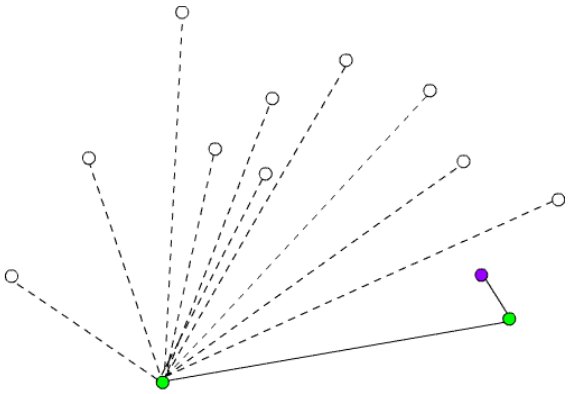
a) Qobiq **b) Qavariq qobiq** **c) Minimal qavariq qobiq**

55-rasm. Qobiq, qavariq qobiq va minimal qavariq qobiq tushunchalari

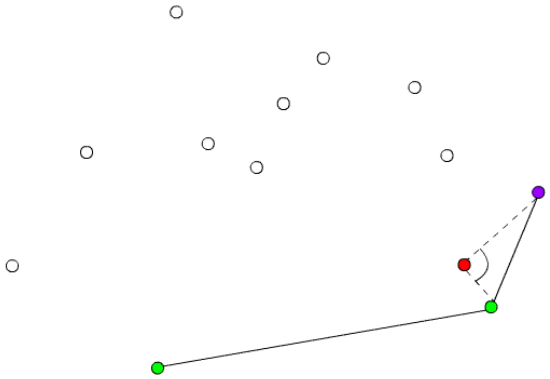
Minimal qavariq qobiqni qurish masalasi hisoblash geometriyasidagi eng oddiy muammolardan biri hisoblanadi; buning

uchun juda ko'p turli algoritmlar mavjud. Quyida biz ikkita algoritmni ko'rib chiqamiz – Grexem (Graham scan) va Jarvis (Jarvis march).

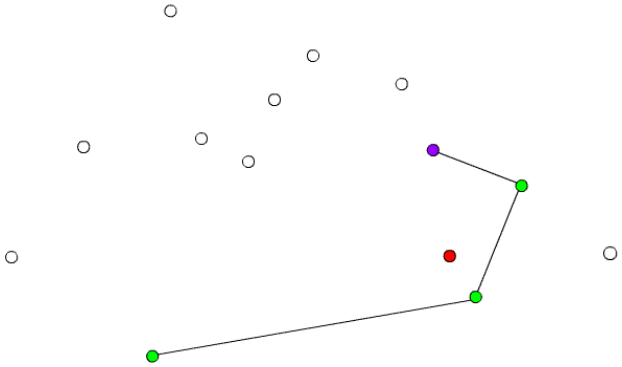
Grexem algoritmi quyidagi 56-rasmda ketma-ket keltirilgan.



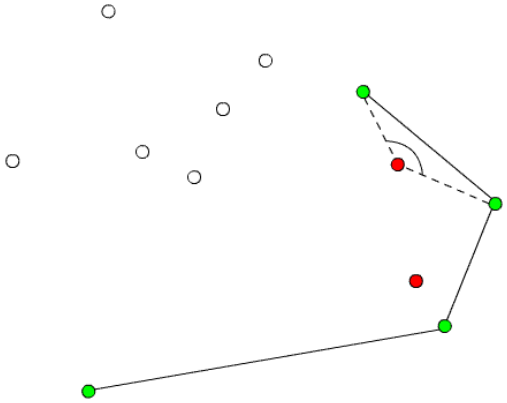
1-qadam



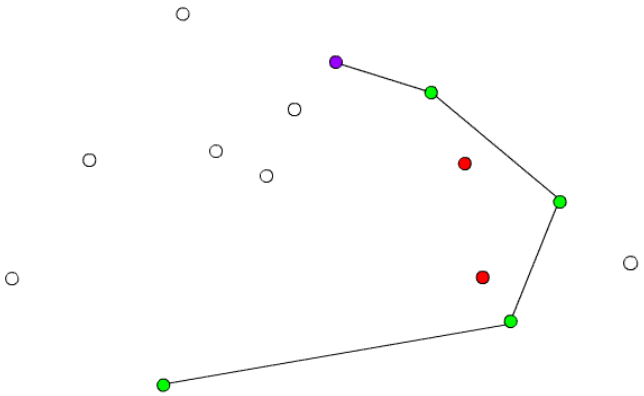
2-qadam



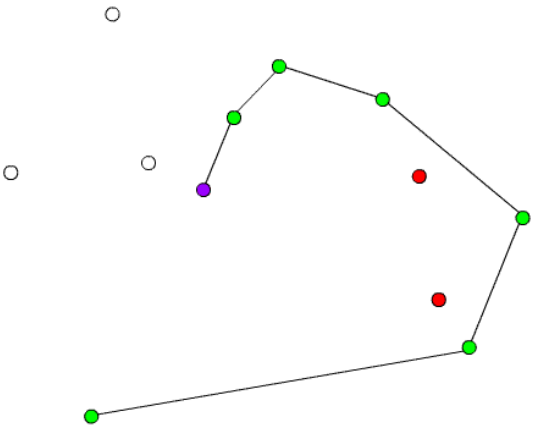
3-qadam



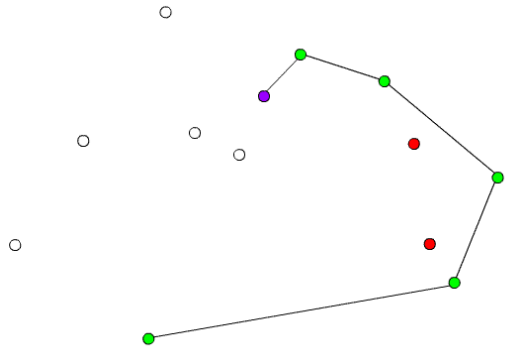
4-qadam



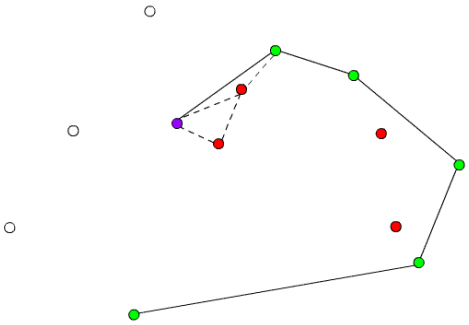
5-qadam



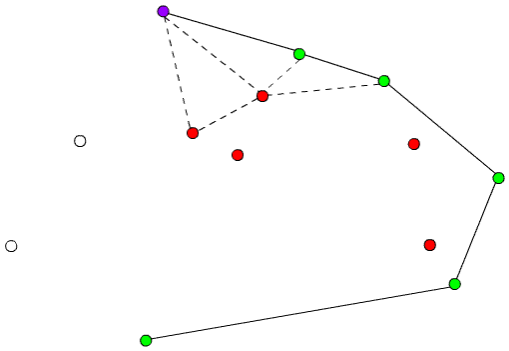
6-qadam



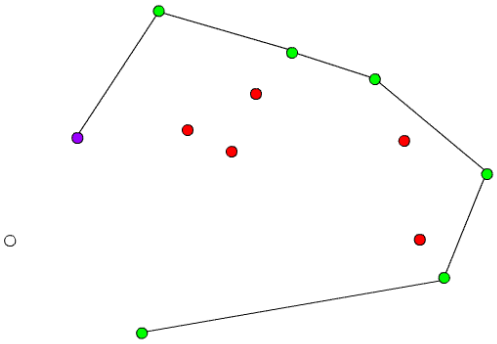
7-qadam



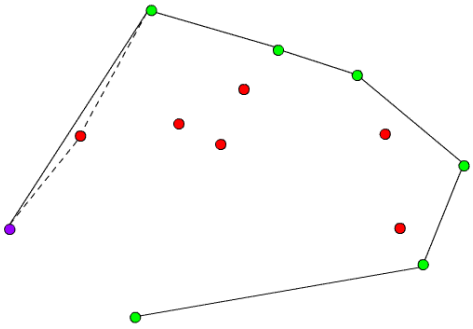
8-qadam



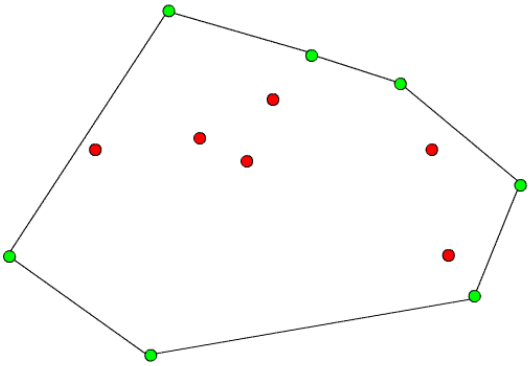
9-qadam



10-qadam



11-qadam



12-qadam

56-rasm. Grexem algoritmining bajarilish ketma-ketligi

Grexem algoritmi realizatsiyasi (C++ tilida)

```
#include <iostream>
#include <stack>
#include <stdlib.h>
using namespace std;

struct Point
{
	int x, y;
};

Point p0;
Point nextToTop(stack<Point> &S)
{
	Point p = S.top();
	S.pop();
	Point res = S.top();
	S.push(p);
	return res;
}
void swap(Point &p1, Point &p2)
{
	Point temp = p1;
	p1 = p2;
	p2 = temp;
}
int distSq(Point p1, Point p2)
{
	return (p1.x - p2.x)*(p1.x - p2.x) +
		(p1.y - p2.y)*(p1.y - p2.y);
}
int orientation(Point p, Point q, Point r)
{
	int val = (q.y - p.y) * (r.x - q.x) -
			(q.x - p.x) * (r.y - q.y);

	if (val == 0) return 0;
```

```
        return (val > 0)? 1: 2;
}
int compare(const void *vp1, const void *vp2)
{
Point *p1 = (Point *)vp1;
Point *p2 = (Point *)vp2;


int o = orientation(p0, *p1, *p2);
if (o == 0)
        return (distSq(p0, *p2) >= distSq(p0, *p1))? -1 : 1;

return (o == 2)? -1: 1;
}
void convexHull(Point points[], int n)
{

int ymin = points[0].y, min = 0;
for (int i = 1; i < n; i++)
{
        int y = points[i].y;

        if ((y < ymin) || (ymin == y &&
                points[i].x < points[min].x))
                ymin = points[i].y, min = i;
}
swap(points[0], points[min]);
p0 = points[0];
qsort(&points[1], n-1, sizeof(Point), compare);
int m = 1; // Initialize size of modified array
for (int i=1; i<n; i++)
{

        while (i < n-1 && orientation(p0, points[i],
points[i+1]) == 0)
                i++;
        points[m] = points[i];
        m++;
if (m < 3) return;
```

```
    stack<Point> S;
    S.push(points[0]);
    S.push(points[1]);
    S.push(points[2]);


    for (int i = 3; i < m; i++)
    {

        while (S.size()>1 && orientation(nextToTop(S), S.top(), points[i])
!= 2)
            S.pop();
        S.push(points[i]);
    }
    while (!S.empty())
    {
        Point p = S.top();
        cout << "(" << p.x << ", " << p.y <<")" << endl;
        S.pop();
    }
    }
    }
    int main()
    {
        Point points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},
                          {0, 0}, {1, 2}, {3, 1}, {3, 3}};
        int n = sizeof(points)/sizeof(points[0]);
        convexHull(points, n);
        return 0;
    }
```

12.2. Tekislikda chiziqlar kesishgan sohalarni qidirish algoritmi (Sweep Line)

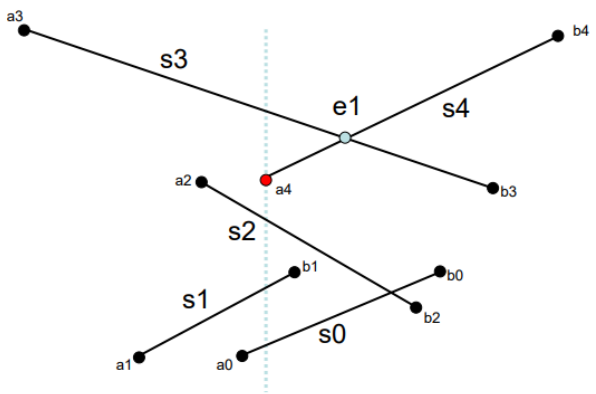
Tekislikda chiziqlar kesishgan sohalarni qidirish algoritmi(Sweep Line) - bu Yevklid fazosidagi turli xil muammolarni hal qilish uchun kesishgan sohalardan foydalanadigan algoritmik paradigma. Bu hisoblash geometriyasidagi asosiy texnikalardan biridir.

Ushbu turdagi algoritmlarning g‘oyasi ba'zi bir nuqtalarda to'xtab, tekislik bo'ylab harakatlanadigan xayoliy to'g‘ri chiziqni (ko'pincha vertikal) tasavvur qilishdir. Geometrik amallar, kesishgan chiziqqa qo'shni bo'lgan geometrik obyektlar bilan cheklanadi va chiziq barcha obyektlardan o'tib ketganda to'liq yechim mavjud bo‘ladi.

Sweep Line – bu tekislik bo'ylab to'g‘ri yo'nalishda siljigan xayoliy vertikal chiziq. Shuning uchun bu konsepsiyaga asoslangan algoritmlarni ba'zan tekisliklarni tozalash algoritmlari deb ham atashadi. Chiziqni diskretlashtirish uchun biz ba'zi hodisalarga asoslanib tozalaymiz.

Yana shuni ta'kidlash kerakki, bu texnikaning samaradorligi biz foydalanadigan ma'lumotlar tuzilmalariga bog‘liq. Umuman olganda, biz C++ da setdan foydalanishimiz mumkin, lekin ba'zida biz qo'shimcha ma'lumotlarni saqlashni talab qilamiz, shuning uchun biz muvozanatlashgan ikkilik daraxtga o'tamiz.

Misol. Sweep Line algoritmining ishlash tartibi



Sweep Line holati (SLH): s0, s1, s2, s3

Navbat (N): a4, b1, b2, b0, b3, b4

Harakatlar:

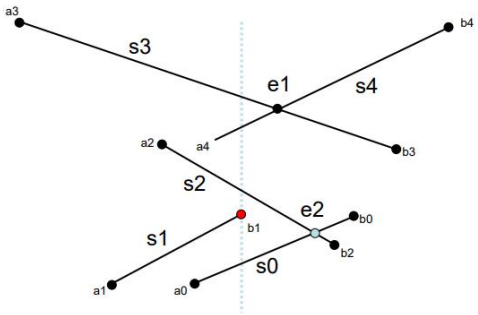
SLH -ga s4 -ni kiriting

Test s4-s3 va s4-s2

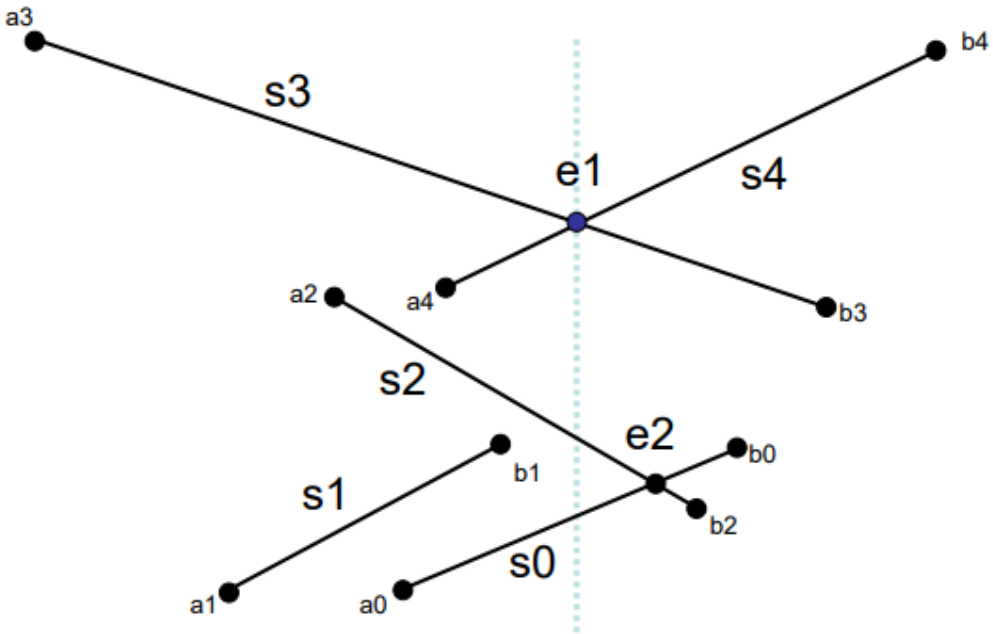
N ga e1 ga qo'shing

Sweep Line holati: s0, s1, s2, s4, s3

Navbat: b1,e1, b2, b0, b3, b4



Harakatlar:
SLH dan s1 ni o‘chiring
Test s0-s2
N ga e2 ga qo'shing
Sweep Line holati: s0, s2, s4, s3
Navbat: e1, e2, b2, b0, b3, b4



Harakatlar:
SLH da s3 va s4 ni almashtiring
Test s3-s2

Sweep Line holati: s0, s2, s3, s4
Navbat: e2, b2, b0, b3, b4

Mavzu yuzasidan savollar:

1. Hisoblash geometriya qanday masalalarni hal qiladi?
2. Yuqoridagi keltirilgan hisoblash geometriya masalalaridan tashqari qanday masalalarni keltira olasiz?
3. Minimal qavariq qobiq tushunchasiga ta’rif bering.

13-§. Xesh jadvallar

13.1. Xesh jadvallar va ularni tashkil etish

Xesh jadvali - bu assotsiativ massiv interfeysini amalga oshiruvchi ma'lumotlar tuzilmasi, ya'ni juftlarni saqlashga (kalit, qiymat) va uchta amalni bajarishga imkon beradi: yangi juftlikni qo'shish, qidirish amali va juftlikni kalit bilan o'chirish.

Xesh jadvallarining ikkita asosiy varianti mavjud: zanjirli va ochiq adreslash. Xesh jadvali ba'zi bir H massivini o'z ichiga oladi, ularning elementlari juftliklar (ochiq adreslash bilan xesh jadvali) yoki juftliklar ro'yxati (zanjir bilan xesh jadvali) bo‘ladi.

Xeshlash – bu ixtiyoriy uzunlikdagi kirish ma'lumotlari majmuasini ma'lum bir algoritm tomonidan bajarilgan, belgilangan o'lchamdagi chiqish massiviga aylantirish jarayoni. Bunday algoritmni amalga oshiruvchi funksiya **xesh funksiya**, transformatsiya natijasi xesh yoki xesh yig‘indisi deyiladi. Xesh funksiyasi quyidagi xususiyatlarga ega:

- bir xil ma'lumotlar bir xil xeshni beradi;
- "deyarli har doim" turli xil ma'lumotlar boshqacha xesh beradi.

Ikkinchi xususiyatdagi "deyarli har doim" izohi xeshlarning aniq o'lchamiga ega bo'lishidan kelib chiqadi, shu bilan birga kirish ma'lumotlari bu bilan cheklanmaydi. Natijada, biz xesh funktsiyasi kirish ma'lumotlari to'plamidan xeshlar to'plamiga xaritalashni amalga oshiramiz, bu esa ularning kardinalligi ancha past bo'ladi. Dirixle prinsipiga ko'ra, har bir xesh uchun bir nechta turli xil ma'lumotlar

to'plamlari bo'ladi. Bunday moslik ***kolliziya*** deb ataladi. Agar biron bir muammoni hal qilishda kirish ma'lumotlari cheklangan bo'lsa, siz bunday xeshlar to'plamini tanlashingiz mumkin, shunda uning aniqligi kirish ma'lumotlari to'plamining muhimligidan oshib ketadi. Bunday holda, biz inyeksion xaritalashni aniqlaydigan xesh funksiyasini qurishimiz mumkin (mukammal xeshlash). Biroq, umuman olganda, kolliziya muqarrar. Kolliziya ehtimoli xesh funksiyasi sifatini baholash uchun ishlatiladi. Yaxshi xesh funksiyasi quyidagicha ishlaydi:

- mavjud bo'lgan barcha xesh oralig‘i maksimal darajada ishlatiladi;

- kirish ma'lumotlarining ozgina o'zgarishi ham mutlaqo boshqacha xeshni berishi kerak, to'qnashuvlar faqat butunlay boshqacha ma'lumotlar uchun ro'y berishi kerak.

Xeshlash o'zi obyektga tasodifiy o'zgaruvchini xaritalashga o'xshaydi. Birinchi xususiyat natijasida xeshlar o'zlarini bir tekis taqsimlangan tasodifiy o'zgaruvchilar kabi tutishi kerak, bu butun diapazondan foydalanishni ta'minlaydi, bu foydali bo'lishi mumkin, masalan, xesh jadvalini tuzishda.

Polinomial xeshlash. Quyida oddiy, ammo samarali xeshlash algoritmini ko'rib chiqamiz. Xesh funksiyamizni quyidagicha aniqlaylik:

$$h(s) = \sum_{i=0}^{N} b^{N-i} \cdot \text{code}(s_i) \qquad (1)$$

yoki

$$h(preff_i) = b \cdot h(pref_{i-1}) + code(s_{i-1}), \qquad (2)$$

bu yerda $N = strlen(s) - 1,\ pref_i -$ uzunlik prefiksi i, b-baza, asos. $code(s_i)$ −simvol kodi.

Agar (1) formulani kengaytirsak, biz N tartibli polinomni olamiz. (2) formula xeshni rekursiv shaklda o'rnatadi va kod yozishda foydalaniladi.

Belgilar kodiga va asosga e'tibor qaratish lozim, chunki bazani tanlash kodlarga bog‘liq bo'ladi. Kod ASCII jadvalidagi belgilar kodi yoki alfavitdagi tartib raqam bo'lishi mumkin. Masalan, agar muammo har qanday satr ingliz alifbosining faqat kichik harflaridan iborat bo'lishiga kafolat beradigan bo'lsa, unda tartib raqami belgilar kodlari uchun yaxshi imkoniyatdir. Simvollar satrdagi mumkin bo'lgan har

qanday belgining maksimal kodidan oshib ketishi kerak va odatda asosiy son tanlanadi (garchi raqamning soddaligi uchun qat'iy talablarga javob bermagan bo'lsa ham). Masalan, 31, 37 va boshqalar asoslari inglizcha kichik harflarning satrlari uchun javob beradi.

Shunga qaramay, shuni ta'kidlash joizki, biz xeshni hech qanday cheklamaymiz, bu xeshlash ta'rifiga ziddir. Bunday holda, ikkita chiqish usuli mavjud: modul bo‘yicha bo‘lish amalidan yoki uzun arifmetikadan foydalanish.

Birinchi variant uzun arifmetikaga ega bo'lmagan tillarda keng qo'llaniladi. Bundan tashqari, xesh saqlanadigan butun sonli ma'lumotlar turi bu bo'linishni avtomatik ravishda amalga oshiradi (turlarning ko'payishi natijasida qo'shimcha bitlar avtomatik ravishda yo'qoladi). Natijada biz cheklangan xeshlar to'plamini olamiz, ammo yana kolliziya xavfi mavjud. Bundan tashqari, ko'p polinomli xeshni "buzish" ehtimoli mavjud.

Ikkinchi variantda kolliziya ehtimoli pastroq. Biroq, kattaroq xeshlar to'plamini qo'llab-quvvatlash, qo'shimcha xotira va ikkita xeshni taqqoslash uchun zarur bo'lgan vaqtni talab qiladi, bu oddiy ma'lumotlarni taqqoslashdan ko'ra tezroq.

Masalan, satrni inglizcha kichik harflardan iborat deb taxmin qilamiz. Quyida 37 raqamini asos qilib olamiz.

```
#include <iostream>
#include <string.h>
using namespace std;

long long Heshlash(char s[])
{
   long long h = 0;
   int base = 37;
   for(int i=0; i<=strlen(s); i++)
   {
      h = h* base + s[i] - 61 +1;
   }
   return h;
}
```

```
int main()
{
   char s[100];

   for(int i=1; i<10; i++)
   {
      cin.getline(s,100);
      cout<<s<<" "<<Heshlash(s);
      cout<<endl;
   }
}
```

9-jadval.

Xesh jadvallardan foydalanish samaradorligi

Konteyner / Amal	**Insert (qo‘shish)**	**Remove (O‘chirish)**	**Izlash (find)**
Massiv	O(N)	O(N)	O(N)
Ro‘yxat	O(1)	O(1)	O(N)
Saralangan massiv	O(N)	O(N)	O(logN)
Ikkilik qidiruv daraxti	O(logN)	O(logN)	O(logN)
Xesh-jadval	O(1)	O(1)	O(1)

Barcha ma'lumotlar yaxshi bajarilgan konteynerlarni, yaxshi tanlangan xesh funksiyalarini taqdim etdi.

Ushbu jadvaldan nega xesh jadvallardan foydalanish kerakligi juda aniq ko‘rinib turibdi. Ammo keyin qarama-qarshi savol tug‘iladi: nega ular doimo ishlatilmaydi? Javob juda sodda: har doimgidek, birdaniga hamma narsani olish mumkin emas, ya'ni: ham tezlikdan, ham xotiradan yutib bo‘lmaydi. Xesh jadvallari noqulay va ular operatsion jarayonning

asosiy savollariga tezda javob berishlari bilan birga, ulardan foydalanish har doim juda qimmatga tushadi.

13.2. C++ dasturlash tilida xesh jadvallarni realizatsiya qilish

C++ dasturlash tilida xesh jadvallarni hosil qilish uchun map konteyneri aniqlangan. map konteyner vector, list, deque kabi boshqa konteynerlarga juda o'xshaydi, lekin ozgina farqi mavjud. Bu konteynerga birdaniga ikkita qiymat qo'yish mumkin. Shunday qilib, bu map misolni batafsil ko'rib chiqaylik:

```
#include <iostream>
#include <map>    //map bilan ishlash uchun kutubxonani ulash
using namespace std;
int main()
{
///map oshkor initsializatsiyalash
  map <string,int> myFirstMap = {{ "Mother", 37 },
                                 { "Father", 40 },
                                 { "Brother", 15 },
                                 { "Sister", 20 }};

  /// initsializatsiyalangan mapni ekranga chiqarish
  for (auto it = myFirstMap.begin(); it != myFirstMap.end(); ++it)
  {
     cout << it->first << " : " << it->second << endl;
  }

  char c;
  map <char,int> mySecondMap;
  for (int i = 0,c = 'a'; i < 5; ++i,++c)
  {
     mySecondMap.insert ( pair<char,int>(c,i) );
  }

  /// initsializatsiyalangan mapni ekranga chiqarish
  for (auto it = mySecondMap.begin(); it != mySecondMap.end(); ++it)
  {
```

```
    cout << (*it).first << " : " << (*it).second << endl;
  }
  return 0;
}
```

Map bilan bog‘liq ba'zi asosiy funksiyalar quyida keltirilgan:

begin() - iteratorni mapdagi birinchi elementga qaytaradi

end() - iteratorni mapdagi oxirgi elementdan keyingi nazariy elementga qaytaradi

size() - mapdagi elementlar sonini qaytaradi

max_size() - mapda saqlanishi mumkin bo'lgan elementlarning maksimal sonini qaytaradi

empty() - mapning bo'shligini tekshiradi

pair_insert(keyvalue, mapvalue) - mapga yangi element qo'shiladi

erase(iterator position) - elementni iterator ko'rsatgan joydan olib tashlaydi

erase(const g) - mapdan "g" kalit qiymatini olib tashlaydi

clear() - mapdagi barcha elementlarni olib tashlaydi

Kolliziya muammosi. Tabiiyki, savol tug‘iladi, nega biz bir qator katakchaga ikki marta kirib olishimiz mumkin emas, chunki har bir elementga mutlaqo boshqacha natural sonlarni taqqoslaydigan funksiyani taqdim etish shunchaki mumkin emas. Kolliziya muammosi xesh funksiyasi turli elementlar uchun bir xil natural sonni hosil qilganda paydo bo‘ladigan muammo.

Ushbu muammoning bir nechta yechimlari mavjud: zanjirlash usuli va ikki marta xeshlash usuli.

Mavzu yuzasidan savollar:

1. Xesh jadval nima?
2. Xesh jadvallardan foydalanish samaradorligini taqqoslang
3. Xesh funksiyasiga misol keltiring
4. Satrlar uchun xesh funksiyasini qo‘llang

5. Xesh funksiya ma’lumotlar strukturasi qo‘llaniladigan sohalarga qaysilar kiradi?

Mustaqil ishlash uchun masalalar:

1. C++ tilida xesh jadvallarni hosil qiling.

2. C++da xesh jadvallarning metodlarini qo‘llang

14-§. Xesh funksiya

Xesh funksiyalar – ixtiyoriy uzunlikdagi kirish ma’lumotini chiqishda belgilangan uzunlikdagi xesh qiymatga aylantirib beruvchi bir tomonlama funksiyalarga aytiladi. Xesh funksiyalar kriptografiya va zamonaviy axborot xavfsizligi sohasida ma’lumotlarni to‘laligini tekshirishda foydalaniladi. Elektron to‘lov tizimlari protokollarida ham istemolchi kartasi ma’lumotlarini bank-emitentga to‘liq yetkazish uchun foydalaniladi.

Xesh funksiya – ixtiyoriy uzunlikdagi M-ma’lumotni fiksirlangan uzunlikga siqish yoki ikkilik sanoq sistemasi ifodalangan ma’lumotlarni fiksirlangan uzunlikdagi bitlar ko‘rinishidagi qandaydir kombinatsiyasi (svertkasi) deb ataluvchi funksiya.

Ta’rif. Xesh-funksiya deb, har qanday

$$h: X \to Y$$

oson hisoblanuvchi va $\forall M$ –ma’lumot uchun h(M) = H fiksirlangan uzunlikga ega bo‘lgan funksiyaga aytiladi.

Berilgan M-ma’lumotning h(M) –xesh qiymatini topish uchun avvalo ma’lumot biror «m» -uzunlikdagi bloklarga ajratilib chiqiladi. Agar M-ma’lumot uzunligi «m» -ga karrali bo‘lmasa, u holda oxirgi to‘lmay qolgan blok «m»- uzunlikga olindan kelishib olingan maxsus usulda biror simvol yoki belgi (masalan “0” yoki “1”) bilan to‘ldirilib chiqiladi. Natijada hosil qilingan M-ma’lumot bloklariga:

$$M= \{ M_1, M_2 ,.....M_n)$$

quyidagicha siqishni (svertkani) hisoblash protsedurasi qo‘llaniladi:

$$H_0 = \nu ,$$

$$H_i = f (M_i , H_{i-1}) , \quad i =1,2,.....n$$

$h(M)= H_n$;

bu yerda ν -qandaydir fiksirlangan boshlang‘ich vektor.

Misol sifatida quyidagi keng tarqalgan:

$$f (M_i , H_{i-1}) = E_k(M_i \oplus H_{i-1}) \quad i =1,2,.....n$$

xesh-funksiyani keltirib o‘tish mumkin.

Bu yerda E-simmetrik shifrlash algoritmi (masalan DES, GOST 28147-87, AES –FIPS 197 va hakoza), k- esa shifrlash algoritmi maxfiy kaliti, $H_0 = 0$, $\oplus$ - XOR (mod 2 bo‘yicha mos bitlarni qo‘shish) amali.

14.1. Xesh funksiyalar turlari

Oddiy xesh funksiyalar: Adler-32, CRC, FNV, Murmur2, PJW-32, TTH, Jenkins hash.

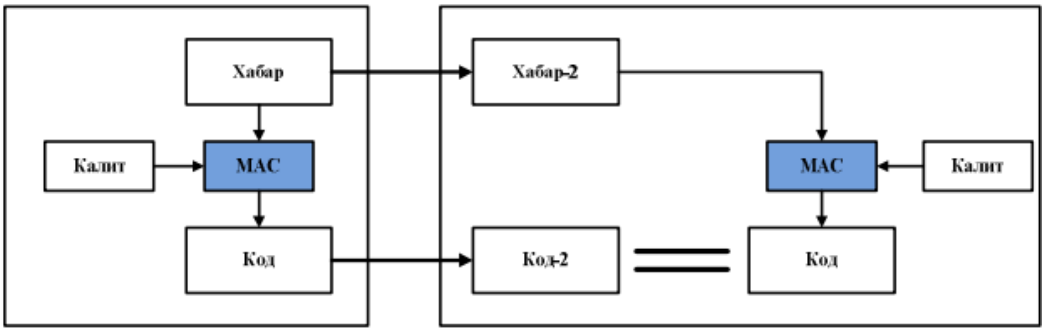
Kriptografik xesh funksiyalar: CubeHash, BLAKE, BMW, ECHO, FSB, Fugue, Grøstl, JH, Hamsi, HAVAL, Keccak (SHA-3), Kupyna, LM-хеш, Luffa, MD2, MD4, MD5, MD6, N-Hash, RIPEMD-128, RIPEMD-160, RIPEMD-256, RIPEMD-320, SHA-1, SHA-2, SHABAL, SHAvite-3, SIMD, Skein, Snefru, SWIFFT, Tiger, Whirlpool, ГОСТ Р 34.11-94, ГОСТ Р 34.11-2012.

Kalit hosil qiluvchi xesh funksiyalar: bcrypt, PBKDF2, scrypt.

Kriptografik xesh funksiyalarning esa quyidagi turlari mavjud:

1) kalitli xesh funksiya; 2) kalitsiz xesh funksiya.

Kalitli xesh funksiyalar simmetrik shifrlash algoritmi tizimlarida qo‘llaniladi. Kalitli xesh funksiyalar berilgan ma’lumot autentifikatsiyasi kodi (message authentication code(MAC)) deb ham yuritiladi. Ushbu kod bir-biriga ishonchi mavjud foydalanuvchilarga berilgan ma’lumotining haqiqiyligi va to‘laligini kafolatini qo‘shimcha vositalarsiz ta’minlash imkoniyatini tug‘diradi.



57-rasm. MAC tizimlari

Kalitsiz xesh funksiyalar xatolarni topish kodi (modification detection code(MDC) yoki manipulation detection code, massage integrrity code(MIC) deb ataladi. Ushbu kod qo‘shimcha vositalar (masalan: himoyalangan aloqa tarmog‘i, shifrlash yoki ERI algoritmlari) yordamida berilgan ma’lumot to‘laligini kafolatlaydi. Bu turdagi xesh funksiyalardan bir-biriga ishonch bildiruvchi va ishonchi bo‘lmagan tomonlar foydalanishlari mumkin.

Odatda kalitsiz xesh funksiyalardan quyidagi xossalarni qanoatlantirishi talab qilinadi:

1) bir tomonlilik;

2) kolliziyaga bardoshlilik;

3) xesh qiymatlari teng bo‘lgan ikkita ma’lumotni topishga bardoshlilik.

Birinchi shart bajarilganda, berilgan xesh qiymatga ega bo‘lgan ma’lumotni topishning murakkab ekanligini, ikkinchi shart bajarilganda bir xil xesh qiymatga ega bo‘lgan ma’lumotlar juftini topishning murakkab ekanligini, uchinchi shart xesh qiymati ma’lum bo‘lgan berilgan ma’lumot uchun xesh qiymati shunga teng bo‘lgan ikkinchi ma’lumotni topishning murakkab ekanligini bildiradi.

Masalan, nazorat yig‘indini topuvchi SRC xesh funksiyasi chiziqli akslantirish bo‘ladi va shuning uchun ham bu uchta shartdan birontasini ham qanoatlantirmaydi.

Ma’lumotlarni uzatishda yoki saqlashda ularning to‘laligini nazoratlashda har bir ma’lumotning xesh qiymati (bu xesh qiymat ma’lumotni autentifikatsiya qilish kodi yoki “imitoqo‘yish”-ma’lumot bloklari bilan bog‘liq bo‘lgan qo‘shimcha kiritilgan belgi deyiladi) hisoblaniladi va bu qiymat ma’lumot bilan birga saqlaniladi yoki uzatiladi. Ma’lumotni qabul qilgan foydalanuvchi ma’lumotning xesh qiymatini hisoblaydi va uning xesh qiymati bilan solishtiradi. Agar taqqoslashda bu qiymatlar mos kelmasa, ma’lumot butunligi buzilganligini anglatadi.

“Imitoqo‘yish”lar hosil qilish uchun foydalaniladigan xesh funksiyalar nazorat yig‘indisidan farqli ravishda ma’lumotni saqlash va uzatishda ro‘y beradigan tasodifiy xatolarni aniqlabgina qolmasdan,

raqib tomonidan qilingan aktiv hujumlar to‘g‘risida ham ogohlantiradi. Buzg‘unchi xesh qiymatni osonlik bilan o‘zi hisoblab topa olmasligi va muvaffaqiyatli imitatsiya qilishi yoki ma’lumotni o‘zgartira olmasligi uchun xesh funksiya 70 buzg‘unchiga ma’lum bo‘lmagan maxfiy kalitga ega bo‘lishi kerak. Bu maxfiy kalit faqatgina ma’lumotni uzatuvchi va qabul qiluvchi tomonlarga ma’lum bo‘lishi kerak. Bunday xususiyatga ega xesh funksiyalarga kalitli xesh funksiyalar deyiladi. Kalitli xesh funksiyalar yordamida hosil qilinadigan “imitoqo‘yish”lar imitatsiya (impersonation) turidagi hujumlarda qalbaki ma’lumotlarni hosil qilishga (fabrication) va “o‘zgartirish” (substitution) turidagi hujumlarda uzatiladigan ma’lumotni modifikatsiya (modification) qilishga yo‘l qo‘ymaslikda foydalaniladi.

Ma’lumot manbaining autentifikatsiyalash masalasi axborot-kommunikatsiya tizimlarining bir-biriga ishonmaydigan ikki tomoni orasida ma’lumot almashinuvida yuzaga keladi. Bu masalani hal qilishda ikkala tomon ham biladigan maxfiy kalitdan foydalanib bo‘lmaydi. Bu holatda ma’lumotning manbaini autentifikatsiya qilishga imkon beradigan elektron raqamli imzo sxemasi qo‘llaniladi. Bunda odatda foydalanuvchining maxfiy kalitiga asoslangan imzo qo‘yishdan oldin xatolik kodini aniqlovchi xesh funksiya yordamida ma’lumot siqiladi. Bu holda xesh funksiya maxfiy kalitga ega bo‘lmaydi hamda u fiksirlangan bo‘lishi va hammaga ma’lum bo‘lishi mumkin. Unga qo‘yilgan asosiy talab imzolangan hujjatni o‘zgartirish hamda bir xil xesh qiymatga ega bo‘lgan ikkita har xil ma’lumotni tanlash imkoniyati yo‘qligining kafolatidir.

Agar bir xil xesh qiymatga ega bo‘lgan ikkita har xil ma’lumot mavjud bo‘lsa, bu ma’lumotlar jufti kolliziya hosil qiladi deyiladi.

Xesh funksiyalarda kolliziya – ikkita har xil ma’lumotdan bir xil xesh qiymat hosil bo‘lib qolishi. Kolliziyaning oldini olish yo‘llaridan biri bu xesh jadval hisoblanadi. Xeshlash algoritmlarining bardoshliligi xa xavfsizliligi kolliziyaga chidamliligi bilan aniqlanadi.

14.2. Xesh funksiyalar qo‘llanilishi va axborot xavfsizligidagi o‘rni

Xeshlash algoritmlarining zamonaviy kriptografiyadagi tutgan o‘rni juda muhimdir va undan hozirda keng ko‘lamda foydalaniladi.

Email clients		Apple Mail · Claws Mail · Enigmail · GPG (Gpg4win) · Kontact · Outlook · p≡p · PGP · Sylpheed · Thunderbird
Secure Messaging	**OTR**	Adium · BitlBee · Centericq · ChatSecure · climm · Jitsi · Kopete · MCabber · Profanity
	SSH	Dropbear · lsh · OpenSSH · PuTTY · SecureCRT · WinSCP
	TLS & SSL	Bouncy Castle · BoringSSL · Botan · cryptlib · GnuTLS · JSSE · LibreSSL · MatrixSSL · NSS · OpenSSL · mbed TLS · RSA BSAFE · SChannel · SSLeay · stunnel · wolfSSL
	VPN	Check Point VPN-1 · Hamachi · Openswan · OpenVPN · SoftEther VPN · strongSwan · Tinc
	ZRTP	CSipSimple · Jitsi · Linphone · Ring · Zfone
	P2P	Bitmessage · RetroShare · Tox
	DRA	Matrix · OMEMO (Conversations · Cryptocat · ChatSecure) · Proteus · Signal Protocol (Google Allo · Facebook Messenger · Signal · TextSecure · WhatsApp)
Disk encryption (Comparison)		BestCrypt · BitLocker · CipherShed · CrossCrypt · Cryptoloop · DiskCryptor · dm-crypt · DriveSentry · E4M · eCryptfs · FileVault · FreeOTFE · GBDE · geli · LUKS · PGPDisk · Private Disk · Scramdisk · Sentry 2020 · TrueCrypt (History) · VeraCrypt
Anonymity		GNUnet · I2P · Java Anon Proxy · Tor · Vidalia · RetroShare · Ricochet
Cryptographic file systems		Freenet · Tahoe-LAFS
Educational		CrypTool
Related topics		Outline of cryptography · Timeline of cryptography · **Hash functions** (Cryptographic hash function · List of hash functions) · S/MIME

58-rasm. Kriptografik xesh funksiyalar ishlatilishi

Yangi xesh algoritmlar xam yaratilmoqda. Yangi xesh algoritmlar kolliziyaga bardoshli, xesh qiymatning tez hisob-kitob qila olishi va.h.k xususiyatlarga ega bo‘ladi.

Xesh funksiyalar asosan, Elektron raqamli imzo (ERI)da, Torrent, DC Hub, Operatsion sistemalarda va fayllarning butunliligini yoki o‘zgartirilganligini nazorat qilish uchun foydalaniladi. Axborot butunligini nazorat qilishning ko‘proq maqbul bo‘lgan metodlaridan biri xesh-funksiyadan foydalanish hisoblanadi. Xesh-funksiyaning qiymatini uning kalitini bilmasdan turib qalbakilashtirib bo‘lmaydi, shu sababli xeshlash kalitini shifrlangan ko‘rinishda yoki jinoyatchining «qo‘li yetmaydigan» joydagi xotirada saqlash kerak.

10-jadval.

Xesh funksiyalar tahlili

	Xeshlanadigan matn uzunligi	Kirish blokining uzunligi	Xesh qiymat uzunligi	Har bir blokni xeshlash qadamlari soni
GOST R 34.11-94	Ixtiyoriy	256	256	19
MD 2	Ixtiyoriy	512	128	1598
MD 4	Ixtiyoriy	512	128	72
MD 5	Ixtiyoriy	512	128	88
SHA-1	$<2^{64}$	512	160	80

CRC32 (Cyclic redundancy check – Davriy kamchilikni tekshiruvchi kod) kompyuter qurilmalarida, ya'ni tarmoq qurilmalari va doimiy xotiradagi ma'lumotlarni xavfsizligini ta'minlashda ya'ni o'zgartirilmaganligini doimiy ravishda tekshirib boradigan oddiy xesh funksiya hisoblanadi. CRC32 xalqaro standarti CRC32-IEEE 802. Bu algoritm juda tez ishlagani bilan, kriptoxavfsizlikni to'liq ta'minlay olmaydi. Shunga qaramasdan keng qo'llaniladi chunki, ishlatilishi juda oddiy va tez. 32-bit xesh-kod odatda 8 ta simvoldan iborat 16 lik sanoq sistemasida ifodalanadi. Bu algoritm kriptografik hisoblanmaydi.

SHA-256	$<2^{64}$	512	256	64
SHA-384	$<2^{128}$	1024	384	80
SHA-512	$<2^{128}$	1024	512	80
STB 1176.1 – 99	Ixtiyoriy	256	$142 \le L \le 256$	77
O'z DSt 1106 : 2006	Ixtiyoriy	128, 256	128, 256	16b+74, 16b+46, Bu erda b-bloklar soni

MD4 xeshlash algoritmi RSA Data Security, Inc. Ronald L. Rivest tomonidan ishlab chiqilgan. MD4 aralashgan algoritm hisoblanadi, Endi ishonchsiz hisoblanadi. Bu algoritm (32-bit protsessorlari uchun) tez va peer-to-peer tarmog'i edonkey 2000 Qo'shma Algoritm hash kodi 32 ta simvoldan iborat bo'lgan belgilar bilan o'n oltilik soni RFC 1320. tasvirlangan hisoblash ishlatiladi.

MD5 xesh funksiyasi algoritmi Massachusets texnologiya instituti professori Ronald Rivest tomonidan 1992 yilda ishlab chiqilgan. Bu

algoritmda kiruvchi ma’lumot uzunligi ixtiyoriy bo‘lib, xesh qiymat uzunligi 128 bit bo‘ladi. MD 5 xesh funksiyasi algoritmida kiruvchi ma’lumot 512 bitlik bloklarga ajratilib, ular 16 ta 32 bitlik qism bloklarga ajratiladi va bular ustida amallar bajariladi. Faraz qilaylik, bizga uzunligi b bit bo‘lgan, bu yerda b – ixtiyoriy nomanfiy butun son, ma’lumot berilgan bo‘lsin va bu ma’lumotning bitlari quyidagicha: *m0m1…m(b-1)*

SHA-1 xesh funksiyasi algoritmi. Kafolatlangan bardoshlilikka ega bo‘lgan xeshlash algoritmi SHA (Secure Hash Algorithm) AQShning standartlar va texnologiyalar Milliy instituti (NIST) tomonidan ishlab chiqilgan bo‘lib, 1992 yilda axborotni qayta ishlash federal standarti (RUB FIPS 180) ko‘rinishida nashr qilindi. 1995 yilda bu standart qaytadan ko‘rib chiqildi va SHA-1 deb nomlandi (RUB FIPS 180-1). SHA algoritmi MD4 algoritmiga asoslanadi va uning tuzilishi MD4 algoritmining tuzilishiga juda yaqin. Bu algoritm DSS standarti asosidagi elektron raqamli imzo algoritmlarida ishlatish uchun mo‘ljallangan. Bu algoritmda kiruvchi ma’lumotning uzunligi 264 bitdan kichik bo‘lib, xesh qiymat uzunligi 160 bit bo‘ladi. Kiritilayotgan ma’lumot 512 bitlik bloklarga ajratilib qayta ishlanadi.

Xesh qiymatni hisoblash jarayoni quyidagi bosqichlardan iborat:

1-bosqich. *To‘ldirish bitlarini qo‘shish.*

Berilgan ma’lumot uzunligi 512 modul bo‘yicha 448 bilan taqqoslanadigan (ma’lumot uzunligi ≡ 448 mod 512) qilib to‘ldiriladi. To‘ldirish hamma vaqt, hattoki ma’lumot uzunligi 512 modul bo‘yicha 448 bilan taqqoslanadigan bo‘lsa ham bajariladi. To‘ldirish quyidagi tartibda amalga oshiriladi: ma’lumotga 1 ga teng bo‘lgan bitta bit qo‘shiladi, qolgan bitlar esa 0 lar bilan to‘ldiriladi. Shuning uchun qo‘shilgan bitlar soni 1 dan 512 tagacha bo‘ladi.

2-bosqich. *Ma’lumotning uzunligini qo‘shish.*

1-bosqichning natijasiga berilgan ma’lumot uzunligining 64 bitlik qiymati qo‘shiladi.

3-bosqich. *Xesh qiymat uchun bufer initsializatsiya qilish.*

Xesh funksiyaning oraliq va oxirgi natijalarini saqlash uchun 160 bitlik buferdan foydalaniladi. Bu buferni beshta 32 bitlik A, B, C, D, E

registrlar ko‘rinishida tasvirlash mumkin. Bu registrlarga 16 lik sanoq sistemasida quyidagi boshlang‘ich qiymatlar beriladi:

A=0x67452301,
B=0xEFCDAB89,
C=0x98BADCFE,
D=0x10325476,
E=0xC3D2E1F0.

Keyinchalik bu o‘zgaruvchilar mos ravishda yangi *a*, *b*, *c*, *d* va *e* o‘zgaruvchilarga yozib olinadi.

4- bosqich. *Ma’lumotni 512 bitlik bloklarga ajratib qayta ishlash.*

Bu xesh funksiyaning asosiy sikli quyidagicha bo‘ladi:

for (t = 0; t < 80; t++){
temp = (a <<< 5) + *ft*(b, c, d) + e + Wt + Kt ;
e = d; d = c; c = b <<< 30; b = a; a = *temp*;
},

Bu yerda <<< - chapga siklik surish amali. K*t* lar 16 lik sanoq sistemasida
yozilgan quyidagi sonlardan iborat:

$$\mathbf{K}_t = \begin{cases} \mathbf{5A827999}, & t = 0,\ldots,19, \\ \mathbf{6ED9EBA1}, & t = 20,\ldots,39, \\ \mathbf{8F1BBCDC}, & t = 40,\ldots,59, \\ \mathbf{CA62C1D6}, & t = 60,\ldots,79. \end{cases}$$

ft(x, y, z) funksiyalar esa quyidagi ifodalar bilan aniqlanadi:

$$f_t(\mathbf{x},\mathbf{y},\mathbf{z}) = \begin{cases} \mathbf{X} \wedge \mathbf{Y} \vee \neg\mathbf{X} \wedge \mathbf{Z}, & \mathbf{t} = 0,\ldots,19, \\ \mathbf{X} \oplus \mathbf{Y} \oplus \mathbf{Z}, & \mathbf{t} = 20,\ldots,39,\ 60,\ldots,79, \\ \mathbf{X} \wedge \mathbf{Y} \vee \mathbf{X} \wedge \mathbf{Z} \vee \mathbf{Y} \wedge \mathbf{Z}, & \mathbf{t} = 40,\ldots,59. \end{cases}$$

Wt lar kengaytirilgan ma’lumotning 512 bitlik blokining 32 bitlik qism bloklaridan quyidagi qoida bo‘yicha hosil qilinadi:

$$\mathrm{W}_t = \begin{cases} \mathrm{M}_t, & t = 0,\ldots,15, \\ (\mathrm{W}_{t-3} \oplus \mathrm{W}_{t-8} \oplus \mathrm{W}_{t-14} \oplus \mathrm{W}_{t-16}) <<< 1, & t = 16,\ldots,79. \end{cases}$$

Asosiy sikl tugagandan keyin *a*, *b*, *c*, *d* va *e* larning qiymatlari mos ravishda A, B, C, D va E registrlardagi qiymatlarga qo‘shiladi hamda

shu registrlarga yozib qo‘yiladi va kengaytirilgan ma’lumotning keyingi 512 bitlik blokini qayta ishlashga o‘tiladi.

5- bosqich*. Natija.*

Ma’lumotning xesh qiymati A, B, C, D va E registrlardagi qiymatlarni birlashtirish natijasida hosil qilinadi.

Mavzu yuzasidan savollar:

1. Xesh funksiya tushunchasiga ta’rif bering.
2. Kriptografik xesh funksiyalarga misol keltiring
3. Xesh funksiyalarning yana qanday turlarini bilasiz
4. Kalit hosil qiluvchi xesh funksiyalarni keltiring

15-§. Graflarda eng kichik uzunlikdagi daraxtlarni qurish algoritmlari

Eng kichik uzunlikdagi daraxt – berilgan grafning eng kam darajaga ega bo‘lgan daraxti, bu yerda daraxtning darajasi uning qirralari daajalari yig‘indisi sifatida tushuniladi.

Misol. Minimal uzunlikdagi daraxtni topish muammosi ko'pincha xuddi shunday sharoitda uchraydi: masalan, har qanday shahardan boshqasiga (to'g‘ridan-to'g‘ri yoki boshqa shaharlar orqali) o'tish uchun ***n ta*** shaharlarni yo'llar bilan bog‘lash kerak. Berilgan juft shaharlar o'rtasida yo'llar qurishga ruxsat beriladi va har bir bunday yo'lni qurish qiymati ma'lum. Qurilishning umumiy narxini minimallashtirish uchun qaysi yo'llarni qurish kerakligini hal qilish talab qilinadi.

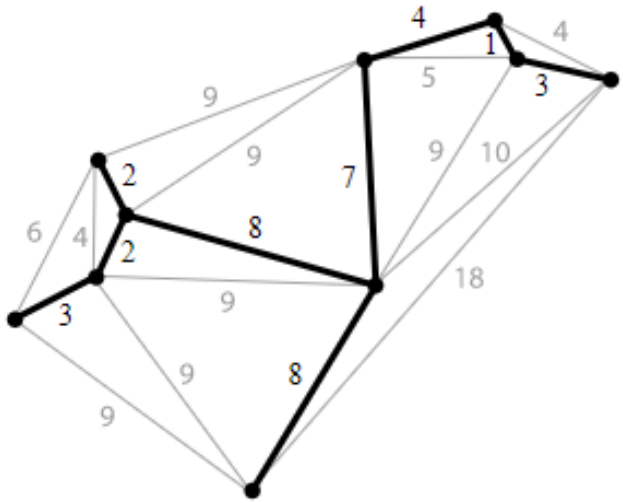
Ushbu muammoni grafika nazariyasi nuqtai nazaridan shakllantirish mumkin. Bu yerda berilgan grafnin uchlari shaharlarni, qirralari esa to'g‘ri yo'l qurilishi mumkin bo'lgan va qirralarning og‘irliklari teng bo'lgan shaharlarni ifodalaydigan minimal daraxtini topish muammosidir.

Minimal uzunlikdagi daraxtni topish uchun bir nechta algoritmlar mavjud. Eng mashhurlari quyida keltirilgan:

1) Prima algoritmi;
2) Kruskal algoritmi;

3) Boruvka algoritmi,
4) Orqadan o'chirish algoritmi

Quyida ushbu algoritmlarni ko‘rib chiqamiz.



59-rasm. Eng kichik uzunlikka ega bo‘lgan daraxt

15.1. Kruskal algoritmi

Kruskal algoritmida qirralarning butun birlashtirilgan ro'yxati kamaymaydigan uch darajalariga muvofiq tartiblangan. Bundan tashqari, qirralar darajalari kichikroq bo'lgan qirralardan yuqori tomonga siljiydi va keyingi uch ilgari tanlangan qirralar bilan sikl hosil qilmasa, karkasga qo'shiladi. Xususan, grafdagi minimal darajadagi qirralaridan biri har doim birinchi bo'lib tanlanadi.

Tanlangan qirralarning sikl hosil qilmasligini tekshirish uchun biz grafni bir nechta bog‘langan komponentlarning birlashishi sifatida namoyish etamiz. Eng boshida, grafning chekkalari tanlanmaganida, har bir uch alohida bog‘langan komponent hisoblanadi. Yangi qirralar qo'shilganda, ulanish komponentlari bitta umumiy ulanish komponenti bo'lguncha birlashadi. Barcha bog‘langan tarkibiy qismlarni raqamlaymiz va har bir uch uchun uning ulangan qismlarini sonini saqlaymiz, shuning uchun har bir uch uchun boshida uning bog‘langan komponentlari soni uchning o'zi soniga teng bo'ladi va oxirgi barcha

uchlar bir-biriga bog‘langan komponentning bir xil raqamlariga tegishli bo'ladi.

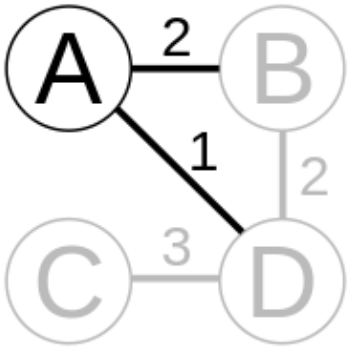
Keyingi qirrani ko'rib chiqayotganda, ushbu qirraning uchlariga to'g‘ri keladigan ulangan komponentlarning raqamlarini ko'rib chiqamiz. Agar bu raqamlar bir xil bo'lsa, unda qirra allaqachon bir xil bog‘langan komponentda joylashgan ikkita uchni birlashtiradi, shuning uchun bu qirrani qo'shish siklni tashkil qiladi. Agar qirra ikki xil bog‘langan komponentni, masalan, a va b raqamlari bilan bog‘lasa, u holda qirra asosiy daraxtning bir qismiga qo'shiladi va bu ikkita bog‘langan komponentlar birlashtiriladi. Buning uchun, masalan, ilgari b komponentida bo'lgan barcha uchlar uchun komponent raqamini a ga o'zgartirish kerak.

Kruskal algoritmini amalga oshirish bosqichlari quyidagicha:

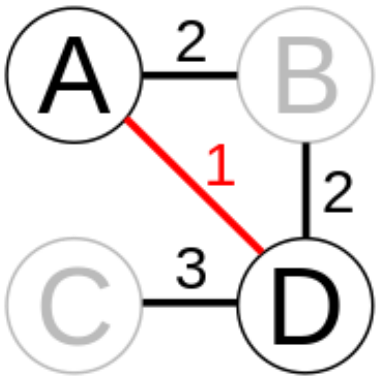
1) Barcha qirralarni quyidan yuqorigacha saralash.
2) Vazni eng kichik qirrasini oling va uni daraxtga qo'shing. Agar qirra qo‘shilganda, sikl hosil bo‘lsa, u holda bu qirrani olib tashlang.
3) Barcha uchlarga yetguncha qirralarni qo'shishni davom eting.

Quyidagi rasmda minimal uzunlikka kiritilgan qirralar qizil rang bilan, qora rang bilan esa nomzodlar ulardan eng kam darajadagi qirra tanlangan.

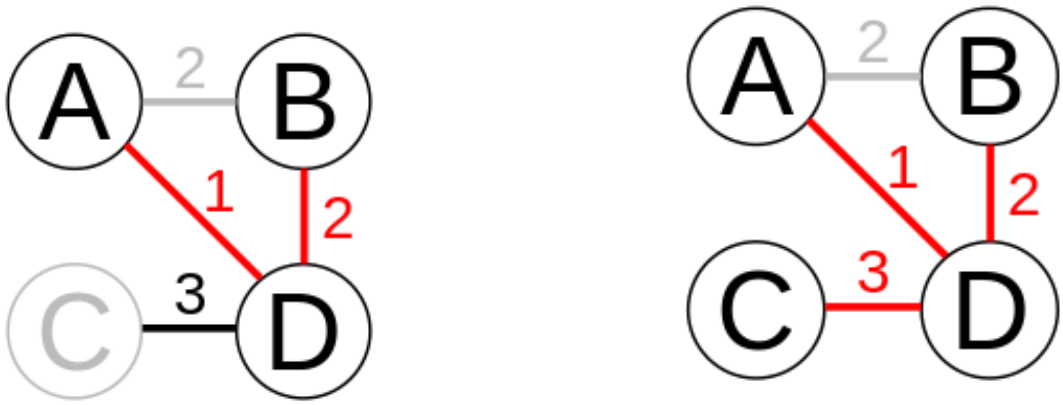
1-qadam



2-qadam



3-qadam *4-qadam. (So‘nggi natija)*



60-rasm. Kruskal algoritmining bajarilish ketma-ketligi

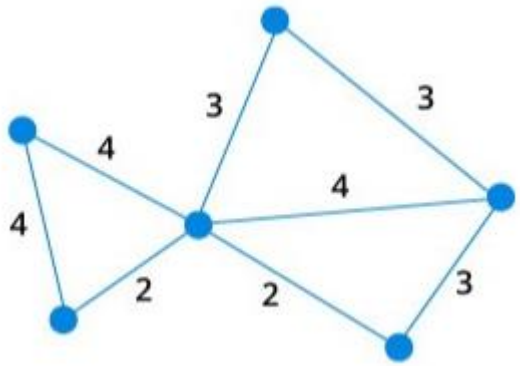
Kruskal algoritmini realizatsiya qilish (C++ kodi)

```
int n, m;
 cin >> n >> m;
 vector <vector <int> > edges(m, vector<int>(3));
 for (int i = 0; i < m; ++i)
    cin >> edges[i][1] >> edges[i][2] >> edges[i][0];
 sort(edges.begin(), edges.end());
 vector <int> comp(n);
for (int i = 0; i < n; ++i)
    comp[i] = i;
 int ans = 0;
for (auto & edge: edges)
{
    int weight = edge[0];
    int start  = edge[1];
    int end    = edge[2];
    if (comp[start] != comp[end])
    {
       ans += weight;
       int a = comp[start];
       int b = comp[end];
       for (int i = 0; i < n; ++i)
          if (comp[i] == b)
```

```
            comp[i] = a;
        }
    }
```

15.2. Prima algoritmi

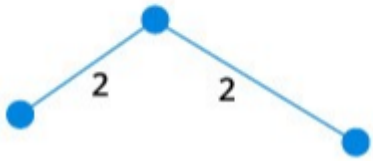
Prima algoritmi quyidagi tartibda ishlaydi



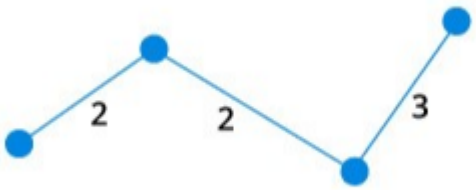
Dastlabki berilgan graf

1-bosqich. Uchni tanlash

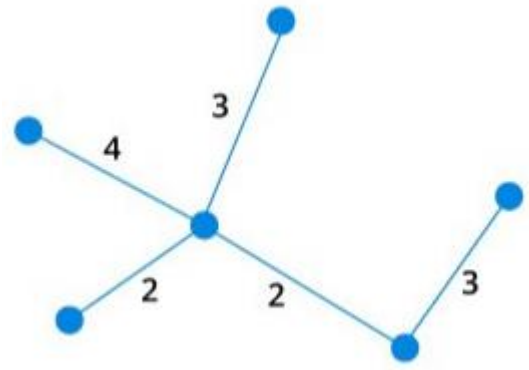
2-bosqich. Ushbu uchdan eng qisqa qirrani tanlash va uni qo'shish



3-bosqich. Grafdan hali tanlanmagan eng yaqin uchni tanlash



4-bosqich. Grafda hali topilmagan eng yaqin uchni tanlang, agar bir nechta variant, tasodifiy birini tanlash



Keyingi bosqichlar. Yuqoridagi ishlarni daraxt hosil bo‘lguncha takrorlash

Prima algoritmining C++ kodi

Quyidagi dastur Primaning algoritmini C ++ da amalga oshiradi. Grafni ko'rsatish uchun qo'shnilik matritsa ishlatilgan bo'lsa-da, ushbu algoritm samaradorligini oshirish uchun qo'shnilik ro'yxati yordamida ham amalga oshirilishi mumkin.

```
#include <iostream>
#include <cstring>
using namespace std;

#define INF 9999999

// grafdagi uchlar soni
#define V 5

//Qo'shnilik matritsasini ifodalash uchun
//5x5 o'lchamdagi ikki o'lchamli massivni yaratish

int G[V][V] = {
  {0, 9, 75, 0, 0},
  {9, 0, 95, 19, 42},
  {75, 95, 0, 51, 66},
  {0, 19, 51, 0, 31},
  {0, 42, 66, 31, 0}
```

```
};

int main () {

  int no_edge;          // qirralar soni

  // tanlangan uchni kuzatish uchun massiv yaratish
  int selected[V];

 // dastlab false qiymatini berish
  memset (selected, false, sizeof (selected));

 // qiralar soniga 0 ni berish
    no_edge = 0;
  selected[0] = true;
  int x;          //  qator raqami
  int y;          //  ustun raqami
  // qirra va og‘irlikni chop etish
  cout << "Qirra" << " : " << "Masofasi";
  cout << endl;
  while (no_edge < V - 1) {
        int min = INF;
     x = 0;
     y = 0;

     for (int i = 0; i < V; i++) {
       if (selected[i]) {
          for (int j = 0; j < V; j++) {
            if (!selected[j] && G[i][j]) {
               if (min > G[i][j]) {
                  min = G[i][j];
                  x = i;
                  y = j;
               }

          }
        }
      }
     }
```

```
    cout << x <<  " - " << y << " :  " << G[x][y];
    cout << endl;
    selected[y] = true;
    no_edge++;
   }

 return 0;
}
```

Mavzu yuzasidan savollar:

1. Eng kichik uzunlikdagi daraxt nima?
2. Prima algoritmining murakkabligini baholang.
3. Kruskal algoritmining murakkabligini baholang.

16-§. Minimal yo‘lni topish masalasi

Amaliyotda uchraydigan ko‘plab masalalarda marshrut uzunligi maksimallashtirilishi yoki minimallashtirilishi talab etiladi. Shunday masalalardan biriga, aniqrog‘i, kommivoyajer masalasiga Gamilton graflari bilan shug‘ullanganda duch kelamiz.

$G = (V, U)$ oriyentirlangan graf berilgan bo‘lsin, bu yerda $V = \{1,2,..., m\}$. G grafning biror $s \in V$ uchidan boshqa $t \in V$ uchiga boruvchi yo‘llar orasida uzunligi eng kichik bo‘lganini topish masalasi bilan shug‘ullanamiz. Bu masalani **minimal uzunlikka ega yo‘l haqidagi masala** deb ataymiz. Quyida bu masalaning umumlashmasi hisoblangan masalani qarab, uni ham o‘sha nom bilan ataymiz.

Grafdagi (i, j) yoyning uzunligini c_{ij} bilan belgilab, $C = (c_{ij}),\ i, j = \overline{1,m}$, matritsa berilgan deb hisoblaymiz. Yuqorida ta’kidlaganlarimizga ko‘ra, C matritsaning c_{ij} elementlari orasida manfiylari yoki nolga tenglari ham bo‘lishlari mumkin. Agar grafda biror i uchdan chiqib j uchga kirruvchi yoy mavjud bo‘lmasa, u holda bu yoyning uzunligini cheksiz katta deb qabul qilamiz ($c_{ij} = \infty$). Bundan tashqari, G grafda umumiy uzunligi

manfiy bo‘lgan sikl mavjud emas deb hisoblaymiz, chunki aks holda uzunligi eng kichik bo‘lgan yo‘l mavjud emas[5].

Minimal uzunlikka ega yo‘l haqidagi masalani hal etish usullari orasida Deykstra[6] tomonidan taklif etilgan algoritm ko‘p qo‘llaniladi. Quyida grafning 1 belgili uchidan chiqib (bu uchni manba deb qabul qilamiz) grafdagi ixtiyoriy k uchgacha (bu uchni oxirgi uch deb hisoblaymiz) eng qisqa uzunlikka ega yo‘lni topish imkonini beruvchi **Deykstra algoritmi** keltirilgan.

Dastlabki qadam. Manbaga (1 belgili uchga) $\varepsilon_1 = 0$ qiymatni mos qo‘yib, bu uchni dastlab $R = \varnothing$ deb qabul qilingan R to‘plamga kiritamiz: $R = \{1\}$. $\overline{R} = V \setminus R$ deb olamiz.

Umumiy qadam. Boshlang‘ich uchi R to‘plamga, oxirgi uchi esa $\overline{R}$ to‘plamga tegishli bo‘lgan barcha yoylar to‘plami $(R, \overline{R})$ bo‘lsin. Har bir $(i, j) \in (R, \overline{R})$ yoy uchun $h_{ij} = \varepsilon_i + c_{ij}$ miqdorni aniqlaymiz, bu yerda ε_i deb $i \in R$ uchga mos qo‘yilgan qiymat (grafning 1 belgili uchidan chiqib i belgili uchigacha eng qisqa yo‘l uzunligi) belgilangan.

$\varepsilon_j = \min\limits_{(i,j)\in(R,\overline{R})} h_{ij}$ qiymatni aniqlaymiz. $(R, \overline{R})$ to‘plamning oxirgi tenglikda minimum qiymat beruvchi barcha elementlarini, ya’ni (i, j) yoylarni ajratamiz. Ajratilgan yoylarning har biridagi $j \in \overline{R}$ belgili uchga ε_j qiymatni mos qo‘yamiz. ε_j qiymat mos qo‘yilgan barcha j uchlarni $\overline{R}$ to‘plamdan chiqarib R to‘plamga kiritamiz.

Ikkala uchi ham R to‘plamga tegishli bo‘lgan barcha (i, j) yoylar uchun $\varepsilon_i + c_{ij} \geq \varepsilon_j$ tengsizlikning bajarilishini tekshiramiz. Tekshirilayotgan tengsizlik o‘rinli bo‘lmagan (ja’ni $\varepsilon_{j_*} > \varepsilon_i + c_{ij_*}$ bo‘lgan) barcha j_* belgili uchlarning har biriga mos qo‘yilgan eski ε_{j_*} qiymat o‘rniga yangi $\varepsilon_i + c_{ij_*}$ qiymatni mos qo‘yamiz va (i, j_*) yoyni ajratamiz. Bunda eski ε_{j_*} qiymat aniqlangan paytda ajratilgan yoyni ajratilmagan deb hisoblaymiz.

---

[5] Agar grafda umumiy uzunligi manfiy bo‘lgan sikl mavjud bo‘lsa, u holda grafning qandaydir s uchidan shu siklning biror i uchiga o‘tib, keyin esa, sikl bo‘ylab harakatlanib, i uchga istalgancha marta qaytish mumkin bo‘lganligidan, istalgancha kichik uzunlikka ega yo‘l tuzish mumkin.

[6] Deykstra Edsger Vayb (Dijkstra Edsger Wybe, 1930-2002) – Gollandiya matematigi.

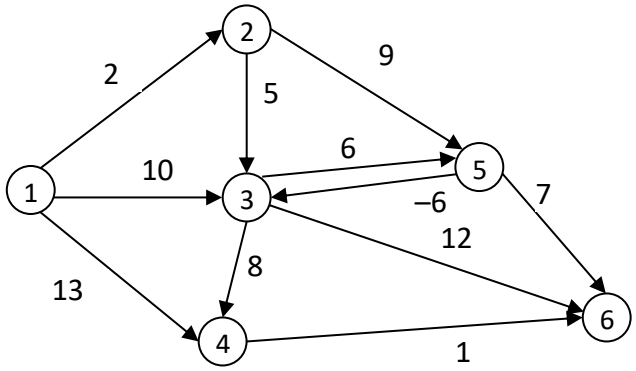
Uchlarga qiymat mos qo‘yish jarayonini oxirgi (k belgili) uchga qiymat mos qo‘yilguncha davom ettiramiz. Grafning 1 belgili uchidan (manbadan) chiqib uning ixtiyoriy k uchigacha (oxirgi uchigacha) eng qisqa yo‘l uzunligi ε_k bo‘ladi.

Oxirgi qadam. Grafning oxirgi uchidan boshlab ajratilgan yoylar yo‘nalishiga qarama-qarshi yo‘nalishda uning 1 belgili uchiga kelguncha harakatlanib, natijada grafdagi 1 belgili uchdan ixtiyoriy k uchgacha eng qisqa uzunlikka ega yo‘l(lar)ni topamiz. ■

1- misol. 2- shaklda tasvirlangan orgrafda oltita uch ($V=\{1, 2, 3, 4, 5, 6\}$) va o‘n bitta yoy bo‘lib, har bir yoy uzunligi uning yoniga yozilgan. Ko‘rinib turibdiki, berilgan grafda manfiy uzunlikka ega $(5,3)$ yoy ham bor. Isbotlash mumkinki, bu grafda umumiy uzunligi manfiy bo‘lgan sikl mavjud emas.

Yuqorida bayon qilingan Deykstra algoritmini berilgan grafga qo‘llab, eng qisqa uzunlikka ega yo‘lni topish bilan shug‘ullanamiz.

Dastlabki qadam. Manbaga (1 belgili uchga) $\varepsilon_1=0$ qiymatni mos qo‘yamiz va $R=\{1\}$ to‘plamga ega bo‘lamiz. Shuning uchun, $\overline{R}=V\setminus R=\{2, 3, 4, 5, 6\}$ bo‘ladi.



Umumiy qadam. 1- iteratsiya. $R=\{1\}$ va $\overline{R}=\{2,3,4,5,6\}$ bo‘lgani uchun boshlang‘ich uchi R to‘plamga tegishli, oxirgi uchi esa $\overline{R}$ to‘plam elementi bo‘lgan barcha yoylar to‘plami $(R,\overline{R})=\{(1, 2),(1, 3),(1, 4)\}$ga ega bo‘lamiz. $(R,\overline{R})$ to‘plamga tegishli bo‘lgan har bir yoy uchun h_{ij}ning qiymatlarini topamiz:

$(1, 2)$ yoy uchun $h_{12}=\varepsilon_1+c_{12}=0+2=2$;

$(1, 3)$ yoy uchun $h_{13}=\varepsilon_1+c_{13}=0+10=10$;

$(1, 4)$ yoy uchun $h_{14}=\varepsilon_1+c_{14}=0+13=13$.

Bu h_{12}, h_{13} va h_{14} miqdorlar orasida eng kichigi h_{12} bo‘lgani uchun $(1, 2)$ yoyni ajratamiz (3- shaklda bu yoy qalin chiziq bilan belgilangan) va 2 belgili uchga $\varepsilon_2 = 2$ qiymatni mos qo‘yamiz. Algoritmga ko‘ra 2 uchni $\overline{R}$ to‘plamdan chiqarib, R to‘plamga kiritamiz. Natijada[7] $R = \{1, 2\}$ va $\overline{R} = \{3, 4, 5, 6\}$ to‘plamlarga ega bo‘lamiz.

Ikkala uchi ham R to‘plamga tegishli bo‘lgan bitta $(1, 2)$ yoy bo‘lgani uchun faqat bitta $\varepsilon_1 + c_{12} \geq \varepsilon_2$ tengsizlikning bajarilishini tekshirish kifoya. Bu tengsizlik $0 + 2 \geq 2$ ko‘rinishdagi to‘g‘ri munosabatdan iborat bo‘lgani uchun 2- iteratsiyaga o‘tamiz.

2- iteratsiya. $(R,\overline{R}) = \{(1, 3),(1, 4),(2, 3),(2, 5)\}$ bo‘lgani sababli $h_{13} = 10$, $h_{14} = 13$, $h_{23} = 7$ va $h_{25} = 11$ qiymatlarni va $\min\{h_{13}, h_{14}, h_{23}, h_{25}\} = h_{23} = 7$ ekanligini aniqlaymiz. Bu yerda eng kichik qiymat $(2, 3)$ yoyga mos keladi. Shuning uchun, $(2, 3)$ yoyni ajratamiz va $\varepsilon_3 = 7$ qiymatni 3 belgili uchga mos qo‘yamiz. 3 belgili uchni $\overline{R}$ to‘plamdan chiqarib, R to‘plamga kiritgandan so‘ng $R = \{1, 2, 3\}$ va $\overline{R} = \{4, 5, 6\}$ to‘plamlar hosil bo‘ladi.

Ikkala uchi ham R to‘plamga tegishli bo‘lgan uchta $(1, 2)$, $(1, 3)$ va $(2, 3)$ yoylardan birinchisi uchun $\varepsilon_1 + c_{12} \geq \varepsilon_2$ tengsizlikning bajarilishi 1- iteratsiyada tekshirilganligi va ε_1, ε_2 qiymatlarning o‘zgarmaganligi sababli faqat ikkinchi va uchinchi yoylarga mos $\varepsilon_1 + c_{13} \geq \varepsilon_3$ va $\varepsilon_2 + c_{23} \geq \varepsilon_3$ munosabatlarni tekshirish kifoya. Bu munosabatlar $0 + 10 \geq 7$ va $2 + 5 \geq 7$ ko‘rinishda bajariladi. Shuning uchun 3- iteratsiyaga o‘tamiz.

3- iteratsiya. Boshlang‘ich uchi $R = \{1, 2, 3\}$ to‘plamga tegishli, oxiri esa $\overline{R} = \{4, 5, 6\}$ to‘plamga tegishli bo‘lgan yoylar to‘rtta: $(1, 4)$, $(2, 5)$, $(3, 4)$ va $(3, 5)$. Shu yoylarga mos h_{ij} ning qiymatlari $h_{14} = 13$, $h_{25} = 11$, $h_{34} = 15$, $h_{35} = 13$ va, shuning uchun, $\min\{h_{14}, h_{25}, h_{34}, h_{35}\} = h_{25} = 11$ bo‘ladi. Demak, bu iteratsiyada $(2, 5)$ yoyni ajratamiz va $\varepsilon_5 = 11$ deb olamiz. Endi, algoritmga ko‘ra, $R = \{1, 2, 3, 5\}$ va $\overline{R} = \{4, 6\}$ to‘plamlarni hosil qilamiz.

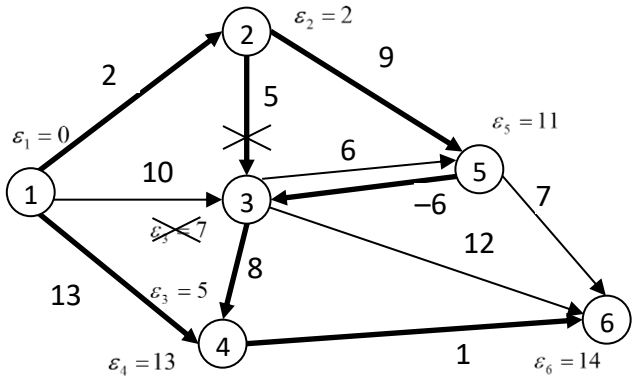
Ikkala uchi ham R to‘plamga tegishli bo‘lgan yoylar oltita: $(1, 2)$, $(2, 3)$, $(1, 3)$, $(2, 5)$, $(3, 5)$ va $(5, 3)$. Bu yoylarning har biri uchun $\varepsilon_i + c_{ij} \geq \varepsilon_j$ tengsizlikning bajarilishini tekshirishimiz kerak. Lekin, 1- va 2-

[7] Yozuvning ixchamligi nuqtai nazardan bu yerda va bundan keyin hosil bo‘lgan to‘plamlar uchun R va $\overline{R}$ belgilar qoldiriladi.

iteratsiyalarda $(1, 2)$, $(2, 3)$ va $(1, 3)$ yoylar uchun bu ish bajarilganligi sababli tekshirishni tarkibida 5 belgili uch qatnashgan $(2, 5)$, $(3, 5)$ va $(5, 3)$ yoylar uchun amalga oshirib, quyidagilarga ega bo‘lamiz: $(2, 5)$ yoy uchun $\varepsilon_2 + c_{25} \geq \varepsilon_5$ munosabat to‘g‘ri $(2+9 \geq 11)$, $(3, 5)$ yoy uchun $\varepsilon_3 + c_{35} \geq \varepsilon_5$ munosabat to‘g‘ri $(7+6 \geq 11)$, lekin $(5, 3)$ yoy uchun $\varepsilon_5 + c_{53} \geq \varepsilon_3$ munosabat noto‘g‘ri $(11+(-6)=5<7)$. Oxirgi munosabatni hisobga olib, algoritmga ko‘ra $\varepsilon_3 = 7$ o‘rniga $\varepsilon_3 = 5$ deb olamiz va $(5, 3)$ yoyni ajratilgan deb, ilgari ajratilgan $(2, 3)$ yoyni esa ajratilmagan deb hisoblaymiz (3- shaklda $\varepsilon_3 = 7$ yozuvning va $(2, 3)$ yoyning qalin chiziq’i ustiga ajratilganlikni inkor qiluvchi $\times$ belgisi qo‘yilgan).

4- iteratsiya. $R = \{1, 2, 3, 5\}$, $\overline{R} = \{4, 6\}$ bo‘lgani uchun $(R, \overline{R}) = \{(1, 4), (3, 4), (3, 6), (5, 6)\}$ va $h_{14} = 13$, $h_{34} = 13$, $h_{36} = 17$, $h_{56} = 18$ hamda $\min\{h_{14}, h_{34}, h_{36}, h_{56}\} = h_{14} = h_{34} = 13$ bo‘ladi. Demak, $(1, 4)$ va $(3, 4)$ yoylarni ajratamiz hamda 4 belgili uchga $\varepsilon_4 = 13$ qiymatni mos qo‘yamiz. Natijada $R = \{1, 2, 3, 5, 4\}$, $\overline{R} = \{6\}$ to‘plamlarga ega bo‘lamiz.

$\varepsilon_i + c_{ij} \geq \varepsilon_j$ munosabatning to‘g‘riligi $(1, 3)$, $(1, 4)$, $(2, 3)$, $(3, 5)$, $(5, 3)$ va



$(3, 4)$ yoylar uchun tekshirilib ko‘rilganda, uning barcha yoylar uchun bajarilishi ma’lum bo‘ladi.

5- iteratsiya. Endi $(R, \overline{R}) = \{(3, 6), (4, 6), (5, 6)\}$ bo‘lgani uchun $h_{36} = 17$, $h_{46} = 14$, $h_{56} = 18$ va $\min\{h_{36}, h_{46}, h_{56}\} = h_{46} = 14$ bo‘ladi. Bu yerda minimum $(4, 6)$ yoyda erishilgani uchun uni ajratib, orgrafning oxirgi 6 belgili uchiga $\varepsilon_6 = 14$ qiymatni mos qo‘yamiz.

Oxirgi qadam. Berilgan orgrafda 1 belgili uchdan 6 belgili uchgacha eng qisqa uzunlikka ega yo‘l(lar)ni topish maqsadida,

algoritmga asosan, grafning oxirgi 6 belgili uchidan boshlab ajratilgan yoylar yo‘nalishiga qarama-qarshi yo‘nalishda harakatlanib, uning 1 belgili uchiga kelishimiz kerak. 6 belgili uchga kiruvchi uchta yoydan faqat bittasi ($(4, 6)$ yoy) ajratilgan bo‘lgani uchun $(4, 6)$ yoy yo‘nalishiga qarama-qarshi yo‘nalishda harakat qilib, 6 belgili uchdan 4 belgili uchga kelamiz. 4 belgili uchga kiruvchi ikkala ($(1, 4)$ va $(3, 4)$) yoylar ham ajratilgan bo‘lgani uchun biz tuzmoqchi bo‘lgan eng qisqa uzunlikka ega yo‘l yagona emas.

Agar harakatni $(1, 4)$ yoy yo‘nalishiga teskari yo‘nalishda davom ettirsak, u holda 4 belgili uchdan 1 belgili uchga kelib, eng qisqa uzunlikka ega yo‘llardan biri bo‘lgan $\mu_1 = (1, 4, 6)$ marshrutni topamiz.

Agarda harakatni $(3, 4)$ yoy yo‘nalishiga teskari yo‘nalishda davom ettirsak, u holda 4 belgili uchdan 3 belgili uchga kelamiz. 3 belgili uchga kiruvchi ikkita yoydan faqat bittasi ($(5, 3)$ yoy) ajratilgan bo‘lgani uchun 3 belgili uchdan 5 belgili uchga kelamiz. Shu usulda davom etsak, oldin 2 belgili, keyin esa 1 belgili uchga o‘tib mumkin bo‘lgan eng qisqa uzunlikka ega bo‘lgan yo‘llardan ikkinchisini, ya’ni $\mu_2 = (1, 2, 5, 3, 4, 6)$ marshrutni aniqlaymiz.

Shunday qilib, 2- shaklda tasvirlangan grafda eng qisqa uzunlikka ega μ_1 va μ_2 yo‘llar borligini aniqladik. Bu yo‘llarning har biri minimal $\varepsilon_6 = 14$ uzunlikka ega.

Mavzu yuzasidan savollar:

1. Minimal yo‘lni topish masalasi yechish uchun qanday algoritmlar ishlab chiqilgan?

2. Deykstra algoritmining murakkabligini baholang.

17-§. Satrlarda qismiy satrlarni qidirish algoritmlari

17.1. Qismiy satrlarni izlashda primitiv algoritmlarning kamchiligi

Satrlardan qismiy satrni qidirish algoritmi – bu matnda (text) qismiy satr (pattern) topishga imkon beradigan satrlar ustidagi algoritmlar sinfi. U matn muharrirlari, MBBT, qidiruv tizimlari, dasturlash tillari va boshqalarda o'rnatilgan funksiya sifatida ishlatiladi.

Qidiruv vazifalarida qidiruv satrni “igna” (inglizchadan - "needle") va qidiruv o'tkaziladigan satrni “g‘aram” (ingliz tilidan - "haystack") deb belgilash odat tusiga kirgan. Shuningdek, biz qidirish olib boriladigan alifboni Σ bilan belgilaymiz.

Primitiv algoritmning muvaffaqiyatsizligi. Agar satrlar birdan boshlab raqamlangan deb hisoblasak, eng oddiy “qo'pol kuch” (Brute force) algoritmi (sodda algoritm) quyidagicha bo‘ladi:

```
for i=0...|haystack|-|needle|
  for j=0...|needle|
    if haystack[i+j + 1]<>needle[j]
      then goto 1
  output("Topildi: ", i+1)
  1:
```

Eng oddiy qidirish algoritmi, eng yaxshisi, |haystack| - |needle| +1 taqqoslash; agar bir-biriga o'xshashliklar ko'p bo'lsa, tezlik O(|haystack|·|needle|) ga tushadi.

Primitiv algoritm o'rtacha 2 soatlik taqqoslashda ishlayotgani isbotlangan.

Bugungi kunda qismiy satrlarni qidirish algoritmlarining xilma-xilligi mavjud. Dasturchi bunday omillarga qarab, mosini tanlashi kerak.

1. Optimallashtirish kerakmi yoki primitiv algoritm yetarlimi? Jimlik bo‘yicha, bu dasturlash tillarining standart kutubxonalari amalga oshiradi.
2. **Foydalanuvchining "dushmanligi".** Boshqacha aytganda: foydalanuvchi ataylab algoritm sekin bajariladigan ma'lumotlarni aniqlaydimi? Eng oddiy holatda O (|*haystack*| · | needle|) ball qo'yadigan juda oddiy algoritmlar mavjud, lekin "muntazam" ma'lumotlarda solishtirishlar soni | *haystack* | dan ancha kam. Faqat 1990-yillarda O (| *haystack* |) ning murakkabligini, eng yomon holatda va | *haystack* | o'rtacha.
3. Tilning grammatikasi qidiruvni "o'rtacha" tezlashtiradigan ba'zi evristikalarga dushman bo'lishi mumkin.
4. **Protsessor arxitekturasi.** Ba'zi protsessorlarda avtomatik kattalashtirish yoki SIMD amallari mavjud bo'lib, ular sizga

ikkita operativ xotirani tez taqqoslashga imkon beradi (masalan, x86-da rep cmpsd). Bunday protsessorlarda “needle”ni “haystack” bilan taqqoslaydigan algoritmni qo'llash juda qiziq - albatta, hamma pozitsiyalarda emas.

5. **Alifbo o'lchami.** Ko'p algoritmlar (ayniqsa, oxirigacha taqqoslashga asoslangan), mos kelmaydigan belgi bilan bog‘liq evristikaga ega. Katta alifbolarda ramzlar jadvali ko'p xotirani egallaydi, kichik alifbolarda tegishli evristik samarasiz bo'ladi.
6. **“haystack”ni indekslash qobiliyati**. Agar mavjud bo'lsa, qidiruv juda tezlashadi.
7. Bir vaqtning o'zida bir nechta satrlarni qidirish kerakmi? Ba'zi algoritmlarning yon xususiyatlari (Axo-Korasik, ikkilik algoritm) bunga imkon beradi.

Qoida tariqasida, matn tahrirlovchisida Boyer-Mur-Xorspul kabi eng oddiy evristik algoritmni olish kifoya-hatto juda sekin kompyuter ham bir soniya ichida qidirishni amalga oshira oladi. Agar matn hajmi gigabaytda o'lchanadigan bo'lsa yoki qidiruv ko'plab so'rovlarni bajaradigan serverda ishlayotgan bo'lsa, siz eng muvaffaqiyatli algoritmni tanlashingiz kerak bo'ladi. Masalan, plagiatni aniqlash dasturlari o'z ma'lumotlar bazasida saqlanadigan ko'plab hujjatlar orasida qismiy satr qidirish algoritmlari yordamida onlayn tekshiruvlarni amalga oshiradi.

17.2. Qismiy satrlarni qidirish algoritmlarining turlari

Rabin-Karp algoritmi. Rabin-Karp algoritmi-bu matnni xeshlash yordamida berilgan satrdan ichki satrni qidiradigan qidirish algoritmi. U 1987-yilda Maykl Rabin va Richard Karp tomonidan ishlab chiqilgan.

Algoritm kamdan-kam hollarda bitta qismiy satrni topish uchun ishlatiladi, lekin muhim nazariy ahamiyatga ega va bir xil uzunlikdagi bir nechta qismiy satr uchun moslikni topishda juda samarali. *n* uzunlikdagi matn va *m* uzunlikdagi qismiy satr uchun uning o'rtacha va eng yaxshi bajarilish vaqti to'g‘ri xesh funksiyasi bilan O (n) dir, lekin eng yomon holatda uning samaradorligi O (nm) ga teng. Bu esa keng qo'llanilmasligining sabablaridan biridir.

Rabin-Karp algoritmining eng oddiy amaliy qo'llanmalaridan biri plagiatni aniqlashdir. Rabin-Karp algoritmi tekshirilgan maqoladagi manba materiallardan ba'zi jumlalar paydo bo'lishining misollarini tezda topishi mumkin. Algoritmning kichik farqlarga sezgirligini yo'q qilish uchun siz ularni olib tashlash orqali harf yoki tinish belgi kabi tafsilotlarni e'tiborsiz qoldirishingiz mumkin. Biz qidirayotgan qatorlar soni juda katta bo'lgani uchun, bitta satrlarni topishning an'anaviy algoritmlari samarasiz bo'lib qoladi.

Quyidagi misol orqali Rabin-Karp algoritmini ko‘rib chiqamiz.

Berilgan matn S= “aevesapng”

Izlanadigan satr P= “esap”

0	1	2	3	4	5	6	7	8
a	e	v	e	s	a	p	n	g

0	1	2	3
e	s	a	p

Quyida simvollar uchun xesh-kod keltirilgan:

a	→	1
b	→	2
c	→	3
d	→	4
e	→	5
f	→	6
…	→	…
z	→	26

1-qadam. Belgilarga tayinlangan xesh kodi yordamida qismiy satrning xesh kodi qiymatini topamiz.

0	1	2	3
e	s	a	p
↓	↓	↓	↓
5	19	1	16

Xesh-kod qiymati: 5+19+1+16=41

2-qadam. Agar *m* qismiy satr uzunligi bo'lsa, biz matn satrining boshidan m uzunlikdagi qismiy satrni olishni boshlaymiz. Shundan so'ng, qismiy satr uchun xesh-kod qiymatini topamiz va shablon satrining xesh-kod qiymatiga mos kelishini tekshiramiz. Agar u mos keladigan bo'lsa, belgini birma-bir tekshiradi, aks holda keyingi qismiy satrga o'tadi.

0	1	2	3	4	5	6	7	8
a	e	v	e	s	a	p	n	g
↓	↓	↓	↓					
1	5	22	5					

Xesh kod qiymati: 1+5+22+5 = 33

Xesh-kod qiymati mos kelmaydi, keyin biz *m* (4) uzunlikdagi keyingi satrga o'tamiz.

3-qadam.

0	1	2	3	4	5	6	7	8
a	e	v	e	s	a	p	n	g
	↓	↓	↓	↓				
	5	22	5	19				

Xesh kod qiymati: 5+22+5+19 = 51

Xesh-kod qiymati mos kelmaydi, keyin *m* uzunlikdagi keyingi satrga o'tamiz.

4-qadam.

0	1	2	3	4	5	6	7	8
a	e	v	e	s	a	p	n	g
		↓	↓	↓	↓			
		22	5	19	1			

Xesh kod qiymati: 22+5+19+1 = 47

Xesh-kod qiymati mos kelmaydi, keyin biz *m* uzunlikdagi keyingi satrga o'tamiz.

5-qadam.

0	1	2	3	4	5	6	7	8
a	e	v	e	s	a	p	n	g
			↓	↓	↓	↓		
			5	19	1	16		

Xesh kod qiymati: 5+19+1+16 = 41

Bu yerda xesh-kodining qiymati bir xil, shuning uchun biz ichki qismiy belgilarini qidiriluvchi satr bilan birma -bir tekshiramiz.

0	1	2	3	4	5	6	7	8
a	e	v	e	s	a	p	n	g
			↓	↓	↓	↓		
			e	s	a	p		

Barcha belgilar mos keladi, keyin biz ichki satrning boshlang‘ich indeksini chop etamiz va iloji bo'lsa, *m* uzunlikdagi keyingi qismiy satrga o'tamiz.

6-qadam.

0	1	2	3	4	5	6	7	8
a	e	v	e	s	a	p	n	g
				↓	↓	↓	↓	
				19	1	16	14	

Xesh kod qiymati: 19+1+16+14 = 50

Joriy ichki satrning xesh qiymati qismiy satrning xesh qiymatiga mos kelmaydi. Shunday qilib, iloji bo'lsa, m uzunligining keyingi ichki satriga o‘tamiz, aks holda to'xtatamiz.

7-qadam.

0	1	2	3	4	5	6	7	8
a	e	v	e	s	a	p	n	g
					↓	↓	↓	↓
					1	16	14	7

Xesh kod qiymati: 1+16+14 +7= 38

Bu yerda ham xesh kodi qiymati mos kelmaydi va bu *m* uzunligining oxirgi ichki satri. Shunday qilib, bu yerda o'z jarayonimizni to'xtatamiz.

Eslatma: Bu yerda xesh funksiyasini yaratish yoki aniqlashning turli usullari mavjud. Yaxshi tushunish uchun oddiy xesh funksiyasidan foydalanildi.

Rabin-Karp algoritmi (C++)

```
#include<bits/stdc++.h>
using namespace std;
void rabin_karp(string &text,string &pattern, int q)
{
/*qismiy satr uzunligi*/
int m = pattern.length();
/*Berilgan satr uzunligi*/
int n = text.length();
int p=0,t=0,h=1,d=26; // bu yerda p - matn uchun xesh qiymati, t – qismiy satrning xesh qiymati;
/*h=pow(d,M-1) bu yerda d - 26, agar matnda faqat katta harflar bo'lsa.*/
for(int i=0;i<m-1;i++)
{
h=(h*d)%q;
}
/* matn va m uzunligining birinchi ichki satri uchun xesh qiymatini hisoblash */
for(int i=0;i<m;i++)
{
p=(d*p+pattern[i])%q;;
t=(d*t+text[i])%q;
}
/* m uzunlikdagi qolgan satr uchun */
for(int i=0;i<=n-m;i++)
{
```

```
/* agar xesh qiymatlari bir xil bo'lsa, xeshni ichki satr va qismiy satridagi belgilar bo'yicha tekshirish */
if(p==t)
{
int flag=0;
for(int j=0;j<m;j++)
{
if(text[i+j]!=pattern[j])
{
flag=1;
break;
}
}
/* agar barcha belgilar mos keladigan bo'lsa, ichki satrning boshlang‘ich indeksini chop etish.*/
if(flag==0)
{
cout<<" Indeksda berilgan satr osti topildi:"<<i+1<<endl;
}
}
/*oldingi ichki satrdan birinchi belgini olib tashlash orqali keyingi ichki satrning xesh qiymatini toping va keyingi satrni oldingi satr oxiriga qo'shing*/
/*xesh qiymatlarini topish uchun O (1) vaqt kerak bo'ladi*/
if(i<n-m)
{
t=(d*(t-text[i]*h)+text[i+m])%q;
if(t<0)
{
t=(t+q);
}
}
}
}
int main()
{
/*o‘zgaruvchilarni kiritish*/
string text;
cin>>text;
string pattern;
```

```
cin>>pattern;
rabin_karp(text,pattern,97);
return 0;
}
```

Boyer-Mur algoritmi. 1977-yilda Robert Boyer va Jey Mur tomonidan ishlab chiqilgan, matnda oldindan ishlov berish imkoniyati bo'lmagan taqdirda, satrda qismiy satrni topish algoritmlari orasida eng tezkori hisoblanadi**.**

Algoritm g‘oyasi quyidagicha:

- Chapdan o'ngga skanerlash, o'ngdan chapga taqqoslash.
- To'xtash belgisini topish
  - agar taqqoslanadigan birinchi harf mos kelmasa, shablon eng yaqiniga o'tkaziladi
  - to'xtash belgisi bo'lmasa, shablon uning orqasiga siljiydi
- Mos keladigan qo'shimchani topish
  - agar 1 yoki undan ortiq belgi mos kelsa, shablon bu qo'shimchaning birinchi mos kelishiga qadar o'ngga siljiydi

1. **q w t e e q e w q r w q w r q r**
   q w r q r
2. **q w t e e q e r q r w q w r q r**
   q w r q r
3. **q w t e e q e w q r w q w r q r**
   q w r q r
4. **q w t e e q e w q r w q w r q r**
   q w r q r

To'xtatish belgisi jadvali. Qismiy satrdagi elementning oxirgi o'rnini belgilaydi (oxirgi harfdan tashqari). Agar qismiy satrda element bo'lmasa, jadvalga 0 kiritiladi (bittadan raqamlash uchun)

Misol. Qismiy satr: qwrqr

Simvol	q	w	r	e	t
Oxirgi pozitsiya	4	2	3	0	0

1. **q t e e q r w q w r e e**
   q w r q r
2. **q t e e q r w q w r e e**
   q w r q r

Suffiks jadvali. Mumkin bo'lgan barcha qo'shimchalar uchun jadvalda qismiy satrni o'zgartirish kerak bo'lgan eng kichik miqdor ko'rsatilgan, u yana qo'shimchaga mos keladi. Agar bunday siljishning iloji bo'lmasa, satrning uzunligi ko'rsatilgan.
Misol. Qismiy satr: qwrqr

Suffiks	Bo‘sh	r	qr	rqr	wrqr	qwrqr
qadam	1	2	5	5	5	5

1. **q t e e q r w q w r e e**
   q w r q r
2. **q t e e q r w q w r e e**
   q w r q r
3. **q t e e q r w q w r e e**
   q w r q r
4. **q t e e q r w q w r e e**
   q w r q r

Algoritmning murakkabligi. O (| haystack | + | needle | + | Σ |) davriy bo'lmagan qismiy satrlar bo'yicha
O (| haystack | · | needle | + | Σ |) davriy
haystack - berilgan satr, needle – qismiy satr, Σ - solishtirish uchun alifbo

1991-yilda Koul shuni ko'rsatdiki, davriy bo'lmagan sxemalar bo'yicha, algoritm satr bo'ylab to'liq o'tishda 3·|haystack| tadan ko'p bo'lmagan taqqoslashni amalga oshiradi.

Boyer-Mura algoritmi (C++)

```
#include <bits/stdc++.h>
using namespace std;
# define NO_OF_CHARS 256


void badCharHeuristic( string str, int size, int badchar[NO_OF_CHARS])
{
	int i;
	for (i = 0; i < NO_OF_CHARS; i++)
		badchar[i] = -1;
	for (i = 0; i < size; i++)
		badchar[(int) str[i]] = i;
}
void search( string txt, string pat)
{
	int m = pat.size();
	int n = txt.size();
	int badchar[NO_OF_CHARS];
	badCharHeuristic(pat, m, badchar);
	int s = 0;
	while(s <= (n - m))
	{
		int j = m - 1;
		while(j >= 0 && pat[j] == txt[s + j])
			j--;
		if (j < 0)
		{
			cout  << s << endl;
			s += (s + m < n)? m-badchar[txt[s + m]] : 1;
		}
		else
			s += max(1, j - badchar[txt[s + j]]);
	}
```

```
}
int main()
{
        string txt= "ABAAABCD";
        string pat = "ABC";
        search(txt, pat);
        return 0;
}
```

Mavzu yuzasidan savollar:

1. Qismiy satrlarni izlash algoritmi nima?
2. Qismiy satrlarni izlash algoritmlarini foydalanish samaradorligini taqqoslang
3. Suffiks jadvali nima?
4. Satrlar uchun xesh funksiyasini qo‘llang
5. Robin-Karp va Boyer-Mur algoritmlarini taqqoslang?

Mustaqil ishlash uchun masalalar:

1. C++ tilida xesh jadvallarni hosil qiling.
2. C++da xesh jadvallarning metodlarini qo‘llang

GLOSSARY

Abstrakt ma’lumotlar turi (ADT – Abstract Data Type) - bu ma’lumotlar turlari uchun matematik model, bu yerda ma’lumotlar turi xatti-harakatlari (semantikasi) bilan foydalanuvchi nuqtai nazaridan aniqlanadi

Algoritm - bu belgilaydigan cheklangan qoidalar to‘plami, muayyan vazifalar to‘plamini hal qilish bo‘yicha amallar ketma-ketligi va beshta muhim xossaga ega: aniqlik, tushunarlilik, kiritish, chiqarish, samaradorlik

Algoritm qadami – Algoritmda bajarilishi tugallangan amallar ketma-ketligi.

Algoritmning vaqt bo‘yicha qiyinligi – Algoritm sarflanayotgan vaqt masalaning o‘lchami funksiyasi

B daraxti (inglizcha *B-tree*) – izlash, qo'shish va o'chirish imkonini beradigan, juda ko‘pshoxli muvozanatlashgan qidiruv daraxti

Barg yoki terminal tuguni – avlodi mavjud bo‘lmagan tugun

Bog‘langan graf - bu har qanday uch juftligi o‘rtasida kamida bitta yo‘l mavjud bo‘lgan graf

Chiziqli algoritmlar – Algoritmning turlari bilan tanishtirganda, avvalo hech qanday shart tekshirilmaydigan va tartib bilan faqat ketma-ket bajariladigan jarayonlarni ifodalaydigan algoritmlar

Daraxt - bu bog‘langan asiklik grafik, ya’ni sikllar yo‘q va tepalik juftligi orasida bitta yo‘l bor

Daraxt osti - bu alohida daraxt sifatida namoyish etilishi mumkin bo‘lgan daraxtga o‘xshash ma’lumotlar strukturasining bir qismi

Eng kichik uzunlikdagi daraxt – berilgan grafning eng kam darajaga ega bo‘lgan daraxti

Grafdagi marshrut - bu har bir uch (oxirgisidan tashqari) ketma-ketlikdagi keyingi uchga qirra bilan bog‘langan uchlarning cheklangan ketma-ketligi.

Graflar - bu chiziqlar bilan bog‘langan nuqtalar to‘plami

Heapsort (Heapsort, "Heap sorting") - n elementlarni saralashda O(nlogn) amallarda eng yomon, o'rtacha va eng yaxshi (ya'ni kafolatlangan) holda ishlaydigan saralash algoritmi

Hisoblash geometriyasi - geometrik masalalarni yechish algoritmlari bilan shug‘ullanadigan informatika bo'limi

Ichki saralash - bu tezkor xotiradagi ma’lumotlarni saralash

Ichki tugun - bu daraxtga avlodi mavjud bo‘lgan har qanday tugun va shuning uchun barg tuguni emas

Ikkilik daraxt - bu har bir tugunda ko‘pi bilan ikkita avlod (bola) bo‘lgan ma’lumotlarning iyerarxik tuzilishi

Ildiz – ixtiyoriy tanlab olingan uchlardan biri

Ildiz tuguni - daraxtning eng yuqori tuguni

Insidentlik matritsasi - bu grafning elementlari (qirra - uch) orasidagi bog‘lanishlar ko‘rsatiladigan grafni tasvirlash shakli

Ma’lumotlar strukturasi (ing. data structure) - bu hisoblashda turli xil bir tipli va (yoki) mantiqiy bog‘liq ma’lumotlarni saqlash va qayta ishlashga imkon beradigan dastur birligi

Navbat - bu FIFO (First In - First Out - "birinchi kelgan – birinchi ketadi") prinsipi bo‘yicha qurilgan ma’lumotlar strukturasi

Oddiy zanjir - bu uchlarni takrorlamaydigan marshrut.

O‘rmon – juda ko‘p daraxtlar

Qo‘shnilik ro‘yxati - bu grafni uchlar ro‘yxati ("ro‘yxatlar ro‘yxati") to‘plami sifatida ko‘rsatish usuli - grafning har bir uchi qo‘shni uchlar ro‘yxatiga to‘g‘ri keladi

Ro‘yxat - bu ikki tomonlama bog‘langan ro‘yxatlarga asoslangan ma’lumotlar strukturasi

Saralash algoritmi – bu ro‘yxatdagi elementlarni saralash algoritmi

Satrlardan qismiy satrni qidirish algoritmi – bu matnda (text) qismiy satr (pattern) topishga imkon beradigan satrlar ustidagi algoritmlar sinfi

Sentroid - uch, u olib tashlanganida hosil bo‘lgan ulanish komponentlarining o‘lchamlari $\frac{n}{2}$ dan oshmaydi

Sikl (oddiy sikl) - bu yopiq zanjir (oddiy zanjir).

Stek - bu LIFO (last in – first out; oxirgi kelgan – birinchi ketadi) prinsipi bo‘yicha ishlaydigan ma’lumotlar strukturasi

Tartiblash – bu berilgan obyektlar to‘plamini muayyan tartibda qayta tartibga solish jarayoni

Tashqi saralash - tashqi xotira (fayllar)dagi ma’lumotlarni saralash

Tugun - bu ba’zi bir qat’iy tabiat obyektiga mos keladigan ikki turdagi graf elementlaridan birining nusxasi

Tugunning balandligi - bu tugundan eng pastki tugunga (chekka tugunga) barg deb ataladigan pastga tushadigan yo‘lning maksimal uzunligi

Uchning darajasi - unga tushgan qirralarning soni

Ustivor navbat - bu yozuvlar bir-biri bilan chiziqli taqqoslanadigan kalitlarga (masalan, raqamlar) ega bo'lgan va ikkita amalni realizatsiya qiladigan axborot tizimidir

Vektor - bu dinamik massiv modeli bo‘lgan ma’lumotlar strukturasi

Xesh funksiyalar – ixtiyoriy uzunlikdagi kirish ma’lumotini chiqishda belgilangan uzunlikdagi xesh qiymatga aylantirib beruvchi bir tomonlama funksiyalar

Xesh jadvali - bu assotsiativ massiv interfeysini amalga oshiruvchi ma'lumotlar tuzilmasi, ya'ni juftlarni saqlashga (kalit, qiymat) va uchta

amalni bajarishga imkon beradi: yangi juftlikni qo'shish, qidirish amali va juftlikni kalit bilan o'chirish

Xeshlash – bu ixtiyoriy uzunlikdagi kirish ma'lumotlari majmuasini ma'lum bir algoritm tomonidan bajarilgan, belgilangan o'lchamdagi chiqish massiviga aylantirish jarayoni

Yo‘l - bu qirralarning takrorlanmagan yo‘lidir.

Yo‘lning (yoki siklning) uzunligi - uni tashkil etuvchi qirralarning soni

Yo‘naltirilgan graf - (qisqacha orgraf) - qirralari yo‘naltirilgan graf

Yo‘naltirilmagan graf - uchlar juftligi tartiblanmagan graf

Yordamchi algoritm – asosiy algoritmni aniqlashtiruvchi algoritm

XULOSA

Ushbu o‘quv qo‘llanma 5330100 – Kompyuter ilmlari va dasturlash texnologiyalari va 5130200 – Amaliy matematika yo‘nalishi talabalari uchun uchun mo‘ljallangan bo‘lib, “Algoritmlar va ma’lumotlar strukturasi” fanining mavzularini o‘z ichiga ma’lumotlar bilan tanishib chiqish mumkin.

Eng qadimiy raqamli algoritmlardan biri Yevklid algoritmi (miloddan avvalgi III asr) deb hisoblanadi - ikki sonning eng katta umumiy bo‘luvchisini topish. Algoritmlarning zamonaviy nazariyasi nemis matematikasi Kurt Gyodel (1931) asarlari bilan boshlandi, ular o‘zlarining rasmiy, izchil aksiomalar tizimi doirasida yechib bo‘lmaydigan muammolar mavjudligini ko‘rsatdi.

Algoritmlar nazariyasida hal qilingan maqsad va vazifalar:

- "algoritm" tushunchasini formallashtirish (rasmiylashtirish) va formal (rasmiy) algoritmik tizimlarni o‘rganish;
- muammolarning algoritmik yechimini rasmiy tasdiqlash;
- vazifalarni tasniflash, murakkablik sinflarini aniqlash va tadqiq qilish;
- algoritmlarning murakkabligini asimptotik tahlil qilish;
- rekursiv algoritmlarni o‘rganish va tahlil qilish;
- algoritmlar sifatini qiyosiy baholash mezonlarini ishlab chiqish.

Hisoblash nazariyasi va hisoblash murakkabligi nazariyasi hisoblash modelini nafaqat hisoblash uchun foydalaniladigan qabul qilinadigan amallar to‘plamining ta’rifi, balki ularni qo‘llashning nisbiy xarajatlari sifatida ham ko‘rib chiqadi. Kerakli hisoblash manbalarini - ijro etish vaqtini, xotira hajmini, shuningdek algoritmlarning cheklanishlarini yoki kompyuterni xarakterlash mumkin - faqat ma’lum bir hisoblash modeli tanlangan taqdirda.

Modelga asoslangan muhandislikda hisoblash modeli va uning tanlovi, agar uning alohida qismlarining xatti-harakatlari ma’lum bo‘lsa, umuman tizim qanday ishlaydi degan savolga javob beradi.

Hisoblash murakkabligining asimptotik bahosida hisoblash modeli ma’lum narx bilan qabul qilinadigan primitiv amallar orqali aniqlanadi.

Xulosa sifatida aytish mumkinki, yuqorida keltirilgan vazifalar ushbu qo‘llanmada keltirilgan fundamental algoritmlar yordamida hal qilinadi. Bu algoritmlarni optimallashtirish yoki uning boshqa turlarini o‘rganish asosida talabalar bilimlarini yanada mustahkamlab borishlari mumkin bo‘ladi.

Foydalanilgan adabiyotlar ro‘yxati

1. Клейнберг Дж., Тардос Е. К48 Алгоритмы: разработка и применение. Классика Computers Science / Пер. с англ. Е. Матвеева. — СПб.: Питер, 2016. — 800 с.: ил. — (Серия «Классика computer science»).
2. To‘rayev H.T. Matematik mantiq va diskret matematika.: Oliy ta’lim muassasalarining talabalari uchun darslik: 11 jildlik. H.T. To‘rayev, 1. Azizov; H.T. To'rayevning umumiy tahriri ostida; O‘zR Oliy va o‘rta-maxsus ta’lim vazirligi. - Toshkent: Tafakkur-Bo‘stoni, 2011. - 288 bet
3. Ахо, Альфред, В., Хопкрофт, Джон, Ульман, Джеффри, Д. Структуры данных и алгоритмы.: Пер. с англ.: Уч. Пос. – М. Издательский дом «Вильямс», 2000. – 384 с.: ил. – Парал. Титю англ.
4. Пышкин Е.В. Структуры данных и алгоритмы: реализация на C/C++. - СПб.: ФТК СПБГПУ, 2009.- 200 с., ил.
5. Овсянников, А. В. Алгоритмы и структуры данных : учебно-методический комплекс для специальности 1-31 03 07 «Прикладная информатика (по направлениям)». Ч. 1 / А. В. Овсянников, Ю. А. Пикман ; БГУ, Фак. социокультурных коммуникаций, каф. информационных технологий. – Минск : БГУ, 2015. – 124 с. : ил. – Библиогр.: с. 122
6. Домнин Л. Н. Элементы теории графов: учеб. Пособие / Л. Н. Домнин. – Пенза: Изд-во Пенз. Гос. Ун-та, 2007. – 144 с. 75 ил., 13 табл., библиогр 18 назв.
7. Никлаус Вирт, Алгоритмы и структуры данных. Новая версия для Оберона + CD / Пер. с англ. Ткачев Ф. В. – М.: ДМК Пресс, 2010. – 272 с.: ил.

O. R. Yusupov, I. Ximmatov, E. Sh. Eshonqulov

ALGORITMLAR VA MA’LUMOTLAR STRUKTURALARI

(Kompyuter ilmlari va dasturlash texnologiyalari va Amaliy matematika yo‘nalishi talabalari uchun)

Muharrir: O. Sharapova

Musahhih: N. Isroilov

Texnik moharrir: O. Shukurov

ISBN 978-9943-

2021-yil _____ tahririy-nashriyot bo‘limiga qabul qilindi.

2021-yil _____da original-maketdan bosishga ruxsat etildi.

Qog‘oz bichimi 60x84.1/16. “Times New Roman” garniturasi.

Offset qog‘ozi. Shartli bosma tabog‘i – __.

Adadi __ nusxa. Buyurtma № __

SamDU tahririy-nashriyot bo‘limida chop etildi.

140104, Samarqand sh., Universitet xiyoboni, 15.