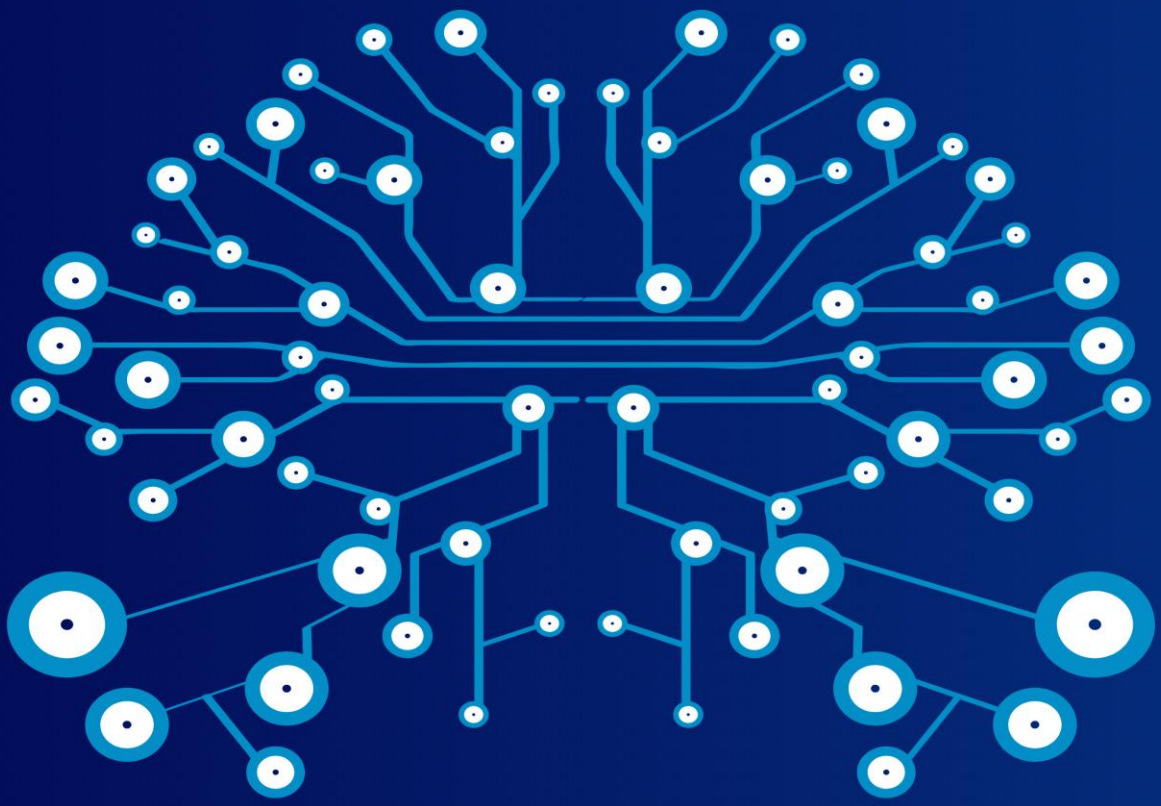
ABLAQULOV KAMOLIDDIN BAHRIDDINOVICH

MA’LUMOTLAR TUZILMASI VA ALGORITMLARI



60610500 - Kompyuter injiniringi bakalavriat yo‘nalishi uchun

O’QUV QO’LLANMA

O‘ZBEKISTON RESPUBLIKASI

OLIY TA’LIM, FAN VA INNOVATSIYALAR VAZIRLIGI

QARSHI DAVLAT TEXNIKA UNIVERSITETI

ABLAQULOV KAMOLIDDIN BAHRIDDINOVICH

MA’LUMOTLAR TUZILMASI VA ALGORITMLARI

60610500 - Kompyuter injiniringi bakalavriat yo‘nalishi uchun

O‘QUV QO‘LLANMA

Qarshi
“BIG MAKRO WORLD” nashriyoti
2025

UO‘T 519.95(075)
KBT 22.18
A 17
Ablaqulov Kamoliddin Bahriddinovich
Ma’lumotlar tuzilmasi va algoritmlari / O‘quv qo‘llanma / – Qarshi. “BIG MAKRO WORLD” nashriyoti, – 2025. – 260 bet.

“Ma’lumotlar tuzilmasi va algoritmlari” o‘quv qo‘llanmasi (“Kompyuter injiniringi”, Dasturiy injiniring, “AT-Servis”, “Axborot xavfsizligi”, “Multimedia texnologiyalari”, Telekommunikatsiya texnologiyalari) ta’lim yo‘nalishi bo‘yicha ta’lim olayotgan bakalavriat yo‘nalishidagi talabalarga mo‘ljallangan.

Mazkur o‘quv qo‘llanmada ma’lumotlar turlari va algoritmlari; chiziqli ma’lumotlar tuzilmalari; stek, navbat va dek; daraxtsimon ma’lumotlar tuzilmalari; graflar bilan ishlash algoritmlari mavzulari batafsil yoritilgan.

O‘quv qo‘llanma dasturlashda qo’llaniladigan ma’lumotlar tuzilmalari, ularning spesifikatsiyasi va amalga oshirilishi bo‘yicha bilimlarning nazariy asoslarini, ma’lumotlarni qayta ishlash algoritmlari va bu algoritmlarni tahlil qilish, algoritmlar va ma’lumotlar strukturalarining o‘zaro bog‘liqligini o‘rgatish hamda ularni amaliyotga tadbiq etish ko‘nikmasini hosil qilishga ko‘maklashadi.

O‘quv qo‘llanmadan soha mutaxassislari, oliy ta’lim talabalari, algoritm loyihalash fan o‘qituvchilari, oliy ta’lim qoshidagi texnikum, dasturlash to‘garaklarining rahbarlari o’z faoliyatlarida foydalanishlari mumkin.

Taqrizchilar:

T.N. Jo’rayev – Qarshi davlat universiteti, “Algoritmlar va dasturlash texnologiyalari” kafedrasi dotsenti, p.f.f.d. (PhD).

Sh.R. Davronov – Muhammad al-Xorazmiy nomidagi Toshkent axborot texnologiyalari universiteti Qarshi filiali “Axborot texnologiyalarining dasturiy ta’minoti” kafedrasi dotsenti, t.f.f.d. (PhD).

Ushbu o‘quv qo‘llanma Oliy ta’lim, fan va innovatsiyalar vazirligining 2024-yil 25-iyundagi 218-sonli buyrug‘iga asosan nashr etishga ruxsat berildi. Ro‘yxatga olish raqami 218-089.

ISBN 978-9910-698-90-3

© K.B. Ablaqulov, 2025
© “BIG MAKRO WORLD” nashriyoti, 2025

KIRISH

Hozirgi kunda O‘zbekistonning iqtisodiy jihatdan rivojlanib borishi va globallashuv sharoitida dunyoda o‘z o‘rniga ega bo‘lishida faoliyatning barcha jabhalarida axborot-kommunikatsiya texnologiyalaridan foydalanish, dasturlashda qo‘llaniladigan ma’lumotlar tuzilmalari va algoritmlarini bilish va ulardan barcha tarmoqlarda keng ko‘lamda foydalanishni talab qilmoqda.

“Ma’lumotlar tuzilmasi va algoritmlari” fanidan qilingan o‘quv qo‘llanma oliy ta’lim talabalari uchun mo‘ljallangan bo‘lib, mazkur fan bo‘yicha oliy ta’lim o‘quv rejasiga binoan, kompyuter injiniringi, dasturiy injiniringi, axborot xavfsizligi, axborot-kommunikatsiya texnologiyalari sohasida kasb ta’limi, raqamli iqtisodiyot, pochta aloqa texnologiyasi, telekommunikatsiya texnologiyalari yo‘nalishlarida “Ma’lumotlar tuzilmasi va algoritmlari” fanining o‘quv dasturidagi materiallarini o‘z ichiga oladi.

Muallifning to‘plagan ish tajribalari asosida yaratilgan ushbu o‘quv qo‘llanma besh bobdan iborat bo‘lib, unda quyidagi mavzular yoritilgan:

— Ma’lumotlar turlari va algoritmlari;
— Chiziqli ma’lumotlar tuzilmalari;
— Stek, navbat va dek;
— Daraxtsimon ma‘lumotlar tuzilmalari;
— Graflar bilan ishlash algoritmlari;

Materiallar bayonida zarur deb hisoblangan o‘rinlarda ma’lumotlar turlari va dasturlash tilidan foydalanilgan.

“Ma’lumotlar tuzilmasi va algoritmlari” fani bo‘yicha amalda bo‘lgan darsliklarda, dasturda qo‘llaniladigan ma’lumotlar tuzilmalari, ularning amalga oshirilishi bo‘yicha bilimlarning nazariy asoslari kabi masalalarga yetarli e’tibor berilmaganini hisobga olib, ushbu o‘quv qo‘llanmada bu mavzular va unga tegishli masalalar tushunarli tarzda bayon qilingan.

1-BOB. MA’LUMOTLAR TURLARI VA ALGORITMLARI

1-§. Ma’lumotlar turlari va algoritmlari

1.1. Ma’lumotlarning abstrakt tuzilmalari. Algoritmlarni ishlab chiqish va tahlil qilish. Ma’lumotlar va ularni ifodalash bosqichlari. Ma’lumotlar tuzilmasining klassifikatsiyasi

Reja:

1. Algoritmlarni ishlab chiqish va tahlil qilish.
2. Ma’lumotlar va ularni ifodalash bosqichlari.
3. Ma’lumotlar tuzilmasini klassifikatsiya qilish.
4. Ma’lumotlarni asosiy abstrakt turlari.

Tayanch iboralar: Ma’lumotlar tuzilmasi, algoritm, ma’lumotlar toifalari, abstrakt bosqich, fizik bosqich, mantiqiy bosqich, ketma-ketlik, diskritlik.

Algoritmlarni ishlab chiqish va tahlil qilish

Kompyuter o‘z hisoblash usuli bilan birga tezkor, aniq va shu bilan birga butunlay o‘zgacha amallarni bajaruvchi hisoblanadi. Turli masalalarni yechishda undan foydalanganda kompyuter shunga mos qismlarni o‘zi o‘ylab topadi, – degan fikr xato, kompyuter ishlashi uchun unga aniq va to‘liq masalalarni tushuntirishi kerak. Bu bilan qo‘yilgan msala uchun algoritmni to‘g‘ri taqsimlash usuliga e’tibor qaratiladi.

Algoritm – so‘nggi natijani hosil qilish uchun kerakli bo‘lgan, biror harakatni amalga oshiruvchi ketma-ket tartiblangan qat’iy o‘rnatilgan tizim. Bu g‘alati bo‘lishi mumkin, lekin real hayotda bunday holatda har doim duch kelinadi. Omadli telefon qo‘ng‘irog‘i uchun kerakli bo‘lgan amallar tartibini o‘z ichiga oluvchi telefon-avtomatdan foydalanish instruksiyasi ham shundan andoza oladi. Maishiy texnikadan foydalanish qoidalari va boshqalar tushunarli shaklda u yoki bu holda nima qilish kerakligidan qat’iy nazar harakatlar algoritmini belgilab ko‘rsatadi. Tarixdan ma’lumki, matematiklarning ta’kidlashicha, “algoritm” so‘zi buyuk ajdodimiz Abu Abdulloh Muhammad ibn

Muso al-Xorazmiy ismidan kelib chiqqan, uning mashhur “Kitob al-jabr va al-muqobola” asari esa yana bir mashhur “algebra” atamasining vujudga kelishiga asos bo‘ldi. Kompyuter ish jarayonida instruksiyalarni ketma-ket boshqarishda asosiy algoritm hisoblanadi. Biroq, algoritmdan o‘z yozuvlarni to‘g‘ridan-to‘g‘ri kompyuterga o‘tkaza olmaydi, sababi ularni kompyuter tushunadigan tilda emas, ular faqatgina insonlar tushunadigan tilda yozilgan bo‘ladi. Kompyuter algoritmni tushunishi uchun ularni mashina tiliga o‘giradi, aynan shunday mashina tilida yozilgan algoritmlar **dastur** yoki **kompyuter tushunadigan til** deb yuritiladi. Quyida bu tushunchani biror-bir mavzu asosida algoritm tushunchasi yordamida aniqlashtirishga harakat qilinadi.

Shuni ta’kidlash kerakki, adabiyotda barcha tushunadigan sodda ko‘rinishdagi algoritmni aniqlash tushunchasi berib o‘tilmagan. Shunday ekan, kompyuter texnologiyalari tushunchasiga mos bo‘lgan algoritm ifodalari berib o‘tildi:

• **Algoritm –** bu masala yechimini hosil qilish uchun va boshlang‘ich informatsiyalarni amalga oshirishda kerak bo‘lgan aniq belgilangan amallar ketma-ketligidir.

Algoritm tushunchasi XX asr boshlarida D. Gilbert, K. Gyodel, S. Klin, A. Chyorch, E. Post, A. Tyuring, N. Viner, A. A. Markov singari olimlarning asarlari tufayli yuzaga keladi. Algoritm tushunchsiga jahon olimlari tomonidan turli ta’riflar berilgan bo‘lib, ulardan ayrimlari keltirib o‘tildi:

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

• **Algoritm –** bu qat’iy belgilangan qoidalar asosida bajariladigan har qanday hisoblash tizimi bo‘lib, bu ma’lum bir qator bosqichlardan so‘ng aniq qo‘yilgan masalani hal qilishga ko‘maklashadi (A. Kolmogorov).

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

Ushbu ta’riflardan xulosa qilib, quyidagi ta’rifni keltirib o‘tiladi:

• **Algoritm** – bu kiruvchi ma’lumotlar asosida chiquvchi ma’lumotlarni hosil qilish jarayoni hisoblanadi.

Ixtiyoriy algoritm muhim xossalarga ega bo‘ladi:

- **Algoritmning aniqligi** – bu har bir qadam bajarilishining bir qiymatliligidir.
- **Tushunarlilik** – ijrochiga tavsiya etilayotgan ko‘rsatmalar uning uchun tushunarli bo‘lishi kerak, aks holda ijrochi oddiy amalni ham bajara olmay qoladi.
- **Diskretliligi** – masalani yechish jarayonini bajarilish vaqtida kompyuter yoki insonga qiyinchilik tug‘dirmasligi uchun bir necha sodda ko‘rinishlarga bo‘lish.
- **Ommaviylik** – belgilangan masalalar sinfini yechish uchun algoritmni foydaliligi jarayoni.
- **Natijaviylik** – oxirgi qadamlarda dastlabki ma’lumotlarga ega bo‘lgan kerakli natijani olishga imkon beruvchi algoritmning yakuni.

Algoritmning amaliyotda quyidagi turlari mavjud:

Chiziqli algoritmga – ketma-ketlikda bajariluvchi amallar va biror-bir shart tekshirilmasdan bajariluvchi algoritmlar kiradi**.**

Tarmoqlanuvchi algoritmga – belgilangan shartlarning o‘zgarishiga bog‘liq holda ko‘rsatmalarning variantlari oldindan mo‘ljallanadigan algoritmlar kiradi.

Sikllik algoritmga – alohida jarayonlar yoki jarayonlar guruhi bir necha marta bajariladigan algoritm kiradi.

Algoritmni yozish usullari turli ko‘rinishlarda bo‘ladi: so‘zli, formulali, jadvalli, grafikli, dasturli.

Informatsiya kompyutyerda qayta ishlanishi uchun u xom ashyo sifatida xizmat qiladi, ya’ni misol sifatida metallurgik ishlab chiqarishda metall ruda xom ashyo hisoblanganidek bo‘ladi. Lekin, ixtiyoriy xom ashyo qayta ishlash mobaynida samarali bo‘lishi uchun dastlabki tayyorgarlikka ega bo‘lishi kerak. Bu to‘laligicha informatsiyaga tegishli bo‘ladi. Demak, o‘rganish uchun hisoblash texnikasiga jalb etmoqchi bo‘lgan biror-bir hodisa oldindan mavjud bo‘lishi zarurdir. Birinchi navbatda qiziqtirayotgan hodisa haqidagi informatsiyalarni yig‘ish, so‘ng bu informatsiyani sistemalashtirish va sinflarga ajratish zarurdir. Shundan so‘ng, berilgan hodisani ifodalovchi model quriladi.

“Model” fransuzcha so‘zdan kelib chiqqan bo‘lib, dastlabki ma’nosi bu – biror fizik ob’ekt yoki hodisaning aniq ko‘rinishini belgilab beruvchi namunadir. Bunda fizik emas, balki **informatsion modellardan** foydalaniladi. U hodisani maxsus matematik apparat, grafik, diagrammalar yordamida ifodalay oladi. Modelda hodisaning harakterli belgilari va asosiy tomonlarini ochib berish imkoniyatlari mavjuddir. Shu bilan birga matematik va imitatsion modellashtirish ham mavjud.

Matematik modellashtirish – hodisa tadqiqoti va ifodasi uchun matematik apparatni qo‘llashdan iborat. Aniq matematik model ob’ektning holatini kuzatish va uni tahlil qilish imkoniyatini beradi.

Imitatsion modellashtirish – bu asosan, sanoatda qo‘llaniladi, hisoblash texnikasi va maxsus dasturiy ta’minot yordamida real mavjud bo‘lmagan qurilmada bir qator tekshirishlar o‘tkazish imkoniyatini beradi. Bunday modellashtirishni qo‘llash xom ashyo ishlab chiqarishni tezlashtiradi, chunki qurish va tadqiq qilish jarayoni qisqaradi, bunda xatolar miqdori va ularning bahosi kamayib boradi.

Bunda modelni qurib bo‘lgandan so‘ng, unga mos algoritmlarni yaratish bosqichlari boshlanadi.

Hisoblash mashinasi tilida (dasturlashda) ketma-ket buyruqlar ko‘rinishida berilgan masalalar yechimlarining algoritmlari **mashinaviy dasturlar** deb nomlanadi.

Mashinaviy dasturlar buyrug‘i yoki mashinaviy buyruq qo‘shimcha ko‘rsatma va tushunchalarsiz avtomatik holda bajariluvchi elementar mashina instruksiyasi deb nomlanadi.

Dasturlash – bu dastur tuzish bilan bog‘liq bo‘lgan nazariy va amaliy faoliyatdir.

Algoritmni mashina tiliga o‘girish jarayoni **translyatsiya** deb nomlanadi.

Mashina tilini «odamiylashtirishning» birinchi qadami – bu mashina kodiga o‘tkazuvchi dasturlar tuzishdan iborat bo‘ladi. Shu bilan birga arifmetik ifodalarni o‘giruvchi dasturlar yaratildi va nihoyat, 1958-yilda dasturlash tilida keng foydalaniladigan Fortran translyatori deb nomlangan dastur kirib keldi. Shundan so‘ng, bir nechta dasturlash tillari yaratiladi.

Kompyuter mashinaviy dastur buyruqlarini boshqargan vaqtda informatsiyaga ishlov beradi, buning uchun ish jarayonida turli berilganlardan foydalana boshlaydi.

Bu berilganlardan foydalanilganda quyidagilarga e’tibor beriladi:

Kiruvchi – kompyuter xotirasiga kiritiladi va masalani yechish uchun shart sifatida foydalaniladi.

Joriy yoki ichki – dasturning ichida informatsiyani saqlash va ishlov berish uchun ishlatiladi.

Chiquvchi – informatsiyaga ishlov berish natijasida dasturda hosil bo‘lgan jarayonlar. Matn, grafik, videotasvir va boshqalar ko‘rinishda bo‘lishi mumkin.

Hodisa tadqiqoti, masala yechimi uchun hisoblash texnikasi yordamida qabul qilish kerak bo‘lgan amallarning umumiy tartibini quyidagicha tasvirlash mumkin:

Hodisa, jarayon bu- masala

– model
– algoritm
– dastur
– kompyuter
– natija

Berilgan algoritmlarning murakkabligi. Hisoblash muammolari cheklangan holda resurslaridan qisqa vaqt ichida oqilona foydalanish kerak. Bunday holat algoritmning vaqt va fazoviy murakkabligi tushunchasiga olib boradi. Qoida tariqasida shuni aytish zarurki, algoritm turli vaqtlarda bajarishi mumkin bo‘lgan turli xil amallarni o‘z ichiga oladi.

Algoritmlarni baholash uchun ko‘pgina mezonlar mavjuddir. Odatda algoritmlarda kirituvchi hamda berilganlarni ko‘payishida masalani yechish uchun kerak bo‘ladigan **vaqt** va **xotira** hajmlarini o‘sish tartibini aniqlash muammosiga katta e’tibor qaratiladi. Har bir aniq kiritiladigan masalaga berilganlarni miqdorini aniqlovchi qandaydir sonni bog‘lash zarur hisoblanadi. Bu yerda son masalaning kattaligi deb qabul qilinadi. Masalan, ikkita matritsani ko‘paytirishda masalaning o‘lchami va kattaligiga e’tibor berish zarurdir yoki graflar haqidagi masalada o‘lcham sifatida graf uchlarining soni bo‘lishi eng muhim ish hisoblanadi.

Algoritmda sarflanayotgan uzoq vaqt, masalaning o‘lchami sifatida uning **vaqt bo‘yicha qiyinligi** deb hisoblanadi. Bunday funksiyada masalaning hajmi oshganda limit ostidagi o‘zgarish yuqori bo‘ladi, bu esa **asimptotik qiyinlik** deb aytiladi.

Murakkablikni baholash – bu algoritmlarning murakkabligi odatda bajarilish vaqti yoki ishlatilgan xotira bo‘yicha hisoblanadi. Ikkala holatda ham, murakkablik jarayoni bu kiritilgan ma’lumotlarning hajmiga bog‘liq bo‘ladi: Bunda misol sifatida 100 ta elementdan iborat massivi xuddi shunga o‘xshash 1000 ta elementdan iborat massivga qaraganda tezroq qayta ishlanadi. Bunda

protsessorga bog‘liq, ma’lumotlar turi, dasturlash tili va boshqa ko‘plab parametrlarga ham bog‘liq. Bu yerda faqatgina **asimptotik** murakkablik muhim rol o‘ynaydi, bunda uning biri kirish ma’lumotlarining kattaligi bo‘lsa, yana biri uning cheksizlikka intilayotgandagi murakkablikga olib keladi.

Masalan, ba’zi bir algoritmga kirish ma’lumotlarining n ta elementlarini qayta ishlash uchun $2*n^3$ 6+n ta shartli amallarni bajarish kerak. n ning ortishi bilan ishning umumiy davomiyligi n ning kubi uni 2 ga ko‘paytirgandan yoki 6+n 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 ishlatib kelinadi. 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 ma’lum qiladi.

O(n) **chiziqli murakkablik –** bu saralanmagan massivdagi eng katta elementni topish algoritmidir. Bu yerda maksimal ekanligini aniqlash uchun massivning barcha n elementlarini e’tibor berish kerak bo‘ladi.

O(log n) **– logaritmik murakkablik**. Bu eng oddiy misol bo‘lib, ikkilik qidirishdir. Agar massiv saralangan bo‘lsa, uni yarmiga bo‘lish orqali ma’lum bir qiymatga ega ekanligini tekshirish zarur. Bunda o‘rta element tekshirib ko‘riladi, agar u kattaroq bo‘lsa, unda massivning ikkinchi yarmini tashlab yuborish zarur. Agar kichikroq bo‘lsa, unda aksincha - dastlabki yarmi tashlanadi va shu tarzda ikkiga bo‘linish davom ettiriladi, natijada (logn) elementlarini tekshirsa bo‘ladi.

$O(n^2)$ **– kvadratik murakkablik**. Bunday murakkablik, masalan, element qo‘shilishi natijasida yangi saralash algoritmiga ega bo‘linadi. Bu kanonik dasturda ikkita ichki sikldan iborat bo‘ladi: biri butun massivni bosib o‘tish, ikkinchisi esa allaqachon ajratilgan qismdan keyingi element uchun joy

topishdir. Shunday qilib, amallar soni n*n, y'ni n^2 kabi massiv o‘lchamiga bog‘liq bo‘lib qoladi.

Murakkablikning boshqa ko‘rinishlari ham mavjuddir, ammo ularning barchasi bir xil prinspda ishlaydi.

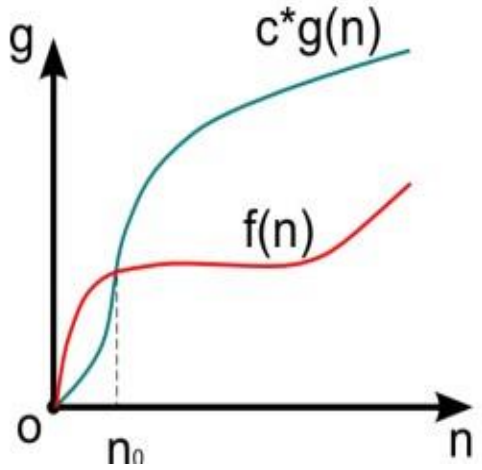
Bunday holatlarda algoritmning ishlash vaqti umuman kiritilgan ma'lumotlarning hajmiga bog‘liq bo‘lmay ham qoladi. Bu holda murakkablik O(1) bilan belgilanadi. Masalan, massivning uchinchi elementi qiymatini aniqlab olish uchun elementlarni eslab qolishingiz yoki ular orqali bir necha bor saralash olib borish shart emas. Siz har doim ma'lumotlarni kiritish oqimidagi uchinchi elementni kutishingiz kerak va bu esa siz uchun natija ega bo‘ladi, bu har qanday ma'lumotni yana bir bor hisoblash uchun bir xil vaqtni egallaydi.

Baholash muhim bo‘lgan taqdirda xotiradan xuddi shu tarzda yana amallar bajarila boshlanadi. Biroq, algoritmlarda kirish ma'lumotlarining hajmi boshqalarga nisbatan ko‘proqligi sezilarli darajada xotiraning kattaroq joyidan foydalanishi mumkin, ammo ular shunday bo‘lsa ham tezroq ishlaydi. Bu hozirgi sharoit va talablar asosida muammolarni hal qilishning eng yaxshi usullarini tanlashga imkon beradi.

Algoritmlar murakkabligining o‘sish tartibi. Murakkablikning o‘sish tartibi katta kirish hajmi uchun algoritmning murakkablik funksiyasining taxminiy xatti-harakatini ifodalaydi. Bundan kelib chiqadiki, vaqt murakkabligini baholashda elementar o‘rtasidagi amallarni ko‘rib chiqishning hojati yo‘q bo‘ladi, bunda faqat algoritm qadamlarini ko‘rib chiqish kifoyadir.

Algoritm qadami – bu ketma-ket joylashtirilgan elementar va amallar to‘plami, uning bajarilish vaqtiga hamda kirish qadamiga bog‘liq emas, ya'ni yuqoridan qandaydir chegaraviy qadamlar bilan cheklangandir.

Asimptotik baholashning ko‘rinishlari. F(n)>0 murakkabligini, bir xil tartibdagi g(n)>0 funksiyasini hamda n>0 o‘lchamliligini 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) bo‘ladi.



1-rasm. Asimptotik baholash

Bu holda g(n) funksiyasi f(n) uchun asimptotik-aniq baho hisoblangan bo‘ladi.Agar f(n) algoritmning murakkablik funksiyasi bo‘lsa, unda murakkablik tartibi f(n) uchun - O(g(n)) deb qabul qilinadi. Ushbu ibora doimiy koeffitsiyentgacha g(n) dan tez o‘smaydigan funksiyalar sinfini belgilab beradi.

Algoritmlar samaradorligini hisoblash holati. Algoritmlar samaradorligini hisoblashda kirish ma’lumotini qanday tanlash ko‘rilayotgan algoritmni bajarilishiga yaxshigina ta’sir ko‘rsatib o‘tadi. Masalan, agar kirish ma’lumotlari allaqachon saralangan bo‘lsa, ba’zi saralash algoritmlari juda yaxshi ishlaydi, ayrimlari ancha past samaradorlik bilan ishlashi mumkin. Agar kirish ma’lumotlari saralanmagan, tartibsiz bo‘lsa, u holda buni aksi bo‘lishi mumkin. Shuni e’tiborga olgan holda algoritmlar tahlil qilinishi zarurdir.

Eng yaxshi holatlari. Bunda algoritm kirish ma’lumotlari tez bajarilishi uchun qulay ko‘rinishda bo‘lishi lozim, ya’ni algoritm kam sonli amallar bilan bajariladi va kam vaqt talab etadi. Masalan, agar tuzilmadan qidirayotgan element tuzilmaning birinchi elementi bo‘lib hisobga olinsa, uni qidirishda eng kam vaqt sarflanadi. Chunki, tuzilmaning uzunligidan qat’iy nazar, bitta solishtirish yetarli bo‘ladi. Algoritmlarni eng yaxshi holatlarini tahlil qilishda odatda, bajarilish vaqti konstanta birga teng bo‘lishi sababli ko‘pincha bunday holatlar tahlillarda ko‘rilmaydi.

Eng og‘ir holatlari. Bunda kirish ma’lumotlari algoritm bajarilishi uchun eng yomon og‘ir holatda amalga oshiriladi va juda sekin bajariladi. Bu eng og‘ir holat tahlillarida muhim hisoblanadi, chunki bu algoritm bajarilishi uchun ketishi mumkin bo‘lgan maksimal vaqtni tahlil qilishga sabab bo‘ladi. Masalan, qidirilayotgan element tuzilmaning eng oxirgi elementi bo‘lsa, uni topish uchun barcha solishtirishlar ketma-ket amalga oshiriladi.

O‘rtacha holatlari. Bunda algoritmning o‘rtacha ishlash imkoniyatini beruvchi kirish ma’lumotlari to‘plami muhokama qilinadi.

Ma’lumotlar va uni ifodalash bosqichlari

Ma’lumot – bu birorta qiymat yoki qiymatlar to‘plamidir. Masalan, bu bironta eksperiment natijalari yoki talabalarning imtihon ballaridan iborat bo‘lishi mumkin.

Ma’lumotlar tuzilmasi - bu xotirada tashkil etiladigan elementlar yig‘indisi hisoblanib, ular ustida dastur yordamida amallar bajarish mumkin.

Ma’lumotlar tuzilmasi – bu bironta toifaga tegishli bo‘lgan va o‘zaro ma’lum munosabatga ega bo‘lgan elementlar to‘plamidir.

Ma’lumotlar tuzilmasi elementi – bu qiymatlar to‘plamining bir bo‘lagi bo‘lib, u tuzilma elementini qiymatlar jamlanmasi hisoblanadi, misol uchun talabalarning sharifi, ismi, yoshi har bir fandan o‘zlashtirish ko‘rsatkichi va boshqalarni keltirish mumkin. Bu yerda elementlar ikkitaga bo‘linishi mumkin.

Element sifatida ma’lumotlarning guruhi olib qaraladi. Bunda yerda elementlar yana qism bo‘laklarga bo‘linishi ham mumkin. Masalan, ota-onalar maydoni, talabalarning ota va onalari haqida ma’lumot saqlaydigan alohida maydonlardan tashkil topishi mumkin.

Elementar bu bo‘linmas hisoblanib, bunda element qism bo‘laklarga ajratilmagan holda bo‘ladi.

Ob’ekt – bu xususiyatlar va attributlariga ega bo‘lgan va bu xususiyatlarga qiymat qabul qilishi mumkin bo‘lgan tuzilmadir. Masalan, talaba bu ob’ekt deb qaralishi mumkin.

Maydon – bu ob’ektlarning attributlari yoki xususiyatlarini ifodalovchi tushuncha bo‘lib, sonli yoki son bo‘lmagan qiymatlarni o‘zlashtirishi imkoniyatini beradi.

Yozuv – bu bironta ob’ektga tegishli turli toifadagi maydonlar yig‘indisidir.

Fayl – bu bir-biriga bog‘liq bo‘lgan yozuvlar jamlanmasidir. Masalan, bunda barcha talabalar haqidagi yozuvlarni o‘z ichiga olishi tushuniladi.

Kalit – bu yozuvdagi maydon bo‘lib, aynan shu yozuvni boshqa yozuvlardan ajratib turishga xizmat qiladi, uning qiymati boshqa yozuvlarda takrorlanmas bo‘ladi.

Ba’zida bittadan ko‘p maydonlar qiymatlari elementlararo betakror bo‘lishi mumkin va bu **karrali kalit** deyiladi. Ko‘pincha bu yerda asosiy kalit hisoblanadigan bitta maydon ma’lumoti ishlatiladi va u **boshlang‘ich kalit** deb yuritiladi, qolganlari esa **alternativ kalit** deb nomlanadi. Ba’zida esa yozuvlarning yagona qiymatli kalit maydonni yo‘qligi sababli kalit sifatida bir nechta maydonlar hisobga olinadi va ularga **tarkibli kalit** deb ataladi. Eng yomon holatda, ba’zan shunaqa ham bo‘lishi mumkinki, unda bironta maydon kalit bo‘la olmasligi, har bit elementga qo‘shimcha, qiymati yagona bo‘lgan kalit maydonlari kiritiladi.

Axborotlar. Ko‘pincha ma’lumot va axborot tushunchalarini bir xil ma’noda ishlatilishidir. Lekin, aslida esa axborot bu ma’lumotga qaraganda kengroq tushuncha hisoblanadi.

Axborot – bu qayta ishlangan ma’lumotlardir. Ma’lumotlar esa qiymatlarning yig‘indisi, ya’ni unda bironta yakuniy xulosa bermaydi. Qaror qabul qilishda unda birorta asos topilmydi. Ma’lumot ma’lum qoidalar asosida qayta ko‘rilgach, yangidan hosil qilingan ma’lumotlar jamlanmasi axborotga aylanadi va qaror qabul qilishda foydali bo‘lib xizmat qiladi.

Ma’lumotlar toifasi – bu qandaydir qiymatlar yig‘indisi hisoblanib, ular ustida ma’lum amallarni bajarish sanaladi. Ma’lumotlar toifalari dasturda

oldindan va ko‘rilgan yoki foydalanuvchi tomonidan ma’lum qilingan bo‘lishi mumkin bo‘lgan quyidagilarni nazarda tutuvchi to‘plamlardan iboratdir: 1. Qiymatlar to‘plami;

Amallar to‘plami. Masalan, int-butun toifalar ustida bajariladigan arifmetik amallardir (,-,*,/).

Ma’lumotlarni ifodalash bosqichlari hamda tarkibi

Ma’lumotlar toifalari uch turga ajratiladi:

1. Primitiv toifalar – bu ma’lumotlarning eng sodda toifalaridir. Bular oldindan ma’lum bo‘ladigan sozlangan, toifalar bo‘lib, turli dasturlash tillarida turlicha bo‘lishi hisobga olinadi.

2. Foydalanuvchi tomonidan aniqlanadigan toifalarda qachonki, mavjud sozlangan toifalar qo‘yilgan masalani yechishga yetarli bo‘lmasa, o‘shandagina hisobga olib, amalga oshiriladi.

3. Abstrakt toifalar – bu ma’lumotlar toifalarining mantiqiy xususiyatlarini aniqlashda eng foydali qurol sifatida hisobga olinadi. “Abstrakt toifa” atamasi aslida bazaviy matematik tushunchaga bog‘liqdir. Ushbu toifalardagi ma’lumotlar qisman apparat va dasturiy ta’minotlar yordamida tuzilma sifatida amalga oshirish mumkin bo‘ladi. Abstrakt toifalarni matematik tushunchalar sifatida aniqlaganda, muhit va vaqtiy munosabatlarni e’tiborga olmaydi. Bular amalga oshirish masalalari bo‘lib hisoblanadi.

Ma’lumotlar tuzilmasini klassifikatsiya qilish

Ma’lumotlar tuzilmasi. Ma’lumotlar turli yo‘llar asosida tashkil etilish mumkin, matematik yoki mantiqiy modellarni tashkil etilishi ma’lumotlar tuzilmasi deb yuritiladi. Aniq bir ma’lumotlar tuzilmasini tanlash quyidagilar orqali amalga oshiriladi:

- Real voqe’likda elementlararo munosabatni yaqqol ifodalay olishi maqbul yechimdir;

- U shunday sodda ko‘rinishda tuzilishi kerakki, zarur bo‘lganda uning ustida samarali amal bajarish noqulaylik tug‘dirmasin.

Ma’lumotlar tuzilmasini o‘rganishda quyidagilarga e’tibor qaratish kerak:

- Tuzilmaning mantiqiy ifodalash yo‘llari;
- Tuzilmaning fizik amalga oshirish yo‘llari;
- Tuzilmani sifatli tahlili, ya’ni elementlarni saqlash uchun qancha xotira hajmi sarflanishini aniqlash hamda qayta ishlashga ketadigan vaqtni hisoblash eng asosiy masala hisoblanadi.

Ma’lumotlar tuzilmasi – informatsion ob’ektning umumiy xossasi bo‘lib, mazkur xossa bilan biror-bir dastur o‘zaro aloqador bo‘lishi lozim. Ushbu umumiy xossa quyidagilar orqali tavsiflanadi:

1) Bu mazkur tuzilmani qabul qilishi mumkin bo‘lgan qiymatlari majmuasi;
2) mumkin bo‘lgan amallar jamlanmasi;
3) tashkil etilganlik tasnifi va unga mosligi.

Oddiy ma’lumotlar tuzilmalarini ba’zi vaqtda ma’lumotlar toifalari deb ham qaraladi.

Odatda, ma’lumotlarni tasniflashda quyidagi ko‘rinishdagi bosqichlarga e’tibor qaratiladi.

1) abstrakt, ya’ni matematik bosqichlar;
2) mantiqiy, ya’ni dasturiy bosqichlar;
3) fizik, ya’ni jismoniy bosqichlar.

Ma’lumki, ixtiyoriy ob’ekt, xodisa yoki biror-bir jarayon tadqiq qilinayotganda uning modeli tuzib olinadi. Model turlicha bo‘lishi ham mumkin, misol uchun, matematik model, fizik model va boshqa modellarni o‘z ichiga oladi. Ob’ekt, xodisa yoki biror-bir jarayonni matematik model qurildi, degani o‘sha qaralayotgan tizimni ma’lum bir matematik qonuniyatlar orqali, ya’ni matematik formulalar orqali ifodalanish jarayoni tushuniladi.

Mantiqiy bosqich ma’lumotlar tuzilmasini biror-bir dasturlash tilida ifodalanishi bosqichiga qaratilgan.

Fizik (jismoniy) bosqichda esa informatsion ob’ektni mantiqiy tavsiflanishiga mos ravishda EHM xotirasida akslantirilishda qaratilgan. EHM xotirasi kichik bo‘lganligi sababli, xotirani bo‘lib chiqish va uni boshqarish muammosi yuzaga chiqadi.

Bundan ko‘rinib turibdiki, mantiqiy bosqich bilan fizik bosqichlar bir-biridan juda katta farqli tomonlari mavjuddir. Shu sababli, hisoblash tizimlarida mantiqiy bosqichni fizik bosqichga va aksincha, fizik bosqichni mantiqiy bosqichga akslantirish muamosi o‘z-o‘zidan vujudga keladi.

Ma’lumotlarni asosiy abstrakt turlari

Abstraktli ma’lumotlarni asosiy turlari – bu ixtiyoriy abstrakt bosqichda tuzilmani <D, R> juftlik ko‘rinishda ifodalay olishi mumkin, bu yerda D – elementlarning chekli to‘plami bo‘lib hisoblanadi. Bu yerda elementlar ma’lumotlar turlari yoki ma’lumotlar tuzilmasi bo‘lishi ham mumkin. R – esa munosabatlar to‘plami bo‘lib, mazkur munosabatlarning xususiyatlari abstrakt bosqichlarda ma’lumotlar tuzilmalarini turlarini aniqlay oladi.

Ma’lumotlar tuzilmalarini asosiy turlari:

1) **To‘plam** - bu munosabat to‘plami bo‘sh R0 bo‘lgan elementlar jamlanmasidir.

2) **Ketma-ketlik** – bu shunday tuzilmaki, bunda R to‘plam faqatgina bitta chiziqli munosabatdan iborat ya’ni, birinchi va oxirgi elementdan tashqari har bir element uchun o‘zidan keyin va oldin keladigan element yig‘indisidir.

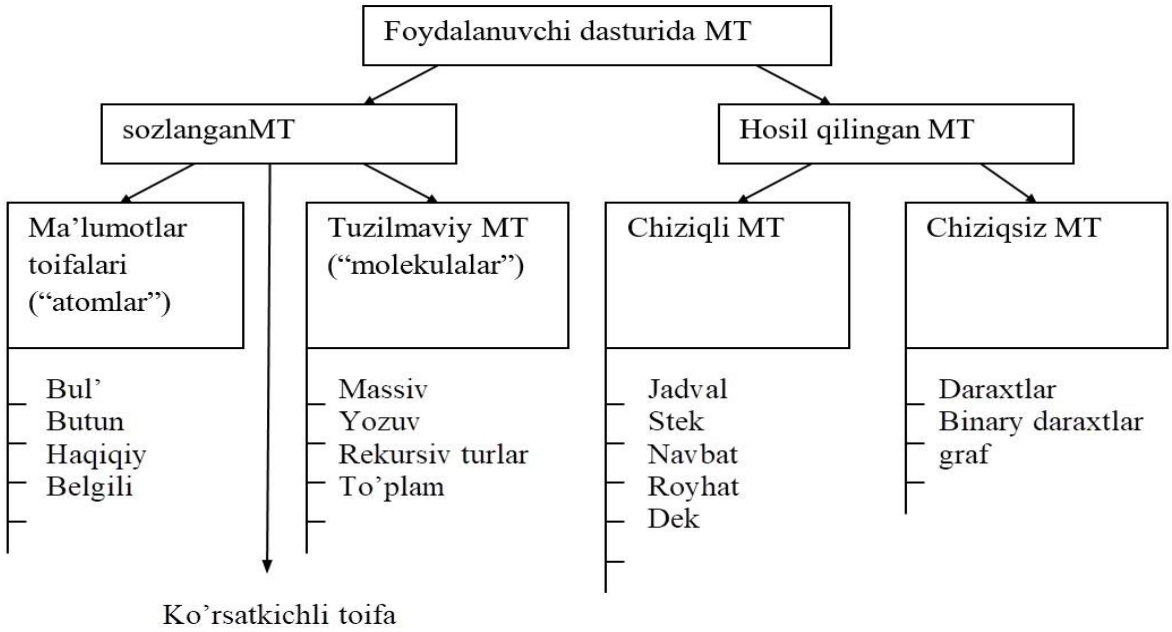
3) **Matritsa** – bu shunday tuzilmaki, bunda R munosabatlar to‘plami ikkita chiziqli munosabatdan tashkil topgan holatidir.

4) **Daraxt** – bunda R to‘plam bitta munosabatdan tashkil topgan iyerarxik tartibda bo‘ladi.

5) **Graf** – bu bunda R munosabatlar to‘plami bitta binar tartibli munosabatdan tashkil topganligidir.

6) **Gipergraf** – bu shunday ma’lumotlar tuzilmasi bo‘lib, bunda R to‘plam ikki yoki undan ko‘proq turli tartibdagi munosabatlardan tashkil topgan bo‘ladi.

2-rasmda foydalanuvchi o‘zining dasturida va EHM xotirasida ma’lumotlar tuzilmasini klassifikatsiya qilish jarayoni ko‘rsatilgan.

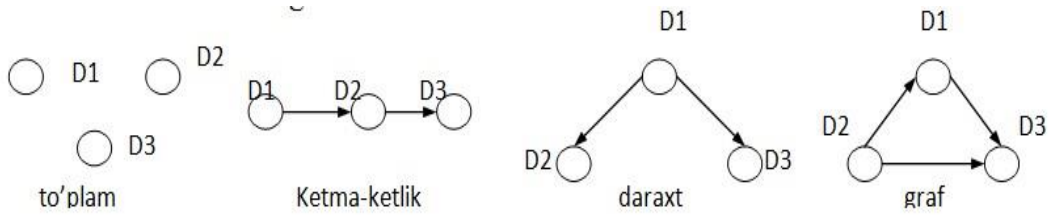


2-rasm. EHM xotirasida ma’lumotlar tuzilmasini klassifikatsiya qilish jarayoni

Ma’lumotlar tuzilmasi klassifikatsiya qilishda asosiy ko‘rinish - bu ma’lumotlar tuzilmasini dastur ishlashi mobaynida o‘zgarish jarayoni hisoblanadi.

Misol uchun agar dastur bajarilishi davrida elementlar soni yoki ular orasida o‘zgarishlar bo‘lsa, u holda bunday ma’lumotlar tuzilmasi dinamik ma’lumotlar tuzilmasi deb aytiladi, aks holda statistik ma’lumotlar tuzilmasi deb yuritiladi.

3-rasmda ma’lumotlar tuzilmasiga misollar keltirilgan.



3-rasm. Ma’lumotlar tuzilmasiga misollar

Ma’lumki, matematikada o‘zgaruvchilarni, ularning ba’zi bir kerakli tavsiflariga mos holda klassifikatsiya qilish jarayoni qabul qilingan.

O‘zgaruvchilarga misol sifatida quyidagilarni aytib o‘tish mumkin: kompleks o‘zgaruvchilar, haqiqiy o‘zgaruvchilar, mantiqiy o‘zgaruvchilar, bundan tashqari bir nechta ba’zi bir qiymatlarni qabul qiluvchi o‘zgaruvchilar va boshqalardir.

Ma’lumotlarni qaytadan ishlash jarayonida ularni klassifikatsiya qilish ham katta ahamiyat talab etadi. Bu yerda ham xuddi shunday klassifikatsiya qilinayotganda har bir konstantalar, o‘zgaruvchilar, ifoda yoki funksiyalar biror-bir toifalarga tegishli bo‘ladi, - degan tamoyillarga asoslangan holda ish yuritiladi. Umuman olganda, turli xil toifalar o‘zgaruvchi yoki ifodalar qabul qilishi mumkin bo‘lgan qiymatlar to‘plamlari orqali ifodalanadi.

Tuzilmalari ustida quyidagi amallar ketma-ketligini bajarish mumkin bo‘ladi:

1. **Ko‘rikdan o‘tkazish** - tuzilma elementlariga bir martadan murojaat qilish amali.

2. **Kiritish** – tuzilmaga yangi element kiritish va chiqarish amali.

3. **O‘chirish** – tuzilmadan bironta elementni o‘chirish yoki tozalash amali. Bunda element shunday o‘chirilishi kerakki, qolgan elementlar o‘z holatida saqlangan holatda bo‘lishi kerak, ya’ni ayrim tuzilmalarda nosozlik holatlari yuzaga kelmasligi kerak.

4. **Qidirish** – bu tuzilmadan birorta elementni joylashgan o‘rnini aniqlash amalidir.

5. **Saralash** – bu elementlarni ma’lum bir tartibda joylashtirish amali hisoblanadi.

6. **Birlashtirish** – bu ikkita tuzilmani birlashtirish amali hisoblanadi.

Mavzuga oid test savollari

1. Ma’lumotlar tuzilmasi nima?

a) bu ma’lumot elementlari va ular orasidagi munosabatlar majmuasi.

b) bu ma’lumot elementlari majmuasidir.

c) bu elementlar orasidagi munosabatlar amalidir.

d) bu ma’lumot elementlari va ular orasidagi relyatsion munosabatlar majmuasidir.

2.Ma’lumotlar tuzilmasi ustida qanday to‘rtta asosiy amal bajariladi?

a) yaratish, o‘chirish, tanlash (ruxsat olish), yangilash.

b) yaratish, o‘chirish, kengaytirish, yangilash holati.

c) yangilash, yaratish, tanlash (ruxsat olish), kengaytirish.

d) yaratish, o‘chirish, kengaytirish, tanlash holatlari.

3. Ma’lumotlar tuzilmasi mazmunli (matematik) bosqichda ...

a) konkret ob’ektning qayta ishlash, ularning xussusiyatlari va munosabatlarini tadqiq qilinadi.

b) kompyuter xotirasida ma’lumotlarni aks ettirilishi tadqiq qilinadi.

c) berilgan talabalar bo‘yicha algoritmni ishlab chiqilishi va tadqiq qilinishi.

d) dasturni yaratish jarayoni tadqiq qilinishi.

4. Ma’lumotlar tuzilmasi mantiqiy bosqichda ...

a) berilgan talabalar bo‘yicha algoritmni ishlab chiqilishi tadqiq qilinadi.

b) kompyuter xotirasida ma’lumotlarni aks ettirilishi tadqiq qilinadi.

c) konkret ob’ektning qayta ishlash, ularning xussusiyatlari va munosabatlari tadqiq qilinadi.

d) dasturni yaratish jarayoni tadqiq qilinadi.

Nazorat savollari

1. Ma’lumotlar tuzilmasi bu nima? Ular haqida gapirib bering.
2. Ma’lumotlar elementlariga nimalar kiradi?
3. Ma’lumotlar va ularni ifodalash bosqichlari nimalardan iborat?
4. Ma’lumotlar tuzilmasini klassifikatsiyasi deganda o‘z fikringizni ayting?
5. Ma’lumotlarni asosiy abstrakt turlari nimalardan iborat bo‘lishi mumkin?

Xulosa

Ma’lumki, kompyuter o‘z hisoblash usuli bilan birga tezkor va aniq hamda shu bilan birga, butunlay o‘zgacha amallarni bajaruvchi universal texnologiya hisoblanadi. U turli masalalarni yechishda qandaydir aniq ketma-ketlik asosida har qanday ma’lumotni tartibga soladi. Undan turli xil vazifalarni berganda kompyuter shunga mos qismlarni o‘zi o‘ylab topadi, – degan fikr xato. Chunki, kompyuter ishlashi uchun aniq va to‘liq masalani tushuntirish kerak. Bu bilan qo‘yilgan masala uchun algoritmni to‘g‘ri taqsimlash usuliga e’tibor qaratiladi.

Shundan so‘ng, qurilmada berilgan har qanday ma’lumotli topshiriqdan qandaydir ketma-ketlik asosida ijobiy natija olinadi.

1.2. Ma’lumotlar tuzilmalarining umumiy ko‘rinishlari. Ma’lumotlarning sozlangan turlari: massivlar, vektorlar, yozuvlar, to‘plamlar va ko‘rsatkichli turlar

Reja:

1. Ma’lumotlarning sozlangan turlari.
2. Massivlar, vektorlar va ular ustida amallar bajarish.
3. Yozuvlar, to‘plamlar va ular ustida amallar.
4. Ko‘rsatkichli turlar.

Tayanch iboralar: bazaviy toifalar, statik tuzilmalar, dinamik tuzilmalar, yarim statik tuzilmalar, massiv, vektor, to‘plamlar, ko‘rsatkichlar.

Ma’lumotlarning sozlangan turlari

Ma’lumot toifalarini shartli ravishda ikki xil turga ajratish mumkin:

1. Sozlangan toifalar: haqiqiy, butun, mantiqiy, belgili, ko‘rsatkichli bo‘ladi. Ushbu oddiy sozlangan toifalardan tuzilmaviy hamda nostandart toifalar yaratilishi mumkin.

2. Hosil qilinadigan toifalar. Bu turdagi toifalarni foydalanuvchilar sozlangan toifalarda foydalanib, o‘zlari turli xil ko‘rinishda yaratishlari mumkin.

Ularga massivlar, yozuvlar, vektorlar, to‘plam, ro‘yxatlar, jadvallar, stek, navbat, daraxtlar, dek, binar daraxtlar, graflar misol bo‘ladi. Bu turdagi toifalarni yaratish va ustida amal bajarish uchun standart funksiyalar kutubxonasida tayyor funksiyalar mavjud, ammo, bu toifalarni foydalanuvchilar noldan boshlab o‘zlari istalgan ko‘rinishda o‘zgartirib oladilar.

C dasturlash tilida standart toifalarda bo‘lgan butun **(**int, long, short) toifa, haqiqiy (float, double, long double) toifa, belgili (char) toifa, mantiqiy **(**bool) toifa, ko‘rsatkichli toifalar va tashqari C dasturlash tilida yana bitta satr (string) toifasi ham mavjuddir.

Statik tuzilma deb, dastur bajarilishi jarayonida elementlari soni yoki ular orasidagi munosabatlar o‘zgaruvchi bo‘lgan tuzilmalarga aytiladi. Shunday ekan, quyida statik tuzilmalarni ko‘rib o‘tiladi. Ko‘plab texnika tushinadigan tillarda ma’lumotlar foydalanuvchi tomonidan beriladigan standart toifalarga ajratib chiqiladi. 1-rasmda ma’lumotlar tuzilmasi toifalari tasvirlangan.

1-rasm. Ma’lumotlar tuzilmasi toifalari

Ma’lumotlarni quyidagi beshta tur standart o‘zgaruvchilar toifalariga ajratish mumkin:

a) butun toifa (INT);
b) mantiqiy toifa (BOOL);
c) haqiqiy toifa (FLOAT);

d) ko‘rsatkichli toifa (*).

e) belgili toifa (simvol) (CHAR);

Butun toifa – INT. Mazkur toifa butun sonlar to‘plamining qandaydir qism to‘plamli toifasi bo‘lib, uning o‘lchami EHM konfiguratsiyasiga bog‘liq ravishda o‘zgaradi. Agar butun sonni mashinada tasvirlash uchun y ta razryaddan foydalanilsa (bunda qo‘shimcha koddan foydalanilganda), u holda z butun sonning qiymat qabul qilish oralig‘i quyidagicha bo‘lishi mumkin, ya’ni quyidagi shartni qanoatlantirishi shart: -2 y-1<z< 2 y-1.

Butun toifadagi ma’lumotlar ustida bajariladigan amallar haqiqiy amalga oshiriladi, - deb hisobga olinadi, ushbu amallar matematikada qabul qilgan o‘zgarmaslarga bo‘ysunadi. Agar ushbu toifada amallar bajarilayotganda natija ruxsat etilgan oraliqlardan chiqib ketsa, u holda hisoblash tezda to‘xtatiladi. Bunday holatlar to‘lib ketish holatlari deb aytiladi.

Mazkur toifaga kiruvchi sonlar ikkiga ajraladi: Biri ishorali va ikkinchisi ishorasiz. Ularning har bir uchun mos ravishda qiymatlar qabul qilish oraliqlari mavjuddir:

a) ishorasiz sonlar uchun (0.2y-1);

b) ishoralilar uchun (-2Y-1.2Y-1-1).

2-rasm. Sonlar

Sonlar mashinada qayta ishlanayotganda vaqtda ularning ishorali ko‘rinishlaridan foydalaniladi. Agar mashina so‘zi yozuv, komandalarini qayta ishlash va ko‘rsatkichlar uchun foydalanilayotgan bo‘lsa, u holda sonning ishorasiz ko‘rinishi orqali bu jarayondan puxta foydalanadi.

Bundan tashqari butun sonlar ustida – ayrish, qo‘shish, ko‘paytirish, ularni bo‘lish, qoldiqni tashlab yuborish orqali berilgan modul bo‘yicha hisoblash, bo‘lishda qolgan qoldiqni hisoblash, berilgan sonlar to‘plamining eng kichik va

eng katta elementlarini aniqlash, butunni darajaga oshirish, sonning qiymatiga qarab o‘zidan keyingi yoki oldingi sonni aniqlashdir. Bu operatsiyalarning natijalari ham butun sonlardan iborat bo‘ladi.

Butun sonlar ustida operatorlar va !, <, <, >, > taqqoslash amallarni ham bajarib borish mumkin. Ammo, bu operatsiyalarning natijalari INT toifasiga kirmay qoladi, ular BOOL toifasiga tegishlidir.

Haqiqiy toifa. Haqiqiy toifa ko‘pincha, kasr qismlari bor chekli sonlar to‘plamidan tashkil topgan bo‘ladi. To‘plamni chekli bo‘lish sharti EHMda sonlarni ifodalash chegaralanganligi bilan bog‘liqdir. Haqiqiy sonlar ustida bir nechta amallarni bajarish mumkin: ayirish, qo‘shish, ko‘paytirish, bo‘lish, trigonometrik funksiyalarini hisoblay olish, darajaga ko‘tarish, kvadrat ildizdan chiqarish, logarifmlay olish, minimum va maksimum elementlarni topa olish va boshqa amallar bajarish mumkin. Bularning natijalari ham haqiqiy toifaga tegishlidir. Bu yerda binar amallarga nisbatan masalaning yechimlari mantiqiy toifaga tegishlidir.

EHM xotirasida haqiqiy sonlar asosan, qo‘zg‘aluvchan nuqtalar formatida saqlanib boradi. Bu formatda z haqiqiy son quyidagi ko‘rinishda ifodalanadi: z /-M * q(/-P) – sonning yarim logarifmik ko‘rinishidagi ifodalanishini quyidagi rasmda keltirilgan. 937,56 93756 * 10-20,93756 * 103

3-rasm. Sonning yarim logarifmik ko‘rinishidagi ifodalanishi

Mantiqiy toifa – bu mazkur toifa mantiqiy mulohazalarni to‘g‘riligini aniqlash uchun turli xil dasturlash tillarida turlicha ifodalaniladigan ifodalarni ikkita true (1), false (0) ko‘rinishda anglatadi.

Mantiqiy ma’lumotlar ustida quyidagi mantiqiy operatsiyalarni bajarsa bo‘ladi: konnuksiya hamda dizungksiya yoki i inkor (yo‘q), qolaversa, qiyinroq bo‘lgan ekvivalentlik, implikatsiya, chiqarib tashlash va boshqa operatsiyalarni

bajarsa bo‘ladi. Bu yerda keltirilgan ixtiyoriy operatsiyaning natijasi shuni ko‘rsatadiki, ularning barchasi mantiqiy qiymatga ega bo‘ladilar. Bunda mantiqiy qiymatni qurilma xotirasida saqlash uchun bitta bit yetarli bo‘ladi. 4-rasmda asosiy mantiqiy funksiyalarning haqiqiy ko‘rinish jadvali berilgan.

A	B	not A	A or B	A and B
1	1	0	1	1
1	0	0	1	0
0	1	1	1	0
0	0	1	0	0

4-rasm. Asosiy mantiqiy funksiyalarning haqiqiy ko‘rinishi

Belgili toifa. Belgili toifaga belgilarning chekli to‘plami, ularga lotin alifbosidagi harflar va unda yo‘q kirill harflar, matematik va maxsus belgilar, o‘nlik raqamlar tashkil topgan. Belgili ma’lumotlar inson va hisoblash texnikasi o‘rtasidagi aloqani o‘rnatishda katta ahamiyatga egadir. Ko‘pincha, dasturlashning har bir tizimida belgilar to‘plami yoritilgan bo‘lib, ular turli tizimlarda turli xil bo‘lishi ko‘zda tutilgan. Shu bilan birga, ular tartiblangan bo‘lib, har biri uning elementiga aniq bir sonli kodga mos qo‘yilib, u to‘plamdagi tartib raqamini anglatadi. Bunda belgini sonli kodiga o‘tib, relyatsion operatorlardan foydalanib, simvollarni taqqoslash mumkin bo‘ladi. Bunday taqqoslashlarning natijalari BOOL toifasiga tegishli bo‘ladi.

C dasturlash tilida belgili toifadan tashqari belgilar massividan tashkil topgan satrli toifalar bilan ham ishlash imkoniyatini beradi, ya’ni char [] ko‘rinishda amalga oshiriladi. Shu o‘rinda aytib o‘tish lozimki, satrlar bilan ishlashda belgilar massividan tashqari satrlar bilan ishlashga mo‘ljallangan maxsus kutubxonalari mavjud bo‘lib, ular String deb nomlangan bo‘ladi. Satr (qator, String) – bu qandaydir belgilar ketma-ketligi hisoblanadi. Satr bitta, bo‘sh yoki bir nechta belgilar birlashmasidan tashkil topgan bo‘lishi mumkin. C dasturlash tilida satr 0 dan tortib to 255 tagacha uzunlikda ega bo‘ladi. Agarda

o‘zgaruvchilar satr toifasiga tegishli bo‘lsa, u holda o‘zgaruvchi toifasi yozilayotganda 2 xil ko‘rinishda char tipi yoki String deb olinadi.

Bu yerda belgili toifadagi amallar:

a) O‘zlashtirish;

b) Taqqoslash;

Massivlar, vektorlar va ular ustida amallar bajarish

Massivlar. Massiv – bu bir xil toifadagi elementlarning tartibli ketma-ketligidan iboratdir. Massiv bir yoki ikki o‘lchovli bo‘ladi. Bir o‘lchovli massivlar C dasturlash tilida quyidagi ko‘rinishda e’lon beriladi:

<toifasi><massiv_nomi>[elementlar_soni];

Masalan, int d[23];

Ikki o‘lchovli massivlar esa quyidagicha yoziladi:

<toifasi><massiv_nomi>[qatorlar_soni] [ustunlar_soni];

Masalan, int d[2][3];

1-misol. Massivni elementlarini yig‘indisini toping. Misolni yechilishi quyidagicha:

```
#include<iostream>                    Natija:
using std::cout;
#include<conio.h>                     S  3
main()  {
float s,j;
 int i;  s 0;
float d[5]{2,3,4,-1,-5};
for  (i0;  i<5;  i)  s  d[i];
cout<<"\n s <<s;
        getch();
          return 0;
       }
```

Vektorlar. Vektor – bu bir xil toifadagi elementlarning tartibli ketma-ketligidan iborat bo‘lib, bunda massivdan farqi, uning o‘lchami dastur bajarilishi mobaynida o‘zgarishi mumkin. Ya’ni, vektor – bu dinamik massivlar qatori hisoblanadi.

Vektorlarni turli xil ko‘rinishda initsializatsiyalash usullari mavjud: vector<int> vec0; // ochiq vektor

const int size 8;

const int value 1024;

vector<int> vec1(size);//8 ta elementga ega bo‘lgan,elementlar qiymatlari esa 0 ga teng.

vector<int>vec2 (size,value);

//8 ta elementga ega bo‘lib, elementlar qiymatlari 1024 ga barobardir.

inta[4] { 0, 1, 1, 2 };

vector<int>vec3(a, a4);//4 ta elementga ega bo‘lib,elementlar qiymatlari a massivinikiga tengdir.

vector<int> vec4(vec2);//vektor 2 ning nusxasidir.

Vektorlarning massivga nisbatan oladigan bo‘lsak, ularning qulayligi quyidagilar orqali bayon qilinadi:

- Vektorlar bilan ishlashda <vector> standart kutubxona funksiyalari mavjud, unda turli amallarni siklsiz, bir qator kod yordamida bajarish imkoniyati mavjuddir. Bularning ayrimlari keltirib o‘tiladi:

• test.at(i) - test[i] yozuv bilan bir xil, faqat bunda agar i-element mavjud bo‘lmasa, dastur xatolik ko‘rsatmay turadi;

• test. assign(n,m)–vektorga k ta elementni z qiymat bilan yozib oladi;

• test.assign (start,end)–boshqa vektorlarning uchi va oxirini ko‘rsatuvchi iteratorlar startdan end gacha bo‘lgan elementlarni ushbu vektor orqali kiritadi;

• test.front() – birinchi elementga murojaat;

• test.back() – oxirgi elementga murojaat;

• test.begin() – vektor birinchi element iteratori;

• test.end() – vektor oxiri iteratori (bu oxirgi elementidan keyingi adresni koʻrsatadi);

• test.clear() – bu vektorni tozalash;

• test.erase(i) yoki test.erase(start,end) – i-iterator elementini yoki start va end oralig'idagi elementlarni yoʻqotadi;

• test.size() – vektordagi elementlar sonini inobatga oladi;

• test.swap(test2) – test va test 2 vektorlar elementlarini aralashtiradi;

• test.insert(a,b) – test vektoriga iterator koʻrsatayotgan s elementdan oldin d elementni joylashtiradi, bunda iterator kiritilgan elementni yoʻl koʻrsatadi;

• test.insert(a,n,b) – d elementdan s ta kiritadi;

• test.insert(a,start,end) – start va end iteratorlari oralig'idagi elementlarni s dan oldin kiritadi.

- begin() hamda end() iteratorlari mos ravishda xotirada vektorning oxirgi va birinchi elementini oʻzining elementidan keyingi manzilni koʻrsatadi. Bu oʻrinda iterator nimaligiga e'tibor beradigan boʻlsak, unda iterator – bu birorta oʻzgaruvchiga koʻrsatkich boʻlib hisoblanadi. Iterator kerakli qiymatlarni qayerdaligini biladi va uning qiymatini chiqarib bera oladi. Itaratorlar, asosan, konteynerlar bilan ishlashda juda qulaylik tug'diradi. Shu bilan birga, iterator yaratish uchun quyidagini yozish kerak boʻladi:

iterator_nomi<toifa>::iterator nomi;

Masalan: vector <float>::iterator beginvec.begin();

string::iterator end, cur;

Bu yerda satrlar ham konteyner hisoblanadi. Endi iterator koʻrsatayotgan element quyidagicha hisobga olinadi.

cout << * cur << endl;

Bu yerda koʻrinadiki, * belgisi car iteratorni emas, u izlayotgan qiymatni koʻrsatadi.

cur; // keyingi elementga sakrash cur10; // <>curcur10 10 ta elementdan keyingia sakrash

1-misol. Vektor va vektorlar ustida bajariladigan amallar dasturi.

```
#include <vector>
 #include <iostream>
 using namespace std;
 int main(){
 vector<int> d;
   d.push_back (10); d.push_back(11); d.push_back(12);
 vector<int> v;
   for (int i0; i<5; i) {
      v.push_back(i);
   }
   // v vektori elementlari 0 1 2 3 4
 std::vector<int>::iterator it  v.begin()  1;
  // 2-el.dan oldin 33 ni kiritish:
 it  v.insert(it, 33);
   // v vektori elementlari 0 33 1 2 3 4
  //q vektor elementlarini v vector 2-el.dan oldin kiritish:
   v.insert (it, d.begin(), d.end());
   // v vektori elementlari 0 10 11 12 33 1 2 3 4
 it v. begin ()  3;
   // it v vektorning 4-elementini ko‘rsatyapti
   // 4-el.dan oldin 3 marta -1ni joylashtiriladi: v. insert (it, 3, -1);
   // v vektor el.lari 0 10 11 -1 -1 -1 12 33 1 2 3 4 // v vektorning 5-elemnti olib tashlanadi it  v.begin()  4;
   v.erase(it);
   // v vektor el.lari 0 10 11 -1 -1 12 33 1 2 3 4    v.clear();//v vektorni yo‘qotish
  return 0;
 }
```

Yozuvlar, to‘plamlar va ular ustida amallar

To‘plamlar. To‘plam – bu bir xil toifadagi elementlarning tartibsiz joylashuvidir va unda elementlar takrorlanmas ko‘rinishda bo‘ladi. To‘plam ustida bajariladigan amallar quyidagi ko‘rinishlarni oladi: element o‘chirish, kiritish, elementlar sonini aniqlash, bo‘shliqqa tekshirishdan iborat.

To‘plam C dasturlash tilida quyidagicha e’lon qilinadi:

set<int> s;

for(int i 1; i < 100; i) {

s.insert(i);

// to‘plamga element joylash }

s.insert(42);

// 42 to‘plamda mavjudligi sababli hech qanday holat yuz bermaydi

for(int i 2;

i < 100; i 2) { s.remove(i);

// juft sonlar yo‘qotiladi }

// set::size() funksiyasi unsigned int toifasida qiymat qaytargani uchun u int toifasiga almashtiriladi int N int(s.size());

// N50

To‘plamda push_back() funksiyasi yo‘qligi sababli unda elementlar tartibsiz va indeks degan tushuncha mutlaqo yo‘q hisoblanadi. Shu sababli to‘plam elementlarini iterator bilan chiqarsak bo‘ladi. set<int> S;

...

// S to‘plam elementlari yig‘indisi hisoblanadi intr0; for(set<int>::const_iterator it S.begin();

it ! S.end();

it) r(*it);

Bu yerda to‘plam bilan ishlashning eng afzalligi – bu uning tezligidir. Ayniqsa, qidiruvda. Set::find() funksiyasi bitta ta argumentga ega, shuning

uchun uning qaytaradigan qiymati yoki topilgan elementi ko‘rsatiladi, yoki end() iteratoriga teng deb olinadi.

```
set<int> s;
...
if(s.find(42) ! s.end()) {
 // 42 borligi mavjud
} else {
 // 42 borligi mavjud emas  }
```

To‘plamdan elementni yo‘qotish uchun erase() funksiyasidan foydalaniladi.

```
set<int> s;
...
s.insert(54);
...
s.erase(29);
s.erase(s.find(57));
```

Bu funksiyaning joylashuv oralig‘li ko‘rinishi ham mavjud bo‘lib:

```
set<int> s;
...
set<int>::iterator it1, it2;
it1  s.find(10);
it2  s.find(100);
if(...) { s.erase(it1, it2); // 10 dan 100 gacha bo‘lgan elementlar //yo‘q qilinadi(10,100 /yo‘q qilinadi) } else {
   // it2 iteratorni bitta elementi keyin so‘raladi
  // set::iterator uchun  operatori ishlatilmaydi, lekin  va –      //qo‘llanilishi mumkin
it2;
s.erase(it1, it2);    }
```

Bunda to‘plamni e’lon qilishning xuddi vektordagi kabi oraliqli konstruktorlari ham mavjud bo‘lib:

int data[5] { 5, 1, 4, 2, 3 };

set<int> S(data, data5);

Yozuv hamda jadvallar. **Yozuv** – bu turli toifadagi ma’lumotlarning tartibli ketma-ketligidan iboratdir. Bu yozuv maydonlardan tashkil topgan bo‘ladi. Har bir maydon o‘z nomi va o‘z toifasiga ega bo‘lib, ular xotirada ketma-ket joylashgan bo‘ladi. Bunda yozuv uchun ajratiladigan xotira hajmi uning maydonlariga ajratilgan xotira hajmlariga hamda yig‘indisidan iborat bo‘ladi. Yozuvlar C dasturlash tilida quyidagicha e’loni ma’lum qilinadi.

struct Guruh{ int t_r;

char fio[30];

float bali;

} talaba1, talaba2;

Bu yerda Guruh nomli nostandart toifa yaratiladi, bu toifaga tegishli ikkita yozuv e’lon qilib boriladi. Shu yozuvlarga tegishli maydonlarga murojaat quyidagicha ko‘rinishda amalga oshiriladi:

talaba1.fio “Jamshid”;

talaba 2.bali1.5;

Bu yerda guruh toifasida ikkita yozuv ochiladi. Agar bu toifaga tegishli massiv ochiladigan bo‘lsa, yozuvlar massividan jadvallar tashkil topadi. Ikkita yozuvni taqqoslash va o‘zlashtirish ham mumkin bo‘ladi, bunda ularning mos maydonlari taqqoslanadi yoki o‘zlashtiriladi. Bunda maydonlarni alohida yozib ko‘rsatish shart ham bo‘lmay qoladi.

Ko‘rsatkichli turlar

Ko‘rsatkichli toifa (Pointer). Ko‘rsatkichli toifalar ma’lumotlarni ko‘rsatkichlarini yoki manzillari to‘plamini namoyon qiladi, ya’ni ko‘rsatkichlar ma’lumotlarni emas, balki bu ma’lumotlar joylashgan xotiradagi manzilni o‘z ichiga qamrab oladi. Ko‘rsatkichlar xotirada bor yo‘g‘i to‘rt bayt joyni egallab

qo‘yadi. Lekin, u ko‘rsatayotgan ma’lumotlar xotiradan ancha katta maydonni egallagan bo‘lishi ham mumkin. Shu damda pointer ma’lumotini ixtiyoriy boshqa biror ma’lumot yoki ma’lumotlar guruhiga yo‘naltirilgan bo‘ladi. Bundan tashqari, ushbu ko‘rsatkich orqali kerakli ma’lumotga murojaatni amalga oshirish mumkin.

Pointer toifasidagi ma’lumotlarni qiymatlar to‘plamiga bitta maxsus qiymat hosil bo‘lib, uni o‘zlashtirish hech qayerga yo‘naltirilmaganligini ko‘rsatib o‘tadi, bu esa nol yoki bo‘sh ko‘rsatkich hisoblanadi. Masalan, C dasturlash tilida bunday qiymat sifatida NULLdan foydalanib olinadi. Bunday ko‘rsatkichlar ustida amallar quyidagicha ko‘rinishda bo‘lishi mumkin: Bundan ko‘rinadiki, biror-bir ko‘rsatkichga boshqa ko‘rsatkich qiymatini o‘zlashtirish mumkin yoki boshqa ma’lumot egallab turgan xotira sohasi manzilini o‘zlashtirish ham mumkin.

Ko‘rsatkichlar o‘zaro bir-biri bilan bog‘langan ma’lumotlar tuzilmasini yaratishga va ular bilan qayta ishlashda katta ahamiyatga ega bo‘ladi. Xotirada ko‘rsatkichlarni ifodalash uchun asosan, dasturlash tiliga mos ravishda manzilni eng yuqori uzunligicha joy olinadi.

Ko‘rsatkichlarni qiymati musbat butun sonlar sifatida sohada bitlarni ketma-ketligi ko‘rinishida saqlab boriladi. C dasturlash tilida ko‘rsatkichli o‘zgaruvchilarni e’lon qilish uchun ularning toifasini aniqlash kerak bo‘ladi. Buning uchun ko‘rsatkich xotirada qanday toifadagi ma’lumotlarni ko‘rsatayotgan bo‘lsa, ko‘rsatkichli o‘zgaruvchiga ham xuddi shunday toifadagi ma’lumotlarni beradi.

```
int a9;
int *p&a;
 float f4.6;
 float *d&f;
FILE*ffopen(“tolib.txt”,’r’);
```

Mavzuga oid test savollari

1.Turli tipdagi ma’lumot maydonlardan iborat tartibli tuzilmasi –bu?

a) Jadval

b) Massiv

c) Yozuv

d) To‘plam

2. Ma’lumotlar tuzilmasini matematikada qanday ifodalash mumkin?

a) S ={D,R}

b) G={V,E}

c) A={D(1..n)}

d) BT = {K,L,R}

3. C dasturlash tilida o‘zgaruvchilarni e’lon qilinganlardan qaysi biri massiv tuzilmasini anglatadi?

a) int A[100]; struct{ int P1,P2;

b) float P3; } A;

struct{ int P1, P2;

c) float P3; } A[100];

d) int A;

4. C dasturlash tilida o‘zgaruvchilarni e’lon qilinganlardan qaysi biri yozuv tuzilmasini anglatadi?

struct{

int P1,P2;

a) float P3; } A;

b) int A[100]; struct{ int P1, P2;

c) float P3; } A[100];

d) int A;

5. C dasturlash tilida o‘zgaruvchilarni e’lon qilinganlardan qaysi biri jadval tuzilmasini anglatadi?

struct{

int P1, P2;

a) float P3; } A[100];

b) int A[100]; struct{ int P1,P2;

c) float P3; } A;

d) int A;

Nazorat savollari

1. Qanday ma’lumotlar dinamik yoki statik turdagi ma’lumotlar tuzilmasi deyiladi?

2. Ma’lumotlarning qanday toifalarini bilasiz, ularni aytib o‘ting.

3. Butun toifadagi ma’lumotlar ustida qanday amallarni bajarish mumkin?

4. Ma’lumotlarning bul toifasida qanday amallar mavjud, ularni sanab bering?

5. CHAR toifasining tuzilmasi qanday? Belgili toifadan qanday amallarni bajarish mumkin?

6. Ko‘rsatkichli toifa ma’lumoti yordamida nimani hisoblash mumkin?

7. Ma’lumotlarning sanaladigan toifasi deganda nimani tushunasiz?

8. Struktura toifasi qanday beriladi?

Xulosa

Ma’lumotlarni bir necha turlar orqali tasvirlashi mumkin. Masalan, matnli, grafikli, jadvalli, rasmli, audio, video va boshqalar. Bularni barchasini kompyuter yordamida qayta ishlash imkoniyati mavjud. Shunday ekan, hozirgi vaqtdagi bu turdagi ma’lumotlarni ko‘pligini tasavvur qilish qiyin, aksincha, ular ustida ishlash juda tez va oson kechdi. Bularning barchasiga bu turdagi ma’lumotlarning turiga qarab ajratilgani juda katta yordam beradi. Bu yerda ma’lumotlar butun, mantiqiy, belgili, ko‘rsatkichli bo‘lishi mumkin. Ularning qanday bo‘lishidan qat’iy nazar ma’lumotlar tuzilmasi va algoritmlari fani yordamida bunday ma’lumotlarni tezda qayta ishlash mumkin.

2-§. REKURSIYA VA UNI DASTURLASHDA QO‘LLASH

2.1. Rekursiya va uni dasturlashda qo‘llash. Rekursiv algoritmlar, ularning tahlili. Rekursiyaga doir misollar

Reja:

1. Rekursiya haqida tushuncha.
2. Funksiya murojaatlari va rekursiyani tarqalishi.
3. Rekursiv murojaat anatomiyasi.
4. Dumli rekursiya.
5. Dumsiz rekursiya.

Tayanch iboralar: rekursiya, funksiya, dumli rekursiya, dumsiz rekursiya, dinamik aloqa, aktivatsiya, faolli rekord.

Rekursiya haqida tushuncha

Yangi ob’ekt yoki tushunchalarga aniq izohlar kiritishning eng asosiy qoidalaridan biri – bu uning ta’riflarida sizga avvaldan ma’lum va tushunarli bo‘lgan atamalardan foydalanib, ta’rif berishdan iboratdir. Shuning uchun ta’riflarda, asosan, o‘z doirasidan chetda bo‘lgan so‘zlarni qo‘llash noto‘g‘ri talqinga olib keladi.

Boshqa tomondan qaraganda, unda o‘z-o‘zini izohlovchi dasturiy tushunchalar juda ham ko‘pdir. Bunday ta’riflar rekursiv ta’riflar deb ataladi va asosan, cheksiz to‘plamlarga ta’rif berilganda foydalaniladi. Shuning uchun to‘plamga ta’rif berilganda, barcha elementlar ro‘yxatini berish imkonsizdir va juda katta aniq to‘plamlar uchun samarasiz hisoblanadi. Buning uchun eng qulay usul – bu ob’ekt bo‘lib, u o‘sha to‘plamga tegishli yoki tegishli emasligini aniqlashdir. Bu yerda rekursiv ta’riflar ikki qismdan iborat bo‘lib, ular birinchi qismida, to‘plamning asos qilib olingan elementlari joylashtirilsa, ikkinchi qismida esa asos qilib olingan yoki avval foydalanilgan elementlardan foydalanilmagan holda yangi ob’ektlar yaratish uchun qoidalar berib boriladi.

Bu qoidalarga yangi ob’ektlarni yaratishda qayta-qayta murojaat qilib turiladi. Masalan: natural sonlar to‘plamlarini yaratish uchun birinchi asosiy

element, 0, bir tomonlama va bir bo‘yicha inkrementlash jarayoni quyidagicha berib boriladi:

1. 0 ∈ N; 4

2. agar n ∈ N bo‘lsa, keyin (n 1) ∈ N;

3. N to‘plamda boshqa ob’ektlar bo‘lmaydi.

Bu qoidalarga ko‘ra, natural sonlar to‘plami N quyidagi birliklar asosida shakllantirilgan: 0, 0 1, 0 1 1, 0 1 1 1, va h.k. N to‘plam bularni natural sonlar deb ataluvchi ob’ekt deb olsak, ta’rifga ba’zi betartib elementlar ro‘yxati ham joylashadi. Siz murakkab sonlar ustida maxsus spetsifikatsiyadan foydalanib, ba’zi amallar bajarishni tasavvur qila olish juda ham qiyindir. Shuning uchun quyidagi arab raqamlaridan tashkil topgan ma’lum chegarali miqdorlardan foydalanib, izoh keltirish maqbul yechimdir:

1. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ∈ N;

2. agar n ∈ N bo‘lsa, so‘ng n0, n1, n2, n3, n4, n5, n6, n7, n8, n9 ∈ N;

3. bular natural sonlar qatoridan joy olgan.

Demak, N asos qilib olingan noldan to‘qqizgacha bo‘lgan sonlardan tuzilgan kombinatsiyalarda ishlashda mo‘ljallangan sonlarni ham o‘z ichiga qamrab oladi. Rekursiv ta’riflar ko‘pincha, ikki xil maqsadda xizmat ko‘rsatadi: yuqorida qayd etilganidek, yangi elementlar yaratish va tanlangan element shu to‘plamga tegishli yoki tegishli emasligini testdan o‘tkazib oladi.

Testlash jarayonida buni hal qilish murakkab bo‘ladigan bo‘lsa, u son soddaroq ko‘rinishga kelmaguncha bir necha marotaba soddalashtirilib boriladi. Masalan: 456 natural sonmi? N to‘plamni izohlovchi ta’rifdagi ikkinchi holatga asosan, 456 ∈ N bo‘ladi, agar 45∈ N va birinchi holatga asosan, 6 ∈ N bo‘lganda, 45 ∈ N bo‘ladi, agar 4 ∈ N va 5 ∈ N bo‘lganda va ikkalasi ham N ga tegishli bo‘ladi. Berilgan misoldagidek, sonni murakkab ko‘rinishdan soddaroq ko‘rinishga o‘tkaza olish ahamiyatli, shuning uchun tez saralash usulidan foydalanish samarali. Bundan kelib chiqadiki, rekursiv ta’riflar asosan, raqamlar ketma-ketligi va funksiyalarni aniqlashda foydalanib boriladi.

Masalan: faktorial funksiyasini quyidagi ko‘rinishda aniqlanish mumkin.

$$n! = \begin{cases} 1 & if\ n = 0\ (anchor) \\ n * (n-1)! & if\ n > 0\ (inductive\ step) \end{cases}$$

Bu ta’rifdan foydalanib, 1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800, . . . sonlarini 0, 1, 2, . . . , 10, . . . raqamlarinining faktoriallarini o‘z ichiga oluvchi ketma-ketlik hosil qilib olinadi. Keyingi misolda

$$f(n) \begin{cases} 1 & if\ n = 0 \\ f(n-1) + \frac{1}{f(n-1)} & if\ n > 0 \end{cases}$$

ratsional sonlar ketma-ketligini hosil qiluvchi misol ko‘rib chiqiladi.

$$1, 2, \frac{5}{2}, \frac{29}{10}, \frac{941}{290}, \frac{969,581}{272,890}, \ldots\ldots$$

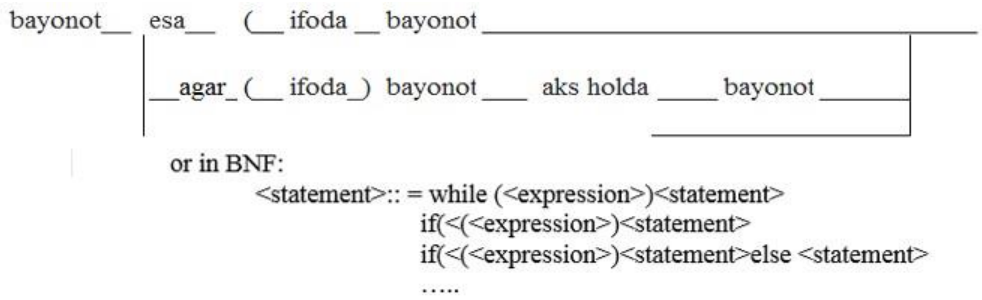
Ketma-ketliklarning rekursiv ta’riflarining bitta yomon xususiyati mavjud bo‘lib, bunda sn ketma-ketlikning elementlari qiymatini hisoblash uchun avvalo, har bir s 1 , . . . , s n–1 larning qiymatlarini hisoblash zarur bo‘ladi. Masalan: 3! Ni hisoblash uchun avval 0!, 1!, 2! larning qiymatini hisoblash kerak bo‘ladi. Bu esa ko‘p hollarda noqulay vaziyatlarga olib keladi. Shuning uchun bu formulaga ekvivalent bo‘lgan formulani topish zarur bo‘ladi.

Umuman olganda, bunday formulani topish murakkab masala hisoblansada, u har doim ham o‘z yechimiga ega bo‘la olmaydi. Lekin, bunday formula rekursiv ta’riflar uchungina mos keladi xolos, chunki, bu misolni soddalashtiradi va butun n ning qiymatini 0, 1, . . . , n – 1 larning qiymatini bilmay turib hisoblash imkoni mavjud bo‘ladi. Misol uchun g ketma-ketlikning ta’rifi:

$$g(n) \begin{cases} 1 & if\ n = 0 \\ 2 * g(n-1) & if\ n > 0 \end{cases}$$

g(n) 2 n ko‘rinishidagi sodda formulaga aylantirish mumkin bo‘lar ekan. Ko‘rilgan tahlillar, rekursiv ta’riflarning faqatgina oddiy ko‘rinishi bo‘lib, bu ta’riflardan matematikada foydalaniladi. Shunday ekan, ko‘rilgan barcha masalalar texnologik sohadandir. Dasturlash tillari grammatikasining maxsus

bo‘limlarida ko‘pincha rekursiv ta’riflarga e’tibor qaratiladi. Grammatika yo blok diagrammalar asosida yoki Backus-Naur Formasida aniq belgilab olinadi. Misol uchun berilgan biror-bir bayonot C dasturlash tilida sintaktik tahlilida quyidagicha blok diagramma orqali tasvirlanishi qulay usul bo‘lib hisoblanadi (1-rasm).



1-rasm. Blok diagramma orqali tasvirlash

Tilning bir elementi hisoblanmish <bayonot(statement)> o‘z-o‘ziga murojaat qilishi bilan rekursiv bajariladi. Bunday ta’riflarda yuqorida keltirilgan turdagi sintaktik qurilma asosida bayonot yoki atamalarning izohini aniqlash imkoniyati tug‘iladi.

Shu bilan bir qatorda, rekursiv ta’riflardan dasturlashda ba’zan foydalaniladi. Funksiyaning rekursiv izohlaridan foydalanish C dasturlash tilida juda yaxshi imkoniyat yaratadi, chunki bu foydalanuvchidan ortiqcha mehnat talab qilmaydi. Oddiy formulada berilgan ta’riflarni C dasturlash tilining sintaksisiga o‘girish ham mumkin. Misol uchun, C dasturlash tilida faktorialga ekvivalent funksiya bo‘lib xizmat qiladi. U quyidagicha:

unsigned int factorial(unsigned int n) { if(n0) return 1; else returnn*factorial(n–1);

}

Muammo juda kritik ko‘rinadi, chunki funksiyada o‘z-o‘zini chaqirishi orqali to‘g‘ri natijani bera olishini imkoniyati ham mavjud bo‘ladi.

Juda ko‘p texnologiyalarda rekursiyalar uchun steklardan foydalanib bajariladi, rekursiyani amalga oshirishning barcha ishini operativ sistemada

amalga oshiradi, buning uchun dastur kodida ba’zi belgilarni kiritish zarurati ham kerak bo‘lmay qoladi.

E.W. Dijkstra tomonidan rekursiyani joriy etishda steklardan foydalanish g‘oyasi ilgari surilgan. Rekursiyani va uning qanday ishlashini yaxshiroq tushunish uchun funksiyaga murojaatlar jarayonini o‘rganish va sistema tomonidan amalga oshirilayotgan operatsiyalarni kuzatish eng ajoyib usuldir.

Funksiya murojaatlari va rekursiyani tarqalishi

Eng avvalo, funksiyaga murojaat qilinganda nimalar sodir bo‘ladi? Bu funksiya formal parametrlarga ega bo‘lsa, amaldagi aktual parametrlar keng tarmoqliligiga o‘zgartirilishiga majbur bo‘ladi. Bundan tashqari, sistema dasturi yakuniy exe variant ijrosini qayerga saqlab davom ettirishi kerakligini bilishi zarur bo‘lib qoladi. Bu yerda funksiyalar boshqa nom bilan yoki asosiy dastur (the function main()) bilan ham chaqirilishi mumkin.

Funksiyani qayerdan chaqirib olish esa sistema tomonidan saqlab qolinishi zarur bo‘ladi. Bu qaytish manzillarni asosiy xotirada saqlab borish orqali amalga oshirilishi mumkin bo‘ladi, lekin qancha joy kerakligi noma’lum bo‘lsada, buning uchun ko‘p ortiqcha joy ajratib ketish ham yaxshi vaziyatlarga olib kelmaydi.

Funksiyaga murojaatlarda esa manzil qaytarishga qaraganda ko‘proq ma’lumotlar saqlab qolinishi zarurdir. Shuning uchun steklardan foydalanib, dinamik joylashtirish yaxshi natijalarga olib kela boshlaydi.

Shu o‘rinda aytish kerakki, funksiyaga murojaatda qanday ma’lumotlar saqlanib qolishi kerak? Buni esa birinchidan, lokal o‘zgaruvchilar to‘planishi orqali topib olinadi.

Agar x lokal o‘zgaruvchisining e’loni mavjud bo‘lgan f(1) funksiya, x o‘zgaruvchini lokal e’lon qiluvchi f(2) funksiyaga murojaat qilsa, sistema bu ikki x o‘zgaruvchilarni farqlay olishi zarurdir. Agar f2() x o‘zgaruvchini ishlatsa, unda o‘z x ni o‘rnini bosadi; agar f2() x ga qiymat belgilasa, unda f(1) ga tegishli bo‘lgan x o‘zgarmasdan qoldirilishi kerak bo‘ladi. Qachonki, f(2)

tugatilganda, f1() f(2)ga murojaat boʻlinmasidan oldin xususiy x ga oʻzlashtirilgan qiymatdan foydalanishi mavjud boʻladi. Bu hozirgi koʻrilayotgan funksiya oʻz-oʻziga rekursiv murojaat qilgandagi qismda, f(1) f(2)dek bir xil funksiya boʻlganda muhim hisoblanadi.

Sistema bu ikki x oʻzgaruvchilarni qanday ajratadi? main() ni oʻz ichiga oluvchi har bir funksiyaning holati ularning tarkibidagi barcha mahalliy oʻzgaruvchilar, funksiya parametrlarining qiymatlari va funksiyaga murojaat qayerdan boshlanishi kerakligini koʻrsatuvchi adres indikatorlari orqali moslashtiriladi. Shunday ekan, shu ma’lumotlarni oʻzida saqlovchi ma’lumotlar maydoni aktivatsiya hisoboti yoki stek ramkasi deb ataladi va stekda saqlanadi.

Aktivatsiya hisobotida ancha vaqtgacha funksiya oʻzining amal doirasini yoʻqotmagan holda saqlab boradi. Bu hisobot – funksiyaning xususiy umumiy birlashtirilgan ma’lumotlar joyi boʻlib, unda dasturlarning yaxshi ishga tushishi uchun zarur boʻlgan barcha ma’lumotlar saqlanadi. Bunda aktivatsiya ma’lumotlari odatda, juda koʻp saqlanmaydi, chunki ular funksiya ishga tushirilganda dinamik holatda joylashib oladi va funksiyadan chiqilganda joylashuvlardan uzoqlashadi. Faqatgina main() funksiyasidagi ma’lumotlar ancha vaqt saqlanib turadi.

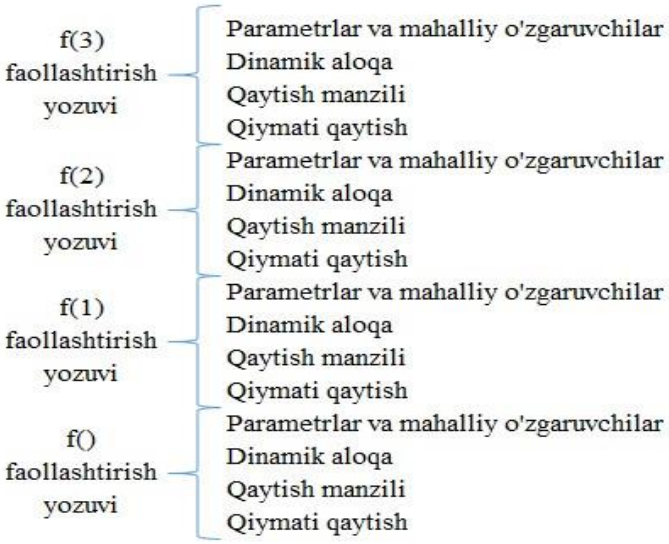
Odatda aktivatsiya hisoboti quyidagi ma’lumotlarni oʻzida saqlab qoʻyadi:

- Funksiyaning barcha parametrlari uchun qiymatlar, massiv yacheykalari joylashgan manzillarni va oʻzgaruvchilarni saqlaydi va barcha boshqa ma’lumotlar bandlaridan oʻziga kerakli nusxa olib qoʻyadi.
- Har qanday holda har qayerda saqlanishi mumkin boʻlgan mahalliy oʻzgaruvchilarning qayerda saqlanishini koʻrsatuvchi deskriptor va koʻrsatkichlarini saqlab qoʻyadi.
- Murojaat etuvchining manzilini va joriy murojaatlar holatini e’tiborga oladi.
- Murojaat koʻrsatkichlarining dinamik aloqalarini ta’minlab beradi.

- Funksiya qaytargan qiymatlar void sifatida e’lon qilinmaydi. Bu yerda aktivatsiya jarayoni hajmi bir murojaatdan boshqasiga farq qilishi mumkin, qaytarilgan qiymat murojaat aktivatsiyasining o‘ng tomonida joylashib boradi.

Agar funksiya main() yoki boshqa funksiya orqali atalgan bo‘lsada, uning aktivatsiya rekordi ishga tushirish vaqti stekida yaratiladi. Stek doimo funksiyaning joriy holatiga ta’sir etadi. Masalan: Agar f3() ishga tushirilayotgan vaqtda uning ishga tushirish vaqti steki chizmada ko‘rsatilgan bo‘lsa, main() f1() ni, f1 () f2 ()ni, f2 () esa o‘z navbatida f3 () ni chaqiradi, - deb tasavvur qilish mumkin. Agarda stek xususiyatiga ko‘ra, f3 () uchun faollashtirish rekordi f3 ()ning qaytaruvchi qiymatining yonida stek ko‘rsatkichini o‘zgartirish orqali amalga oshirilsa, u holda darhol keyingisiga o‘tish orqali f2 () ni ijrosi amalga oshiriladi va shu bilan birga, hozirda ma’lumotlar xususiy omborini uning ijrosini qaytarish uchin aktivlashtirishga muhim bo‘lgan bepul kirib chiqishga ruxsat etiladi.

Boshqa tomondan, agar f3() boshqa f4() funksiyani chaqirsa, keyin ishga tushirish vaqti steki o‘zining darajasini ko‘paytirib boradi, chunki f4() uchun faollashtirish rekordi stekda amalga oshiriladi va f3()ning faolligi saqlab qolinmaydi, aksincha oshiriladi. Stekni ishga tushirish vaqti main() f1()ni, f1() f2()ni, f2() f3() ni quyidagi takibiy qismdek bo‘ladi:



2-rasm. Stekni ishga tushirish vaqti

Aktivatsiya, ya’ni faollik rekordini yaratish funksiyaga murojaat qilinganda sistemaga rekursiyani o‘z holatida saqlab turishiga imkoniyat beradi. Bu yerda qachonki, bir xildagi murojaat mavjud bo‘lganda rekursiya funksiyaga murojaat qila boshlaydi.

Shuning uchun rekursiv murojaat funksiyaning o‘ziga murojaat qilish degani emas, balki bir xilda ko‘rinishdagi uchrashi mumkin bo‘lgan funksiyalarga murojaat deganidir. Bular har xil faollik rekordlarida namoyon bo‘ladi va sistema ularni farqlab bilib oladi.

Rekursiv murojaat anatomiyasi

Ixtiyoriy o‘suvchi x ning musbat butun n-darajasini aniqlovchi funksiya rekursiv funksiyaga bemalol misol bo‘ladi. Uni aniqlovchi funksiyasi quyidagicha:

$$x^n \begin{cases} 1 & if\ n = 0 \\ x * x^{n-1} & if\ n > 0 \end{cases}$$

X ning n-darajasini hisoblovchi funksiya C dasturlash tilida quyidagi pow orqali to‘g‘ridan-to‘g‘ri hisoblanishi mumkin:

```
/* 102 */ double power (double x, unsigned int n) {
/* 103 * /              if ( n    0 )
/* 104 * /              return 1.0;
// else
/ * 105 * /    return x * power (x, n-1); }
```

Bundan foydalanib, x ning to‘rtinchi darajasining qiymati quyidagicha yo‘nalishda hisoblanishi mumkin:

x^4x*x^3x*(x*x^2)x*(x*(x*x^1))x*(x*(x*(x*x^0)))

x*(x*(x*(x*1)))x*(x*(x*(x)))x*(x*(x*x)) x*(x*x*x)x*x*x*x

Bunday ketma-ketlik rekursiv murojaatlar zanjirining oxirgi bosqichiga olib boradi. Oxirgi bosqichida natija daraja 0 bo‘lganda 1 qiymatni beradi; shu bilan natija undan oldingi murojaatlarga o‘tkaziladi.

Ijrosi amalga oshayotgan murojaat x*1x degan natija beradi. Keyin esa bu raqam ikkinchi murojaaat uchun x*x orqali shunaqa natijani qaytara boshlaydi.

Bunda oxiri x * x * x natijasiga olib keladi.

call 1 x 4 x · x 3 x · x · x · x

call 2 x · x 2 x · x · x

call 3 x · x 1 x · x

call 4 x · x 0 x · 1 x

call 5 1or alternatively, as

call 1 power(x,4)

call 2 power(x,3)

call 3 power(x,2)

call 4 power(x,1)

call 5 power(x,0)

call 5 1

call 4 x

call 3 x · x

call 2 x · x · x

call 1 x · x · x · x

Funksiyalar ijrosi amalga oshayotganda sitema qanday operatsiyalarni amalga oshiradi? Bunda ma’lumki, sistema amalga oshirish vaqti stekida barcha murojaatlar izlarini saqlab turadi.

Funksiya ijrosidan so‘ng, ijro exe si qayerda saqlanganligini bilish u uchun zarurat tug‘diradi. Buning uchun misol qilib, 102 yoki 105 raqamlari kiritilgandagi power() funksiyasi bir chiziqda joylashadigan bo‘lsin.

```
Int main()
{…
/* 136 * / y  power(5.6, 2) ;
….
}
```

Rekursiv murojaatlar izi – bu diagrammada ko‘rsatilganidek, oddiy holatda amalga oshiriladi.

call 1 power(5.6,2)

call 2 power(5.6,1)

call 3 power(5.6,0)

call 3 1

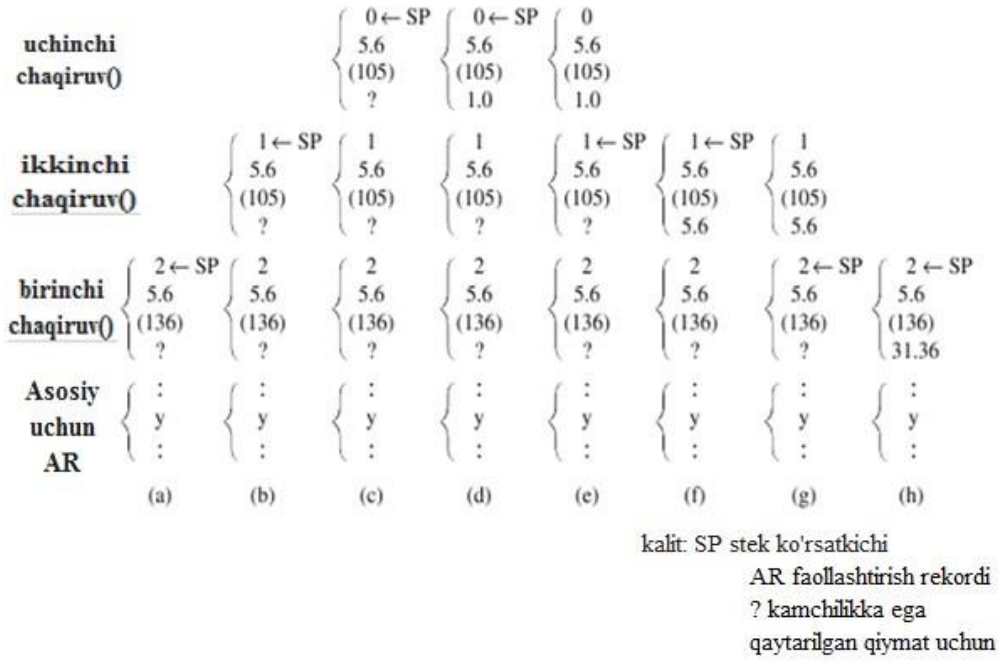
call 2 5.6

call 1 31.36

Chunki, ko‘pgina operatsiyalar amalga oshirish vaqti stek orqali amalga oshiriladi. Funksiya 1-marotaba ishga tushirilganda, 4 ta belgi orqali stekida suriladi va xotiradan joy egallaydi: qaytuvchi manzil esa 136, aktual parametrlar 5.6 va 2, va 1 ta power() orqali qaytarilgan qiymat uchun joyni aniqlovchi ishga tushadi.

Yuqorida hozir power() daraja()) funksiyasi amalga oshirilgan holat ko‘rildi. Power(5.6,2) darajasi orqali amalga oshirish jarayonini ko‘rish uchun vaqt stekidagi o‘zgarishlar ko‘zdan kechiriladi.

Buning uchun Power(5.6,2) darajasini ijrosi bilan tanishib chiqiladi.



3-rasm. Power(5.6,2) darajasini ijrosi

Bu jarayonga e’tibor beradigan bo‘lsak, bu juda qisqa vaqtda amalga oshmaydi. Chunki, sistema power(5.6,1) ning qiymatini aniq qaytarmaydi; avval yuqoridagi amal hisoblanishi lozim.

Shuning uchun power() ga 5.6 va 1 argumentlari orqali qayta murojaat beriladi. Ammo, bu murojaat amalga oshirmasidan avval, ishga tushirish vaqt steki yangi belgilarni qabul qila boshlaydi.

Shundan so‘ng, 2-argument 0 yoki 0 emasligi tekshiriladi. Bu 1 ga tengligi sabab, bu safar 5.6 va 1 argumentlari bilan power() 3-marotaba chaqiriladi.

Funksiya ijrosidan keyin sistema argumentlarni eslab qoladi va ularni steklarga joylashtirib, ularning manzillarini eslab qoladi, shunday bo‘lsa-da, u bitta yacheykani natija uchun saqlab qo‘yadi.

Chizma 5.2 c da ham shu jarayonlar davomini ko‘rish mumkin. Bu yerda natija saqlanadigan stek yacheykasi adashib ketmasligi eng muhim hisoblanadi. SP har bosqichda oshib boraveradi.

Endi power()ga 2-murojaat yakunlanishni olib boradi, chunki bu power(5.6; 1) ning natijasini olib boradi. Bu natija, 1.0, 5.6 ga ko‘paytiriladi va natija maydonida saqlab qo‘yiladi.

Chizma 5.2 f SP ning qiymati o‘zgarmasdan oldingi stekning holatini ko‘rsatadi va chizma 5.2 g stekning bu o‘zgarishdan keyingi holatini ko‘rsatib turadi. Shu o‘rinda, power() 1-murojaat natijasini 2-murojaat natijasiga ko‘paytirib tugallanishi mumkin bo‘ladi.

1-argumenti orqali yana 5.6. va y ga o‘zlashtiriladigan natijaviy qiymat 31.36 ga yetkaziladi. Bundan oldingi stek holati chizma 5.2 h dagidek bo‘ladi. Power() funksiyasi hech qanday rekursiyasiz quyidagicha boshqacha tarzda bajariladi:

```
double nonRecPower(double x, unsigned int n) { doubleresult1;
 for(resultx;n>1; --n)  result *x;
        returnresult; }
```

Dumli rekursiya

Barcha rekursiyalar uchun aniqlanayotgan to‘plamlar yoki funksiyalarning ma’lumotlari o‘zida jamlanishi kerak. Bunaqa ma’lumotlarning juda ko‘p turlari joriy etilishi muhimdir. Bu ma’lumotlardan ko‘p marotaba, turli yo‘nalishlarda foydalanish mumkin. Rekursiyaning turli darajalari mavjud bo‘lib, ularning murakkablik darajalari ham har xil ko‘rinishda bo‘ladi.

Quyida bunday turlarning ba’zilari muhokama qilinadi. Ularning oddiysi – dumli rekursiyadir. Dumli rekursiya faqatgina bitta rekursiv murojaatni funksiya oxirida qo‘llashi mumkin va uni xarakterlaydi. Boshqacha qilib aytganda, murojaat bo‘lganda, funksiya orqali amalga oshirilishi kerak bo‘lgan birorta so‘zni qoldirishga imkon bermaydi.

Rekursiv murojaat eng so‘nggi murojaat emas, balki unda to‘g‘ridan-to‘g‘ri yoki o‘zlashtirmaganlari ham mavjud bo‘lishi mumkin. Misol uchun, tail() funksiyasi quyidagicha aniqlanadi:

```
void tail(int i){
        if(i>0){cout<<i<<‘‘;
tail(i-1);}
}
```

tail (dumli) rekursiya li funksiyaga misol, bundagi nonTail funksiyasi quyidagicha izohlanib o‘tiladi.

```
 void nonTail(int i) {
if (i > 0) {
nonTail(i-1);
 cout << i << ‘‘;
 nonTail(i-1);
}  }
```

Dumli rekursiyani bajarilishi oddiygina va biror foydalanuvchi tomonidan osonlikcha joylashtirilishi mumkin. Buni quyidagi I o‘zgaruvchining qiymatini unga bo‘lgan rekursiv murojaatlar darajasi orqali kamaytirib borish misolida

ko‘rish ham mumkin. Quyida tail() iterativ funksiya orqali ifodalanishi ko‘rib chiqiladi:

```
void iterativeEquivalentOfTail(int i) {
 for ( ; i > 0; i--)
 cout << i << ‘‘; }
```

Dumli rekursiyani iteratsiya orqali foydalanishning afzalligi shundan iboratki, bunda ko‘p tillar uchun solishtiriladigan hech qanday foydali tomonlari bo‘lmasligi mumkinligi hamda Prolog kabi tillarda bu juda katta ahamiyatga ega ekanligidir. If goto bilan birga qo‘llaniladigan hollarda dumli rekursiyadan foydalanmaslik eng maqbul yechimdir.

Dumsiz rekursiya

Rekursiyalarda mavjud boshqa muammo qiymat kiritish qatorlarini teskari tartibda chiqarib berishdan iboratdir. Quyida oddiy rekursiyani qo‘llanilishi ko‘rsatib o‘tilgan:

```
/* 200 */          void reverse() {
                       char ch;
/* 201 */            cin.get(ch);
/* 202 */           if (ch ! ‘\n’) {
/* 203 */            reverse();
/* 204 */           cout.put(ch);
                                  } }
```

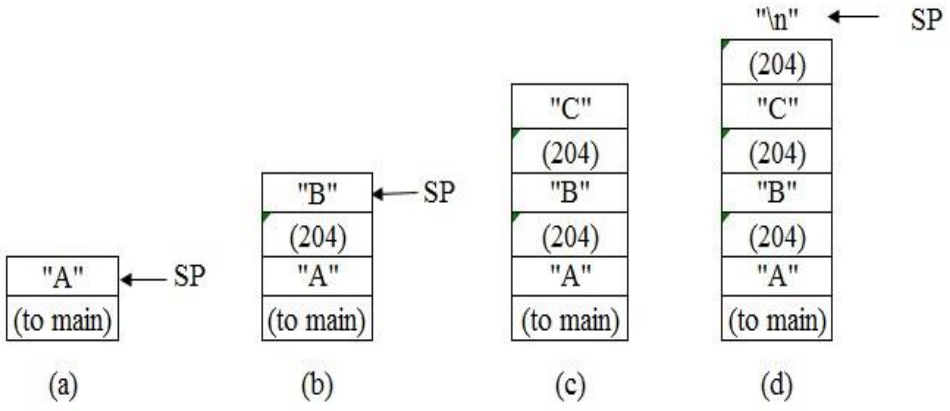
Bu funksiya o‘z-o‘zidan biror amalni bajarmaydi, lekin bu rekursiyaning ahamiyatliligi bilan main() ning reverse() ko‘rinishidagi turi va kiritish “ABC” orqali amalga oshiradi. Avvalo, buni faollashtirish yacheykalar orqali ch o‘zgaruvchi uchun amalga oshiriladi va manzilni qaytaradi.

Funksiya nomi yonida void ishlatilsa, funksiya hech qanaqa qiymat qaytarmaydi. Shuning uchun natija yacheykasi uchun ham hech qanaqa

almashtirishlar amalga oshirilmaydi. Bunda get() funksiyasini birinchi belgisi sifatida “A” ni o‘qiydi.

Chizma 5.3 a birinchi marotaba reverse() o‘z-o‘ziga murojaat qilgandagidek ishga tushiriladi.

4-rasmda reverse() amalga oshirilganda, ishga tushirish vaqt stekidagi o‘zgarishlar ko‘rsatilgan.



4-rasm. reverse() amalga oshirilganda ishga tushirish vaqt stekidagi o‘zgarishlar

Ikkinchi belgi o‘qib olinadi va so‘nggi belgi ekanligiga tekshirilib olindi hamda unday bo‘lmasa, reverse() ga qayta murojaat etiladi. Lekin, har qanaqa vaziyatda ham ch ning qiymati qaytarilgan qiymat bo‘yicha ishga tushirish vaqti stekga joylashtiriladi. Reverse 3-marotaba chaqirilmasidan oldin stekda ikki yoki undan ortiq belgilar mavjudligiga tekshiradi, keyingi urinishlarda ko‘rsatilgan belgiga yetmaguncha funksiyaga qayta-qayta murojaatlar bo‘laveradi.

Bu misolda, reverse() ga 4 marta murojaat etilgan va so‘nggi murojaatgacha bo‘lgan o‘zgarishlar va jarayonlar 5.3 d da namoyon etilgan. To‘rtinchi murojaatda, get() so‘nggi belgini topadi va reverse() ishni yakunlaydi.

Sistema faollashtirish natijalaridan uning manzilini qaytaradi va SPni orttirish orqali natijada aniq bitlar yordamida bu rekordni yaroqsiz qilib ishlatilmay qolinadi. Yuqoridagi amallarni chop etish uchun 204 qatordan

boshlanadi. Uchinchi murojaat faollik rekordi faol bo‘lganda, ch ning qiymati, “C” harfi, birinchi belgi sifatida ekranga chop etiladi.

Nihoyat, reverse() ning birinchi murojaatining faollik rekordiga yetib boriladi. Keyin “A” chiqariladi va ekranda ko‘rilishi mumkin bo‘lgan qator “CBA” ko‘rinishida o‘tadi.

Birinchi murojaat butunlay yakunlanadi va dastur ijrosi main() da bo‘ladi.

Quyida bir xil funksiya uchun rekursiv hamda rekursiv bo‘lmagan ijrolarini taqqoslang:

```
void simpleIterativeReverse() {
char stack[80];
 register int top  0;
 cin.getline(stack,80);
 for (top  strlen(stack) - 1; top > 0;
 cout.put(stack[top--]));  }
```

Funksiya ancha qisqa va tushunarsizroq holda bo‘lishi mumkin. Bunda unda qisqalik va soddalikning sababi va belgilardan tashkil topgan satr yoki massivni teskarisiga o‘zgartirmoqchi bo‘lganda ekanligini yodingizda saqlashingiz kerak.

Bu shuni anglatadiki, strlen() va getline() ga o‘xshagan funksiyalardan C standart kutubxonasidan foydalanish ham mumkin. Agar bunaqa funksiyalar mavjud bo‘lmasa, unda iterative funksiya boshqacharoq bajarilishi ham mumkin:

```
void iterativeReverse() {
char stack[80];
 register int top 0;
cin.get(stack[top]);
wxile(stack[top]!’\n’)
cin.get(stack[top]);
for (top - 2; top > 0;
```

cout.put(stack[top--])); }

Wxile ga tegishli getline() va yuqori qiymatlarning avtoinkrementi strlen() o‘rnida kelgan. For esa odatdagidek ko‘rinishda bo‘ladi.

Bu tahlil albatta, aniq teoretik emas, chunki butun sonlarni o‘z ichiga oluvchi kiritish qatorini teskarisiga aylantirish stekning ma’lumotlar tipini char dan int ga o‘zgartirgandan keyin iterativeReverse() dek bir xil ko‘rinishda bajariladi.

Massiv uchun stekda foydalanilgan o‘zgaruvchi nomi tasodifiy emasligini yodga olish kerak. Shunchaki, sistema tomonidan shubhasiz bajarilganlar oydinlashtiriladi.

Stek amalga oshirish vaqti stekidan ko‘proq vaqt talab etadi. Bu muhimdir, chunki, birgina oddiy murojaatlar bunda dumli rekursiyada yetarli bo‘lganidek juda ham ko‘p emas.

Rekersiv versiyadagi put() ham bunda hisobga olinishi zarurdir. IterativeReverse() funksiyasi uchun stek o‘zgaruvchisi mahalliy ekanligini ham yodda saqlash zarurdir. Shunday bo‘lsa-da, agar global tarzda st ob’ekti yaratilishi talab etilsa, unda buning amalda qo‘llanilishini quyidagicha yozish mumkin bo‘ladi:

```
void nonRecursiveReverse()  {
int ch;
cin.get(ch);
wxile (ch ! ‘\n’) {
st.push(ch);
cin.get(ch); }
wxile (!st.empty())
cout.put(st.pop());
}
```

stek <char> st ning funksiyaning tashqaridagi e’loni bilan qaraladigan bo‘lsa, iterativeReverse() ni nonRecursiveReverse()ga taqqoslagandan keyin,

shunday xulosa qilish mumkinki, birinchi versiya yaxshiroq, chunki, u tezroq ishlaydi va hech qanaqa funksiya murojaatlari yo‘q.

Shu bilan birga, funksiya o‘z-o‘zini qoniqtiradi, lekin nonRecursiveReverse() esa har bir iteratsiyada kamida bitta murojaatdan foydalanadi, dastur ishlashi ham sekin bo‘ladi.

Funksiya rekursiv likdan iteraktivversiyaga o‘zgartirilganda, dastur aniqligiga zarar yetishi mumkin va dastur kodining kamyishi yo‘qolishi mumkin. C dagi rekursiv funksiyalarning iteraktiv versiyalari boshqa dasturlash tillaridagi kabi uzun emas, shu sababli dastur qisqaligi ahamiyatga molik bo‘lmasligi ham mumkin.

Xulosa o‘rnida, Helge Von Kochning qor uchquni qurilishi ko‘rib chiqiladi. Egri chiziqlarga 1904-yilda shved matematigi Helge Von Koch tomonidan ishlab chiqilgan noaniq uzunlik va atrofini qurshab olgan aniq maydonga ega bo‘lgan davomiy hamda differensiallanuvchi funksiyani misol qilish mumkin. Bunday egri chiziqlarga qor uchqunlarning noaniq ketma-ketligiligining limiti sifatida qarash mumkin.

Haqiqiy qor uchqunlarida bo‘lganidek, bularning ikkitasi oltita gultojbargli naqshlarga ega, lekin uning algoritmi yengillashtiriladigan bo‘lsa, u uchta oddiy egri chiziqning birlashmasi sifatida qaraladi.

Shunday egri chiziqlarning bittasi quyidagicha izohlanadi:

1. Interval qismini uchta bir xil bo‘laklarga bo‘linishi;
2. Tomonning uchdan bir qismini burchak bo‘yicha harakatlantirish;
3. O‘ngga 60°ga burilgani va tomonning uchdan bir qismini yo‘naltirilgani;
4. Chapga 120°ga burilgani va qolgan uch qismida ham davom ettiring;
5. O‘ngga 60°ga burilgani va tomon bo‘ylab yana chiziq chizing.

Bu besh qadamlarning natijasi kompyuter grafikasida ishlaganda, u juda uzoq masofalarga cho‘zilishga aslo hojat qoldirmaydi, chunki agar chiziqlar piksellarning diametridan kichikroq bo‘lsa, ekranda faqatgina bir nuqta sifatida ko‘rish mumkin bo‘ladi, xolos.

Mavzuga oid test savollari

1. … - ob’ektni mazkur ob’ektga murojaat qilish orqali aniqlashdir?

a) Rekursiya

b) Algoritm

c) Dastur

d) Tuzilma

2. Rekursiya masalasini hal qiluvchi bosqichlari qanday nomlanadi?

a) Rekursiv triada

b) Rekursiv algoritm

c) Rekursiv munosabat

d) Rekursiv ob’ekt

3. Rekursiv triada qaysi bosqichlardan iborat?

a) parametrizasiya, rekursiya bazasi va dekompozitsiya

b) aniqlash, chaqiruv, o‘zgartirish

c) oson, o‘rta, qiyin

d) qo‘shish, ayirish, ko‘paytirish

4. Rekursiv triadaning qaysi bosqichida masala shartini tasniflash va uni hal etish uchun parametrlar aniqlanadi?

a) parametrizasiya

b) rekursiya bazasi

c) dekompozitsiya

d) chaqiruv

Nazorat savollari

1. Rekursiya deganda niani tushunasiz?
2. Rekursiv funksiyalar haqida nimalarni bilasiz?
3. Rekursiv algoritmlar nima? Rekursiv algoritmlar haqida gapiring.
4. Rekursiv ma’lumotlar tuzilmasi nima?

Xulosa

Ma’lumki, har qanday rekursiv funksiyaga murojaat qilinganda, bu funksiya formal parametrlarga ega bo‘lsa, amaldagi aktual parametrlar keng tarmoqliligiga o‘zgartirilishiga majbur bo‘ladi.

Bundan tashqari, sistema dasturi yakuniy exe variant ijrosini qayerga saqlab davom ettirishi kerakligini bilishi zarur bo‘lib qoladi. Bunda funksiyalar boshqa nom bilan ham chaqirilishi mumkin, lekin oldingi exe nusxasi qurilmaning asosiy xotirasida joylashgan bo‘ladi. Rekursiv funksiya shu xususiyati bilan ma’lumotlar ustida ishlanishiga yordam beradi.

2.2. Ma’lumotlarni qidirish algoritmlari. Qidiruv tushunchasi va uning vazifasi. Chiziqli qidiruv. Binar qidiruv. Qidirish usullari samaradorligi va optimallashtirish

Reja:

1. Qidiruv tushunchasi va uning vazifalari.
2. Chiziqli qidiruv tushunchasi va ular ustida amallar bajarish.
3. Binar qidiruv ustida amallar bajarish.
4. Qidirish usullarining samaradorligi va ularni optimallashtirish.

Tayanch iboralar: qidiruv, chiziqli qidiruv, binary qidiruv, optimallashtirish, qidiruv algoritmlari.

Qidiruv tushunchasi va uning vazifalari

Ma’lumki, bugungi kunda axborot texnologiyalari jadal sur’atlar rivojlanayotgan bir vaqtda EHMda ma’lumotlar ustida bir qancha amallar bajarilmoqda. Shulardan biri bu ma’lumotlarni qayta ishlashda qidiruv amali eng asosiy ammallardan biri bo‘lib hisoblanadi. Uning vazifasi berilgan argument bo‘yicha massiv ma’lumotlari ichidan mazkur argumentga mos ma’lumotlarni qidirish va mavjud bo‘lsa, uni topishdan iboratdir.

Ixtiyoriy ma’lumotlar jamlanmasi jadval yoki fayl deb ataladi. Ixtiyoriy ma’lumot yoki element boshqa ma’lumotdan biror-bir belgisi orqali tubdan farq

qiladi. Bunda mazkur belgi esa kalit deb ataladi. Kalit noyob bo‘lishi mumkin, ya’ni mazkur kalitga ega ma’lumot jadvalda yagona bo‘lishi ham mumkin. Bunday noyob kalitga boshlang‘ich kalit deb aytiladi.

Ikkinchi kalit bir jadvalda takrorlansa-da, u orqali ham qidiruvni amalga oshirish mumkin bo‘ladi. Ma’lumotlar kalitini bir joyga yig‘ish yoki yozuv sifatida ifodalab, bu ularni bitta maydonga kalitlarni yozish deganidir. Bu yerda agar kalitlar ma’lumotlar jadvalidan ajratib olinib, alohida fayl sifatida boshqa joyda saqlansa, u holda bunday kalitlar tashqi kalitlar ham deb yuritiladi. Aksincha, ya’ni yozuvning bir maydoni sifatida aynan o‘sha jadvalda saqlansa, u holda bu ichki kalit deb nomlanadi. Eng asosiysi kalitni berilgan argument bilan mosligini aniqlovchi algoritmga berilgan argument bo‘yicha qidiruv deb yuritiladi. Qidiruv algoritmi vazifasi kerakli ma’lumotni jadvalda topib qo‘yish yoki u izlanayotgan belgi yo‘qligini aniqlashdan iborat. Agar kerakli ma’lumot topilmasa, u holda ikkita ishni amalga oshirish mumkin bo‘ladi:

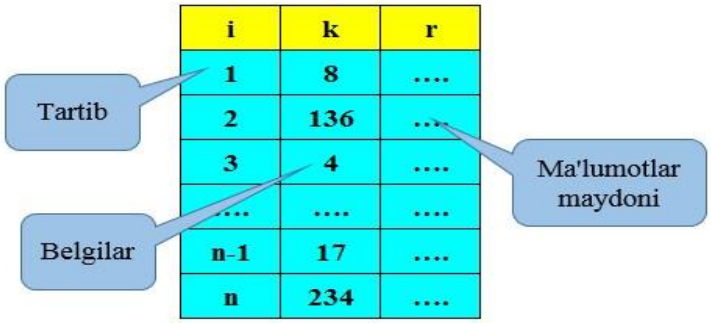
1. Ma’lumot yo‘qligiga ishonch hosil qilib, uni belgilab olish;
2. Jadvalga o‘sha ma’lumotni kerakli joyiga qo‘yish.

Faraz qilaylik, t – kalitlar massivi bo‘lsin. Har bir t(i) uchun y(i) – ma’lumot mavjud bo‘ladi. Key – qidiruv argumentiga e’tibor qaratadigan bo‘lsak, unga rek - informatsion yozuvi mos qo‘yiladi. Shunday ekan, jadvaldagi ma’lumotlarning tuzilmasiga qarab, qidiruvni bir necha turlari mavjudir.

Chiziqli qidiruv tushunchasi va ular ustida amallar bajarish

Mazkur ko‘rinishdagi qidiruvda agar ma’lumotlar tartibsiz yoki ular tuzilishi jihatdan noaniq bo‘lganda, bu usul qo‘llaniladi. Bunday holda qidirish uchun ma’lumotlar butun jadval bo‘yicha operativ xotirada kichik manzillardan boshlab, to katta manzillargacha ketma-ket qarab chiqiladi.

Massivda ketma-ket qidiruvning qulayligi bu (search) o‘zgaruvchi topilgan element raqamini saqlab qoladi.



1-rasm. C dasturlash tilida dastur kodi ko‘rinishi

C dasturlash tilida dastur kodi quyidagi ko‘rinishda bo‘ladi:

```
#include <iostream.h>
#include <conio.h>
int search(int a[], int N, int key)
{
 int i0;
            wxile (i!N)
if (a[i]key) return i;
else i;
 return -1;
}
main ()
{
int i, N, mas[1000], key, P;
cout<<”Masiv uzunligini kiriting!”<<endl;
cin>>N;
cout<<”Massiv elementlarini kiriting!”<<endl;
for (i0; i<N; i)
 cin>>mas[i];
cout<<”Qidiruv elementini kiriting!”<<endl;
 cin>>key;
Psearch(mas,N,key);
if (P-1) cout<<”Bunday element massivda yoq”;
```

else cout<<”Bunday element bor”<<P1;

getch();

return 0;

}

Massivda ketma-ket qidiruv algoritmi samaradorligi bajarilganda taqqoslashlar soni N bilan aniqlash mumkin. V_{min} 1, N_{max} v. Agar ma’lumotlar massiv yacheykasida bir xil ehtimollik bilan taqsimlangan bo‘lsa, u holda N_{sr} (v 1)/2 bo‘ladi. Agar kerakli element jadval ko‘rinishda bo‘lmasa va mazkur elementni jadvalga qo‘shish mumkin bo‘lsa, u holda yuqoridagi dasturning oxirgi ikkita operatori quyidagicha o‘ringa almashtiriladi:

n: n1;

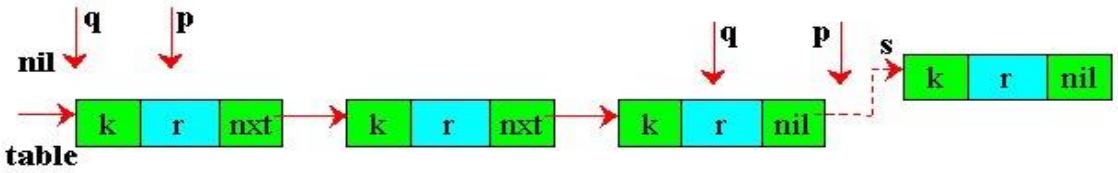
k[n]: key;

r[n]:rec;

search: n;

exit;

Agar ma’lumotlarning jadvali bir bog‘lamli ro‘yxat ko‘rinishida berilgan bo‘lsa, bunday holatda u ketma-ket qidiruv ro‘yxatda amalga tadbiq qilinadi.



2-rasm. Bir bog‘lamli ro‘yxat ko‘rinishi

C dasturlash tilida dasturni yozilishi:

#include <iostream.h>

#include <conio.h>

#include <alloc.h>

#include <stdlib.h>

struct TNode {

int value;

```
  TNode* pnext;
  TNode(int val): pnext(0), value(val) {}
};
//Ro‘yxatga element qo‘shish
void add2list(TNode **pphead, int val) {
 TNode **pp  pphead, *pnew;
  pnew new TNode(val);
  pnew->pnext *pp;
   *pp pnew;
}
//Ro‘yxat elementlarini ekranga chiqarish
void print(TNode *phead) {
  TNode* p  phead;
 wxile(p) {
    cout <<""<< p->value<<"|" <<" "<<"|"<<p<<"|"<< "--> ";
     p p->pnext;
  }
  cout << endl;
}
// Ro‘yxatda element qidirish, C
TNode* Find(TNode *phead, int x)
{
TNode *pphead;
  wxile(p)
  if (p->valuex) return p;
 else p p->pnext;
 return 0;
}
//Ro‘yxat elementini o‘chirish
```

```
void deleteList(TNode *phead) {
 if(phead) {
deleteList(phead->pnext);
   if(phead)
              delete phead;
   }
}
 //Asosiy qism
int main() {int mas[1000], N, key; TNode* T;
clrscr();
   TNode *phead  0;
  //srand(time(0));
 cout<<"Ro‘yxat uzunligini kirit"<<endl;
  cin>>N;
   cout<<"Elementlarni kirit!"<<endl;
 for(int j0; j<N; j)
    cin>>mas[j];
   for(int i  0; i < N; i)
add2list(&phead,mas[i]);
  cout<<"Qidiruv elementni kiriting!"<<endl;
   cin>>key;
     // rand() % 100);
 cout << "Elements of the list:" << endl;
   print(phead);
TFind(phead,key);
   if (T0) cout <<"Bunday element yo‘q"<<endl;
 else cout <<"Bunday element bor"<<" "<<T->value<<" "<<T<<endl;
   //deleteList(phead);
   getch(); }
```

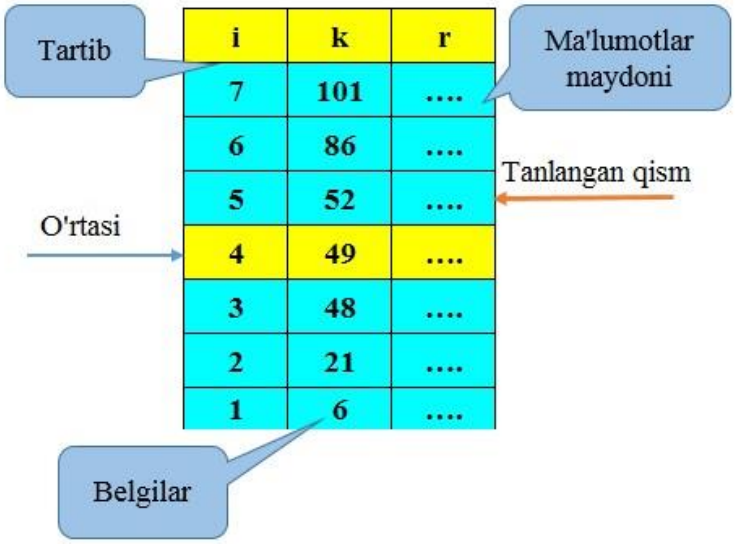
Bu yerda ro‘yxatli tuzilmaning afzalligi shundan iboratki, ro‘yxatga elementni qo‘shish yoki o‘chirish tez amalga oshiriladi, bunda elementni qo‘shish yoki o‘chirish uning soniga bog‘liq bo‘lmaydi, massivda esa elementni qo‘shish yoki o‘chirish taxminan barcha elementlarni yarmini siljitishni talab etadi. Ro‘yxatda qidiruvni samaradorligi taxminan massivniki bilan bir xil kuchga ega bo‘ladi. Umuman olganda, ketma-ket qidiruvda samaradorlikni oshirish eng qulay variant hisoblanadi. Faraz qilaylik, ma’lum vaqt ma’lumotlar yig‘ilib, yana ancha vaqtdan so‘ng, ular qayta ishlansin. Bu qayta ishlash mobaynida bu ma’lumotlar to‘plangan joyda qayta saralanadi.

Binar qidiruv ular ustida amallar bajarish

Faraz qilaylik, o‘sish tartibida tartiblangan sonlar massivi berilgan bo‘lsin. Bu usulni asosiy g‘oyasi shundan iboratki, tasodifiy qandaydir A_M element olinadi va u X qidiruv argumenti bilan taqqoslanadi. Bunda agar A_MX bo‘lsa, u holda elementlarni qidiruv jarayoni yakunlanadi; agar A_M <X bo‘lsa, u holda bu indekslari M dan kichik yoki teng bo‘lgan barcha elementlarni keyingi qidiruvdan chiqarib yuborishga to‘g‘ri keladi. Xuddi shu kabi, agar A_M >X bo‘lsa hamdir. Bu yerda M ixtiyoriy tanlanganda ham taklif qilinayotgan algoritmlar korrekt ko‘rinishida ishlaydi. Shu sababli bunda M ni shunday tanlash kerakki, tadqiq qilinayotgan algoritm samaraliroq natija bersin, ya’ni uni shunday tanlash kerakki, iloji boricha kelgusi jarayonlarda ishtirok etuvchi elementlar soni ancha kam eshtirok etsin. Bunday holatda o‘rtacha elementni, ya’ni massiv o‘rtasi tanlansa, yechim mukammal bo‘ladi.

Quyidagi chizma ko‘rinishida berilgan massivlarni qarab chiqaylik. Faraz qilaylik, kaliti 52 ga teng bo‘lgan elementni topish talab etilsin. Bu yerda dastlabki taqqoslanadigan element 49 bo‘ladi. 49<52 bo‘lgani uchun keyingi qidiruv 49 dan yuqorida turgan elementlar orasida amalga oshirila boshlaydi. Bunda yangi hosil bo‘lgan massiv o‘rtasi 86 ga teng bo‘ladi. Agar oldingi berilgan kalit bilan ushbu kalitni taqqoslasak, 86>52 bu ko‘rinishda keladi. Demak, navbatdagi qidiruvlar 86 bilan 49 orasidagi elementlar ichida amalga

oshirilishi kerak bo‘ladi. Endi keyingi qadamda ma’lum bo‘ldiki, izlanayotgan elementlar o‘rtasidagi element kaliti 52 ga teng ekan. Shunday qilib, massivda berilgan kalitga teng bo‘lgan element topib olindi.



3-rasm. Massivlar

Yuqoridagi algoritmni amalga oshirish dasturi C dasturlash tilida quyidagicha ko‘rinishda bo‘ladi:

```
#include <iostream.h>
 #include <conio.h>
 int Binsearch(int a[], int N, int key, int *t){
int l0, rN-1, mid(lr)/2;
            wxile (l<r){*t1;
if (a[mid]key) return mid;
if (a[mid]>key) rmid-1;
        else lmid1;
mid(lr)/2;}
a[N]key;
 return N;
}
 main ()
```

```
{
int i, N, mas[1000], key, P, t0;
cout<<endl<<"Masiv uzunligini qo‘shish!"<<endl;
cin>>N;
 cout<<"Massiv elementlarini qo‘shish!"<<endl;
for (i0; i<N;i)
 cin>>mas[i];
cout<<"Qidiruv elementini qo‘shish!"<<endl;
cin>>key;
PBinsearch(mas,N,key, &t);
if (PN) cout<<"Bunday elementni massivga qo‘shis lozim"<<" "<<P1<<"
"<<t<<endl;
 else cout<<"Bunday element mavjud"<<" "<<P1<<" "<<t<<endl;
 getch();
return 0;
}
```

Agar key 101 ga teng bo‘lsa, kerakli yozuv 3 marta taqqoslash orqali aniqlanadi, bunda ketma-ket qidiruvda taqqoslashlar soni 7 ta bo‘lar edi.

Agar S – taqqoslashlar soni va n esa jadvaldagi elementlar soni bo‘lsa edi, u holda S $\log_2 n$.

Masalan, n 1024.

Ketma-ket qidiruv esa S 512, binar qidiruvda S 10.

Agar katta hajmdagi ma’lumotlarning ichidan qidiruvlar amalga oshirilayotgan bo‘lsa, u holda binar va indeksli ketma-ket qidiruvni umumlashtirib olib borish eng ma’qul yechimdir. Buning sababi, bunda har ikkala qidiruvda ham tartiblangan massiv orqali amalga oshiriladi.

Qidirish usullarining samaradorligi va ularni optimallashtirish

Ketma-ket qidiruvni samaradorligi shundaki, bunda ixtiyoriy qidiruvning samaradorligi jadvaldagi ma’lumotlarning kalitlari bilan solishtirishlar soni – **S** bilan baholanadi.

Agar solishtirish soni qancha kichik bo‘lsa, qidiruv algoritmi samaradorligi shuncha yaxshi kechadi.

Massivda ketma-ket qidiruvning samaradorligi quyidagi ko‘rinishda bo‘ladi:

C 1 n, C (n 1)/2.

Umuman olganda, ro‘yxatda ham samaradorlik yuqoridagi ko‘rinishda bo‘ladi. Garchi massivda ham bog‘langan ro‘yxatda ham qidiruv samaradorligi bir xil bo‘lsada, ma’lumotlarni massiv va ro‘yxat ko‘rinishda tasvirlashning o‘ziga xos kamchilik va afzalliklari o‘ziga xos ko‘rinishda mavjud bo‘ladi.

Qidiruvning maqsadi - quyidagi jarayonlarni bajarilishidan iboratdir:

1) Ma’lumotlar ichida topilgan yozuvni o‘qish;
2) Ma’lumotlar ichida qidirilayotgan yozuv topilmasa, uni jadvalga qo‘yish;
3) Ma’lumotlar ichida topilgan yozuvni o‘chirish.

Birinchi jarayon bu qidiruvning o‘zi bo‘lib, massiv uchun ham ro‘yxat uchun ham bir xil bo‘ladi. Ikkinchi va uchinchi jarayonda esa qidiruv ro‘yxatli tuzilma ichida samaraliroq bo‘ladi. Buning sababi shundan iboratki, massivda elementlarni siljitish lozim bo‘ladi. Agar q massivda elementlarni siljitishlar soni bo‘lsa, u holda q (n 1)/2 bo‘ladi.

Mavzuga oid test savollari

1. Qidiruvni vazifasi nimadan iborat?

a) berilgan argumentga mos keluvchi ma’lumotlarni massiv ichidan topish
b) massivda ma’lumot yo‘qligini aniqlashdan
c) ma’lumotlar yordamida argumentni topish yoki yangilashdan

2. Berilgan argumentga mos keluvchi ma’lumotlarni massiv ichidan topish – …?

a) Qidiruv

b) Saralash jarayoni

c) Algoritmlash

3. Jadvalning tuzilmasiga qarab nechta qidiruv usullari mavjud?

a) 4

b) 5

4. Chiziqli qidiruv g‘oyasi nimadan iborat?

a) har bir element ketma-ket ko‘rib chiqiladi.

b) elementlar ketma-ket jadval o‘rtasidan boshlab ko‘rib chiqiladi.

c) elementlarni ko‘rib chiqish ketma-ket ravishda boshidan oxirigacha va aksincha, 2 ta element tashlab qaraladi.

5. Katta O notasiyada belgilangan chiziqli qidiruv samaradorligini ko‘rsating?

a) $O(N)$

b) $O(\log_2(N))$

c) $O(1)$

6. Katta O notasiyada belgilangan binar qidiruv samaradorligini ko‘rsating?

a) $O(\log_2(N))$

b) $O(N)$

c) $O(1)$

Nazorat savollari

1. Qidiruv tushunchasi deganda nimani tushunasiz?
2. Qidiruv turlariga misollar keltiring?
3. Chiziqli qidiruv haqida ma’lumot bering?
4. Binar qidiruv turi deganda nimani tushunasiz?

Xulosa

Ma’lumki, ma’lumotlar bugungi kunda tasavvur qilib boʻlmaydigan darajada koʼp. Shunday boʻlsa-da, ular ustida turli xil amallarni bajarish har bir foydalanuvchi uchun ancha ishini yengil qiladi. Masalan, toʻplangan ma’lumotlar orasidan foydalanuvchi qidiradigan ma’lumotni tez va oson topishi unga har tomonlama ishini yengil kechganidan darak beradi. Bunga misol qilib, Google dagi ma’lumotlar bazasidan maʻlumot qidirganda bunday holatni yaxshi sezish mumkin.

Qurilmalarni ishlash qoidasiga binoan, uning vazifasi berilgan argument boʻyicha massiv ma’lumotlarining ichidan mazkur argumentga mos ma’lumotlarni qidirish va mavjud boʻlsa, uni topishdan iboratdir. Bu esa ma’lumotlarni qidirishni optimallashtirilganidan darak beradi.

2.3. Ma’lumotlarni xeshlash algoritmlari. Xesh jadval va xesh funksiyalari

Reja:

1. Xeshlash tushunchasi va uning vazifasi.
2. Xesh jadvallar va ularning turlari.
3. Xesh funksiyalari va ular ustida amallar bajarish.

Tayanch iboralar: xeshlash, xeshlash algoritmlari, xesh funksiyalari, xesh jadvali, polinominal xeshlash, saralash.

Xeshlash tushunchasi va uning vazifasi

Heshlash tushunchasi – bu berilgan ma’lumotlardan matematik jadvallar yordamida yangi ma’lumotlarni yaratish tushuniladi. Heshlashni esa amaliy hisoblash, statistika, arifmetik amallar, funksiyalar va turli jadvallar yordamida amalga oshirilishi mumkin boʻladi.

Heshlash, matematikadagi eng asosiy amallardan biri boʻlib hisoblanadi. Heshlash hisoblashning asosiy turidir. Bunga qoʻshimcha matematik amallariga qaraganda heshlash amaliyoti jadvallar va grafikalar yordamida amalga oshirilishi lozim boʻladi. Juda koʻp sohalarda heshlash amaliyoti qoʻllaniladi.

Misol uchun turli tadbirlarni rejalashtirish, moliya hisob-kitob, statistik analiz, mexanika va fizika, elektronika, kibernetika va boshqa sohalarda heshlashdan juda yaxshi foydalaniladi. Hozirgi vaqtda heshlashni avtomatlashtirish, ya’ni, kompyuterlar yordamida bajarish uchun turli heshlash algoritmlari ishlab chiqilgan. Heshlash algoritmlari kompyuterlar yordamida kattalashtirilgan hajmli ma’lumotlarni boshqarish, ma’lumotlar tahlili, mashqlarni rejalashtirish, turli masalalarni hal qilish uchun ishlatib kelinadi.

Heshlashtirish, matematikadagi eng asosiy amallardan biri bo‘lib, ma’lumotlardan yangi ma’lumotlar yaratishni o‘z ichiga oladi. Bu amalni amaliy hisoblash, statistika, arifmetik amallar, jadvallar, grafiklar va boshqa metodlari yordamida bajarish mumkin.

Heshlashtirishning bir qancha turlari mavjud bo‘lib, bunda ular sonlar, kengligi, balandligi, shakllar va boshqa turlardan tashkil topgan. Aynan shunga ko‘ra, hozirgi kunda heshlashtirishni avtomatlashtirilib, kompyuterlar yordamida yana ham kattalashtirilgan hajmli ma’lumotlar sifatida boshqariladi. Heshlashtirishning bu xususiyati, uning xususiyatlarini o‘rganish uchun turli heshlash algoritmlari yaratilgan. Bugungi kunda bu algoritmlar boshqa bir necha sohalarda heshlashtirishga qo‘llanilib kelmoqda, masalan, ma’lumotlar tahlili, moliya hisob-kitobi, mexanika va fizika, elektronika, kibernetika va boshqa sohalar bundan mustasno emas.

Hozirda heshlashtirish va avtomatlashtirish, ya’ni, kompyuterlar yordamida heshlashtirish, har bir sohani kattalashtirishni ta’minlash uchun juda muhim hisoblanadi. Bunday holatda hozirgi kunda kichik va katta tashkilotlar matematik jadvallarini yaratish va ularni ishlatishni osonlashtirish uchun turli matematik dasturlarini va heshlashda ulkan dasturlarni yaratishmoqda. Ayni damda heshlashtirish usullari, matematik modellar yordamida amalga oshiriladi. Unda matematik modellar, bir qator matematik formulalardan iborat bo‘lib, tizimlarning davr ko‘rsatishini, tahlil qilishni yoki o‘zgaruvchilarning tahlilini yaratishga yordam beradi.

Matematik modellar, tizimning ta’sirini va o‘zgaruvchanlarini hamda natijalarni to‘g‘ridan-to‘g‘ri ifodalaydi. Bunda heshlash usuli, tizimning nazariy tahlilini va tasnifini hamda optimallashtirishini qulaylashtiradi. Matematik modellar o‘z ichiga ko‘plab heshlash usullarini oladi. Heshlash usullari ko‘pincha tizimning umumiy holatini, joriy va kelajakdagi ko‘rsatuvlarini aniqlashda juda katta yordam beradilar.

Matematik modellar yordamida yaratilgan bu heshlash o‘z ichiga statistik, integrallar, differensial tenglamalar, algoritmik heshlash va boshqa usullarni qamrab olgan. Bunday usullar matematikning bir qator asosiy fanlariga asoslangan bo‘ladi.

Shunday qilib, heshlashtirish usullari va matematik modellar, turli sohalarda amaliyotni tizimga solishda juda katta ahamiyatga egadir. Hozirgi kunda heshlashtirish va matematik modellarni foydalanish orqali tadbirlarni rejalashtirish, yangi mahsulotlarni yaratish, ma’lumotlar tahlili va shunga o‘xshash ko‘plab sohalarda ishlatilib kelinmoqda.

Xesh jadvallar va ularning turlari

Xesh jadvallari, bir qator amallarni o‘z ichiga oladi. Bu jadvallar, berilgan ko‘rsatkichlarga qarab, amalni tez-tez bajarishni ta’minlash uchun ishlab chiqilgan. Hesh jadvallari, qo‘shma hisoblashning bir qator qiyinchiliklarini yechishda juda yaxshi ish olib boradi. Bunda, o‘zgaruvchilar va ulardan hosil bo‘lgan natijalar kiritiladi va heshlash amallari o‘z jadvallarida joylashtiriladi.

Ayni damda hesh jadvallari, amalni tez-tez bajarishda va amallarni tez aniqlashda juda yaxshi xizmat ko‘rsatadi. Masalan: moliya hisob-kitoblarda, ma’lumotlar tahlilida, o‘quv dasturlarida va boshqa sohalarda heshlash usuli keng qo‘llaniladi.

Bundan tashqari hesh jadvallari, ko‘pincha arifmetik amallarni bajarish uchun ishlatilib kelinadi. U asosan, statistik amallar, funksiya hisoblanishi, matritsa amallari va boshqa hajmli matematik amallarini bajarishda ham hesh jadvallaridan foydalaniladi.

Shu bilan birga hesh jadvallari turli xil shakllarda ham ishlatiladi. Misol uchun katta jadvallar ma’lumotlar tahlilida, kelajakdagi ko‘rsatuvlarini bashoratlashda yoki kichik jadvallar, ro‘yxatlar va grafikalar yaratishda foydalaniladi.

Hesh jadvallari, elektronika, kibernetika, mexanika va boshqa sohalarda ham keng qo‘llanilib kelinmoqda. Misol uchun elektronika sohasida hesh jadvallari elektronik qurilmalar, mexanika sohasida hesh jadvallari yengil burg‘ular va boshqa mexanik qurilmalar uchun sarflanadi.

Shunga ko‘ra, hesh jadvallari turli sohalarda amalni tez va yaxshi bajarish uchun qo‘llaniladigan juda muhim bir vosita sifatida hisobga olinadi.

Xesh jadvali – bu assotsiativ massiv interfeysini amalga oshiruvchi ma’lumotlar tuzilmasi bo‘lib, ya’ni juftlarni saqlashga va uchta amalni bajarishga imkon beradi: yangi juftlikni qo‘shish, qidirish amali hamda juftlikni kalit bilan o‘chirishda foydalaniladi.

Hozirda xesh jadvallarining ikkita asosiy varianti mavjud: Biri zanjirli va ikkinchisi ochiq manzilli. Xesh jadvali ba’zi bir massivini o‘z ichiga olib, ularning elementlari juftliklar yoki juftliklar ro‘yxati bo‘lib hisoblanadi.

Xeshlash – bu ixtiyoriy uzunlikdagi kirish ma’lumotlari majmuasini ma’lum bir algoritm tomonidan bajarilgan, belgilangan o‘lchamdagi chiqish massiviga aylantirish jarayoni bo‘lib xizmat qiladi. Bunday algoritmni amalga oshiruvchi funksiya **xesh funksiya** yoki **transformatsiya natijasi xesh** qolaversa, **xesh yig‘indisi** deb yuritiladi.

Xesh funksiyasi quyidagi xususiyatlarga egadir:

- 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 cheklanmay qoladi. Natijada, xesh funksiyasi kirish ma’lumotlari

to‘plamidan xeshlar to‘plamiga xaritalash amalga oshiriladi, bu esa ularning kardinalligi ancha past bo‘lganidan darak beradi.

Dirixle prinsipiga ko‘ra, har bir xesh uchun bir nechta turli xil ma’lumotlar to‘plamlari mavjud bo‘ladi. Bunday moslik **kolliziya** deb ataladi.

Agar biror-bir muammoni hal qilishda kirish ma’lumotlari cheklangan bo‘lsa, siz bunday xeshlar to‘plamini tanlashingiz lozim bo‘ladi, shunda uning aniqligi kirish ma’lumotlari to‘plamining muhimligidan oshib ketadi.

Bunday holda, inyeksion xaritalashni aniqlaydigan xesh funksiyasini qurish mumkin bo‘ladi. Biroq, umuman olganda, kolliziya muqarrardir. Bu kolliziya ehtimoli xesh funksiyasi sifatini baholash uchun ishlatiladi. Eng 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 zarur.

Bunda xeshlash o‘zi ob’ektga tasodifiy o‘zgaruvchini xaritalashga o‘xshash bo‘ladi.

Birinchi xususiyat natijasida xeshlar o‘zlarini bir tekis taqsimlangan tasodifiy o‘zgaruvchilar kabi tutishi kerak, bu butun diapazondan foydalanishni ta’minlaydi, bu juda foydali bo‘lishi mumkin, misol uchun xesh jadvalini tuzishda foydalaniladi.

Polinomial xeshlash. Ko‘rinishi oddiy, ammo samarali xeshlash algoritmini ko‘rib chiqaylik. Xesh funksiya quyidagicha aniqlanadi:

$$h(s) = \sum_{i=0}^{N} b^{N-i} \cdot code(s_i) \qquad (1)$$

yoki $$h(pref\ f_i) = b \cdot h(pref_{i-1}) + code(s_{i-1}) \qquad (2)$$

bu yerda $N = strlen(s) - 1$, $pref_i$ uzunlik prefiksi i, b -baza, asos. $code(s_i)$ simvol kodi.

Agar (1) formula kengaytirilsa, N tartibli polinom olinadi. (2) formula xeshni rekursiv shaklda o‘rnatadi va kod yozishda foydalaniladi.

Belgilar kodiga va asosga e’tibor qaratish lozim bo‘ladi, chunki bazani tanlash kodlarga bog‘liq bo‘lmay qoladi. Kod ASCII jadvalidagi belgilar yoki alfavitdagi tartib raqam bo‘lishi ham 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 imkoniyat yaratadi. Simvollar satrdagi har qanday belgining maksimal kodidan oshib ketishi kerak va odatda asosiy son tanlanadi.

Masalan: 31, 37 va boshqalar asoslari inglizcha kichik harflarning satrlari uchun javob bo‘lib boradi. Bu yerda shuni ta’kidlash joizki, xesh hech qanday cheklovga olib kelmaydi, balki bu holat xeshlash ta’rifiga zid keladi. Bunday holda, ikkita chiqish usuli mavjud bo‘lib, ularning biri modul bo‘yicha bo‘lish amalidan ikkinchisi esa uzun arifmetikadan foydalanishdir.

Birinchi variant bu 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. Natijada, esa cheklangan xeshlar to‘plami olinadi, ammo yana kolliziya xavfi mavjud bo‘ladi. Bundan tashqari, ko‘p polinomli xeshni "buzish" ehtimoli ham mavjuddir.

Ikkinchi variantda kolliziya ehtimoli pastroq bo‘ladi. Biroq, kattaroq xeshlar to‘plamini qo‘llab-quvvatlash, qo‘shimcha xotira va ikkita xeshni taqqoslash uchun zarur bo‘lgan vaqtni talab etadi, bu oddiy ma’lumotlarni taqqoslashdan ko‘ra tezroq amalga oshiriladi.

Masalan, satrni inglizcha kichik harflardan iborat, - deb taxmin qilaylik. Quyida 37 raqamini asos qilib olib, uni C dasturida ko‘rib chiqaylik.

```
#include <iostream>
 #include <string.h>
 using namespace std;
long long Heshlash(char s[])
```

```
 {
long long h  0;
int base 37;
for(int i0;  i<strlen(s); i)
{
h  h* base  s[i] - 61 1;  }
return h;
} 163
 int main()  {
char s[100];
for(int i1; i<10; i) {
cin.getline(s,100);
cout<<s<<" "<<Heshlash(s);
cout<<endl; }  }
```

1-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 konteynerlar, yaxshi tanlangan xesh funksiyalarini taqdim etib boradi. Bu jadvaldan xesh funksiyalaridan foydalanganligi aniq ko‘rinib turibdi. Ammo, bunda yana bitta savol tug‘iladi: bu savolda nega bular doimo ishlatilmaydi, - degan muammo mavjud. Javob shundan iboratki, har doimgidek, birdaniga hamma narsani bajarib ko‘rsatish

mumkin emas, ya’ni ham tezlikdan, ham xotiradan foydalanish kishiga sal noqulaylikni tug‘diradi. Bunday holatda xesh jadvallari noqulay va ular operatsion jarayonning asosiy savollariga tezda javob berishlari bilan birga ulardan foydalanish har doim oson bo‘lavermaydi.

Xesh funksiyalari va ular ustida amallar bajarish

C dasturlash tilida xesh jadvallarni realizatsiya qilish. C dasturlash tilida xesh jadvallarni hosil qilish uchun map deb nomlangan konteyneri aniqlangan. map konteyner list, vector, deque kabi boshqa konteynerlarga juda o‘xshaydi, lekin ozgina farqi mavjuddir.

Bu konteynerga birdaniga ikkita qiymat qo‘yish mumkin bo‘ladi. Bu map misolni dasturda 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 elementlarni sonini qaytaradi;

max_size() - mapda saqlanishi mumkin bo‘lgan elementlarni maksimal sonini qaytaradi;

empty() - mapning bo‘shliqka tekshiradi;

pair_insert (keyvalue, mapvalue) - mapga yangi elementni qo‘shiladi; **erase**(iterator position) - elementni iterator ko‘rsatgan joydan chiqaradi; **erase**(const g) - mapdan "g" kalit qiymati chiqariladi;

clear() - mapdagi barcha elementlarni chiqarib tashlanadi.

Kolliziya muammosi. Bu yerda tabiiyki, savol tug‘iladi, nega bir qator katakchaga ikki marta kirib olish mumkin bo‘lmaydi, chunki har bir elementga mutlaqo boshqacha natural sonlarni taqqoslaydigan funksiyani taqdim etish shunchaki, mumkin bo‘lmaydi.

Kolliziya muammosi xesh funksiyasi turli elementlar uchun bir xil natural sonni hosil qilganda paydo bo‘ladigan muammodir.

Ushbu muammoning bir nechta yechimlari mavjuddir. Ular zanjirlash usuli va ikki marta xeshlash usulidir.

Mavzuga oid test savollari

1. Xeshlash – bu …?

a) Funksiya yordamida xesh-jadval to‘ldiriladi va undan qidiriladi.

b) Ma’lumotlar butun jadval bo‘yicha operativ xotirada kichik manzillardan boshlab, to katta manzillargacha ketma-ket qarab chiqiladi.

c) Berilgan massiv o‘rtasidagi element olinadi, ya’ni $m = (L + R)/2$, va u qidiruv argumenti bilan taqqoslanadi. Topilmasa chegaralar mos ravishda o‘zgaradi.

d) Indekslar jadvalidan guruh topiladi va unda ko‘rsatilgan mos chegaralarda chiziqli algoritm oshira boshlanadi.

2. Katta O notasiyada belgilangan xeshlash va rexeshlash qidiruv samaradorligini ko‘rsating?

a) $O(1)$

b) $O(N)$

c) $O(\log_2(N))$

d) $O\sqrt{(N)}$

3. Xeshlashtirish algoritm tartibi qanday?

a) Konstantali

b) Chiziqli

c) Logarifmik

d) Eksponensial

Nazorat savollari

1. Xesh jadval deganda nima tushunasiz?
2. Xesh jadvallardan foydalanish samaradorligini taqqoslay olasizmi?
3. Xesh funksiyasiga misol keltiring.
4. Satrlar uchun xesh funksiyasini qo‘llang.
5. Xesh funksiga ma’lumotlar strukturasi qo‘llaniladigan sohalarga qaysilar kiradi?

Xulosa

Ma’lumki, xeshlash tushunchasi – bu berilgan ma’lumotlardan matematik jadvallar yordamida yangi ma’lumotlarni yaratishga imkon beradi. Xeshlashni esa amaliy hisoblash, statistika, arifmetik amallar, funksiyalar va turli jadvallar yordamida amalga oshirilish mumkin bo‘ladi. Shunday ekan, xeshlash, matematikadagi eng asosiy amallardan biri bo‘lib hisoblanadi.

Xeshlash hisoblashning asosiy turidir. Bunga qo‘shimcha matematik amallariga qaraganda xeshlash amaliyoti jadvallar va grafikalar yordamida amalga oshirilishi lozim bo‘ladi. Shuning uchun bu xeshlash yordamida foydalanuvchi jadvalli ma‘lumotlar ustida turli xil amallarni bajarishi mumkin.

2.4. Ma’lumotlarni saralash algoritmlari. Saralash tushunchasi va uning vazifasi. Saralashning qat’iy usullari

Reja:

1. Saralash tushunchasi va uning vazifasini misollar yordamida tushunish.
2. Saralashning qat’iy usullari va ularning samaradorligini ko‘rish.

Tayanch iboralar: saralash, algoritm samaradorligi, almashtirish, statik, dinamik, massivlar, chiziqli konteynerlar, iteratorlar.

Saralash tushunchasi va uning vazifasini misollar yordamida tushunish

Saralash – bu berilgan to‘plam elementlarini biror-bir tartibda joylashtirish jarayoni bo‘lib hisoblanadi. Saralashni maqsadi tartiblangan to‘plamda kerakli elementni topishni osonlashtirishdan iboratdir.

Saralash dasturlarni translyatsiya qilinayotganda, kataloglar, kutubxonalar, ma’lumotlar majmuasini tashqi xotirada joylayotganda, ma’lumotlar bazasi yaratilayotganda vujudga keladi. Ma’lumki, saralashning turli xil algoritmlari mavjuddir. Sababi, bitta masalani saralash uchun juda ko‘plab turli xil algoritmlardan foydalanish mumkin bo‘ladi. Berilgan masalani hal qilishda ba’zilari mukammal bo‘lishi ham mumkin. Shuning uchun saralash masalasida algoritmlarni qiyosiy tahlilini o‘tkazish zarurati uchrab turadi.

Saralash masalasini qo‘yilishini quyidagicha yozish mumkin. Faraz qilaylik, a_1, a_2 ,…, a_n, elementlar ketma-ketligi berilgan bo‘lsin. U holda saralash algoritmi elementlarni massivga shunday joylashtiradiki, natijada ular qandaydir munosabatga nisbatan $f(a_{k1})$ $f(a_{k2})$ … $f(a_{kn})$ tartibga keladi. Odatda, f tartiblash funksiyasi qandaydir maxsus qoida bilan hisoblanmasdan, balki elementni kalit qiymati bo‘yicha massiv elementlari bilan tartiblanadi.

Ko‘p hollarda ma’lumotlarga qayta ishlov berilayotganda ma’lumotni informatsion maydonini hamda uni mashinada joylashishini bilish zarurdir.

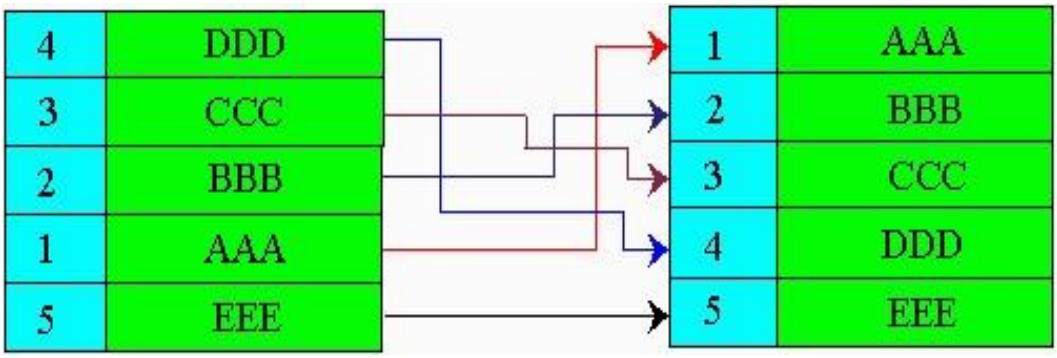
Saralashni ikkita turi mavjud bo‘lib, bular - ichki va tashqi.

Ichki saralash – bu operativ xotiradagi saralash.

Tashqi saralash – tashqi xotirada saralashdir.

Saralash – bu ma’lumotlarni kalitlari bo‘yicha xotirada regulyar ko‘rinishda saqlashdir.

Regulyarlik – deganda, ma’lumotlar kalit qiymatlari bo‘yicha massivda boshidan oxirigacha o‘sish yoki kamayishi tushuniladi.



1-rasm. Massivda o‘sish yoki kamayishi

Agar saralanayotgan yozuvlar xotirada katta hajmni egallasa, u holda ularni almashtirishlar katta vaqt va xotira talab etadi. Ushbu sarfni kamaytish maqsadida, saralash kalitlar manzili jadvalida amalga oshiriladi. Bunda faqatgina ma’lumot ko‘rsatkichlari almashtirilib, massiv o‘z joyida qolaveradi. Bunday holatda yuqoridagi usul manzillar jadvalini **saralash usuli** deb aytiladi.

Saralanayotganda bir xil kalitlar bir-biri bilan uchrashi mumkin, bu holda ular saralangandan keyin bir xil kalitlilar boshlang‘ich tartibda qanday

joylashgan bo‘lsa, ushbu tartibda ham shunday maqsadga muvofiq bo‘ladi. Bunday usulga **turg‘un saralash** deb yuritiladi.

Saralash samaradorligini bir necha me’zonlarda baholash mumkin:

− saralashga ketgan vaqt;

− saralash uchun talab qilingan operativ xotira;

− dasturni ishlab chiqishga ketgan vaqt.

Bunda birinchi me’zonni oladigan bo‘lsak, saralash bajarilganda taqqoslashlar yoki almashtirishlar sonini hisoblash mumkin bo‘ladi. Faraz qilaylik, N $0{,}01n^2$ 10 n – taqqoslashlar soni. Agar n < 1000 bo‘lsa, u holda ikkinchi qo‘shiluvchi katta, aks holda ya’ni, n > 1000 bo‘lsa, birinchi qo‘shiluvchi katta bo‘ladi. Demak, kichkina n larda taqqoslashlar soni n ga teng bo‘ladi, katta n larda esa n^2 ga teng bo‘ladi.

Saralashda taqqoslashlar soni quyidagi oraliqlarda mavjud bo‘ladi:

0(n log n) dan 0 (n^2) gacha; 0 (n) – ideal holatda.

Saralashni quyidagicha usullari mavjud:

− qat’iy, ya’ni to‘g‘ridan-to‘g‘ri satralash usullar;

− yaxshilangan usullar.

Qat’iy usullar:

1. to‘g‘ridan-to‘g‘ri qo‘shish usuli;

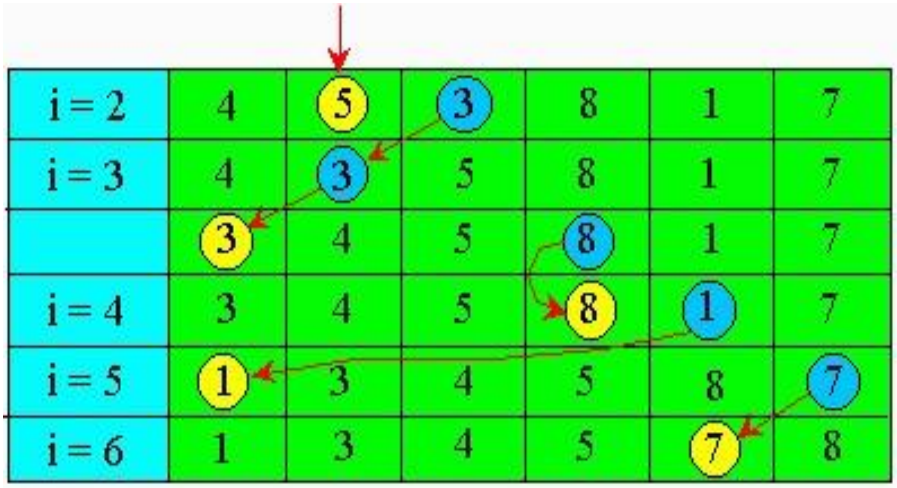
2. to‘g‘ridan-to‘g‘ri tanlash usuli;

3. to‘g‘ridan-to‘g‘ri almashtirish usulidir.

Yuqorida keltirilgan usularda almashtirishlar soni deyarli bir xil bo‘ladi.

Saralashning qat’iy usullari va ularning samaradorligi ko‘rish

To‘g‘ridan-to‘g‘ri qo‘shish usuli bilan saralash. Bunday usul karta o‘yinlarida keng qo‘llaniladi. Elementlar hayolan “tayyor” a(1),..,a(i-1) va boshlang‘ich ketma-ketliklarga bo‘linadi. Har bir qadamda boshlang‘ich ketma-ketlikdan i-chi element ajratib olinib, tayyor ketma-ketlikning kerakli joyiga qo‘shilib boradi. Taklif qilinayotgan usulni quyidagi misolda ko‘rib chiqaylik. Faraz qilaylik, kalit qiymati 4, 5, 3, 8, 1, 7 bo‘lgan elementlar berilgan bo‘lsin.



2-rasm. To‘g‘ridan-to‘g‘ri qo‘shish usuli bilan saralash

Kerakli joyni qidirish jarayonini quyidagi tartibda olib borish qulay sanaladi. Taqqoslashlarni amalga oshirish mobaynida, navbatdagi a(j) element bilan solishtiriladi, keyin esa x bo‘sh joyga qo‘yiladi yoki a(j) o‘nga suriladi va jarayon chapga “ketadi”. Shuni e’tiborga olish lozimki, saralash jarayoni quyidagi shartlarni birortasi bajarilganda to‘xtatiladi:

1. x elementi kalitidan kichik kalitli a(j) element topildi.
2. tayyor ketma-ketlikning chap tomoni oxiriga yetib borildi.

Taklif etilayotgan usul algoritmi quyidagicha bo‘ladi:

```
    for (int i1;i<n;i) {
 int ji;
    wxile(a[j]<a[j-1]) {
int ta[j-1];
        a[j-1]a[j];
      a[j]t;
       jj-1;  }  }
```

Algoritm samaradorligi.

Faraz qilaylik, taqqoslashlar soni S, o‘rinlashtirishlar soni M bo‘lsin. Agar massiv elementlari kamayish tartibida bo‘lsa, u holda taqqqoslashlar soni eng katta bo‘ladi, ya’ni u $C_{max} = \frac{n(n-1)}{2}$ ga teng bo‘ladi, $O(n^2)$.

O‘rinlashtirishlar soni esa $M_{max} = C_{max} + 3(n-1)$ ga teng bo‘lib qoladi, ya’ni $O(n^2)$. Agar berilgan massiv o‘sish tartibida saralangan bo‘lsa, u holda taqqoslashlar va o‘rinlashtirishlar soni eng kichik bo‘lib qoladi, ya’ni $C_{min} = n-1$, $M_{min} = 3(n-1)$.

To‘g‘ridan-to‘g‘ri tanlash usuli bilan saralash.

Faraz qilaylik, $a_1, a_2, \dots, a_n$ elementlar ketma-ketligi namoyon qilingan bo‘lsin.

Mazkur usul quyidagi tamoyillarga asoslangan bo‘ladi:

1. Berilgan elementlar ichidan eng kichik kalitga ega element tanlanadi.

2. Ushbu element boshlang‘ich ketma-ketlikdagi birinchi element a_1 bilan o‘rin almashtiradi.

3. Undan keyin ushbu jarayon qolgan n-1 ta element, n-2 ta element va xokazo, toki bitta eng “katta” element qolgunicha davom ettirilib boriladi.

Bu yerda taklif qilinayotgan usul algoritmi va dasturi quyidagicha bo‘ladi:

C dasturlash tilida:

```
for(int i0;i<n-1;i)
 for(int ji1;j<n;j)
    if (a[i] > a[j]) {
      int k a[j];
   a[j] a[i];
       a[i] k;    }
```

Algoritmning samaradorligi.

Taqqoslashlar soni – bu

$$M \ \frac{n}{2}(n-1) = \frac{n^2 - n}{2}.$$

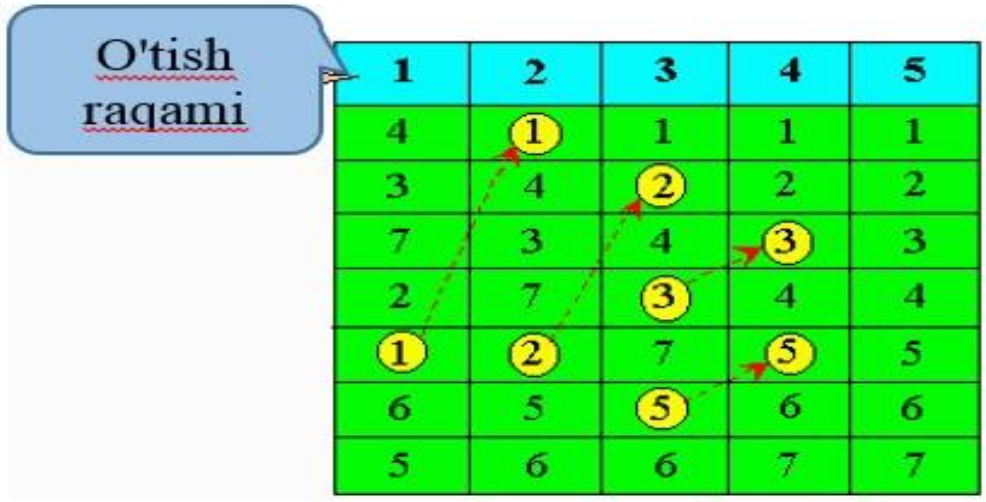
Almashtirishlar soni

$$C_{min} \ 3(n-1), C_{max} \ 3(n-1)\frac{n}{2} \qquad (n^2 \text{ tartib}).$$

Ushbu usul bo‘yicha saralash bajarilsa, eng yomon holda taqqoslashlar va almashtirishlar soni ya’ni tartibi n^2 hosil bo‘ladi.

To‘g‘ridan-to‘g‘ri almashtirish usuli bilan saralash. Ushbu usulni g‘oyasi quyidagicha bo‘ladi: n - 1 marta massivda quyidan yuqoriga qarab yurib kalitlar juftijufti bilan taqqoslanib boradi.

Agar pastki kalit qiymati yuqoridagi jufti kalitidan kichik bo‘lsa, u holda ular o‘rni almashtirish zarur bo‘ladi.



3-rasm. To‘g‘ridan-to‘g‘ri almashtirish usuli bilan saralash

C tilidagi dasturi:

```
 for (int i0;i<n;i)
for (int jn-1;j>i;j--)
    if (a[j] < a[j - 1])
 {
      int x a[j - 1];
      a[j - 1]  a[j];
      a[j] x;
  }
```

Bu holda bitta o‘tish “bekor” bo‘ldi. Elementlarni ortiqcha o‘rinlashtirmaslik uchun unda bayroqcha belgisi kiritiladi. Pufaksimon usulni

yaxshilangan usuli sikl saralash usuli bo‘lib, har bir o‘tishdan keyin sikl ichida yo‘q bo‘ladi:

Taqqoslashlar soni

$$M \ \frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4},$$

Almashtirishlar soni

$$C_{max} \ 3\frac{n^2}{4}.$$

Mavzuga oid test savollari

1. **Operativ xotirada bajariladigan saralash qanday ataladi?**

a) ichki saralash

b) to‘liq saralash

c) qo‘shish orqali saralash

d) manzilli jadvalini saralash

2. Saralash usullari orasidan noto‘g‘risini toping.

a) dinamik

b) yaxshilangan

c) logarifmik

d) qat’iy

3. Saralashning qaysi usullari (N ^{2}) **kalitlarni taqqoslash tartibiga ega?**

a) qat’iy

b) binar

c) yaxshilangan

d) logarifmik

4. Berilgan to‘plam elementlarini biror-bir tartibda joylashtirish jarayoni nima deyiladi?

a) Saralash

b) Qidiruv

c) Algoritmlash

d) Uslubiyot

5. Qo‘yish orqali saralash g‘oyasi qaysi javobda to‘g‘ri ko‘rsatilgan?

a) Ob’ektlar hayolan tayyor a(1),...,a(i-1) va boshlang‘ich ketma-ketliklarga bo‘linadi. Har bir qadamda boshlang‘ich ketma-ketlikdan i-chi element ajratib olinib, tayyor ketma-ketlikning kerakli joyiga qo‘shiladi.

b) Berilgan ob’ektlar ichidan eng kichik kalitga ega element tanlanadi. Ushbu element boshlang‘ich ketma-ketlikdagi birinchi element bilan o‘rin almashadi. Undan keyin ushbu jarayon qolgan elementlarda amalga oshiriladi.

c) n - 1 marta massivda quyidan yuqoriga qarab yurib kalitlar jufti-jufti bilan taqqoslanadi. Agar pastki kalit qiymati yuqoridagi jufti kalitidan kichik bo‘lsa, u holda ular o‘rni almashtiriladi.

d) Boshlang‘ich ketma-ketlikning har r o‘rinda joylashgan elementlari guruhlanib, har bir guruh alohida qo‘shish usuli orqali saralanadi.

Nazorat savollari

1. Saralash deganda nimani tushunasiz?
2. Saralashning asosiy usullarini aytib bering.
3. Saralashning qaysi usulari qat’iy usulga tegishli?
4. Saralashning yaxshilangan usullarini aytib bering.
5. Qanday saralash turg‘un deyiladi?
6. To‘g‘ridan-to‘g‘ri qo‘shish usuli g‘oyasi nimadan iborat?
7. To‘g‘ridan-to‘g‘ri tanlash usuli g‘oyasi nimadan iborat?
8. To‘g‘ridan-to‘g‘ri almashtirish usuli g‘oyasi nimadan iborat?
9. Yuqoridagi usullarning bir-biridan farqini aytib bering.

Xulosa

Ma’lumki, saralash – bu berilgan to‘plam elementlarini biror-bir tartibda joylashtirish jarayoni bo‘lib hisoblanadi.

Saralashning maqsadi – tartiblangan to‘plamda kerakli elementni topishni osonlashtirishdan iboratdir. Saralash dasturlarni translyatsiya qilinayotganda, kataloglar, kutubxonalar, ma’lumotlar majmuasini tashqi xotirada joylayotganda, ma’lumotlar bazasi yaratilayotganda vujudga keladi.

Ma’lumki, saralashning turli xil algoritmlari mavjuddir. Sababi, bitta masalani saralash uchun juda ko‘plab turli xil algoritmlardan foydalanish mumkin bo‘ladi. Bunday holatlarda foydalanuvchining ancha vaqti tejaladi. Shuning uchun saralash ma‘lumotlar tuzilmasi va algoritmlari fanining asosiy bo‘g‘ini bo‘lib hisoblanadi.

2.5. Ma’lumotlarni saralash algoritmlari. Saralashning yaxshilangan usullari

Reja:

1. Saralashning yaxshilangan usullari va quiksort – tez saralash algoritmlari ustida amallar bajarish.
2. Birlashtirish orqali (MergeSort) saralashga doir misollar yechish.
3. Shell saralashi (qisqarib boruvchi qadamlar orqali saralash).

Tayanch iboralar: saralash, quicksort saralash, Shell saralashi, MergeSort saralash, ichki saralash, to‘liq saralash.

Saralashning yaxshilangan usullari va quiksort – tez saralash algoritmi amallar bajarish

Ma’lumotlarni saralashda ularni saralash algoritmlaridan foydalaniladi. Saralashning yaxshilangan usullariga quyidagilar kiradi:

- Quiksort – tez saralash usuli.
- Birlashtirish orqali (MergeSort) saralash.
- Shell saralashi (qisqarib boruvchi qadamlar orqali saralash).

Quyida saralashning yaxshilangan usullari va ularning samaradorliklari haqida bayon qilingan.

Bu algoritm “bo‘lib ol va egalik qil” tamoyiliga yaqqol misol bo‘la oladi. Bu algoritm rekursiv bo‘lib, o‘rtacha **N*log$_2$N** ta solishtirish natijasida saralay oladi. Algoritm berilgan massivni saralash uchun uni 2 ga ajratib oladi. Bo‘lib olish uchun ixtiyoriy elementni tanlab, uni 2 ta qismga ajratadi. Lekin, o‘rtadagi elementni tanlab, massivning teng yarmidan 2 ga ajratgan ma’qul bo‘ladi. Tanlangan kalit elementga nisbatan chapdagi va o‘ngdagi har bir element solishtiriladi. Kalit elementdan kichiklari chapga, kattalari esa o‘ng tomonga o‘tkaziladi. Endi massivning har ikkala tomonida ham xuddi yuqoridagi amallar takror bajariladi, ya’ni, bu oraliqlarning o‘rtasidagi elementlar faqat kalit sifatida qaraladi.

Misol uchun rasmdagi massivni saralash algoritmini qarab chiqaylik.

1. Oraliq sifatida **0** dan **n-1** gacha bo‘lgan massivning barcha elementlari olinadi.

2. Oraliq o‘rtasidagi kalit element tanlanadi, ya’ni **key(<oraliq_boshi><oraliq_oxiri>)/2, i<oraliq_boshi>, j<oraliq_oxiri>** ko‘rinishida bo‘ladi**.**

1-rasm. Quicksort algoritmini o‘rinlashtirish

3. Chapdagi i-elementni **key** bilan solishtiriladi. Agar **key** kichik bo‘lsa, keyingi qadamga o‘tadi. Aks holda **i** esa shu qadamni o‘zida takrorlanadi.

4. O‘ngdagi **j**-element bilan **key** solishtiriladi. Agar **key** katta bo‘lsa, keyingi qadamga o‘tadi, aks holda **j--** va shu qadamni o‘zida takrorlanadi.

5. Bunda esa **i-** va **j**-elementlarning o‘rni almashtiriladi. Agar **i<j** bo‘lsa, 3-qadamga o‘tiladi. Birinchi o‘tishdan keyin tanlangan element o‘zining joyiga kelib joylashib oladi.

6. Endi shu ko‘rilayotgan oraliqda **key** kalitning chap tomonida elementlar mavjud bo‘lsa, ular ustida yuqoridagi amallarni bajarish lozim

bo‘ladi, ya’ni ko‘riladigan oraliq **0** dan **key-1** gacha deb belgilanadi hamda u 2-qadamga o‘tiladi. Aks holda keyingi qadamga o‘tiladi.

7. Endi shu ko‘rilayotgan oraliqda **key** kalitning o‘ng tomonida elementlar mavjud bo‘lsa, ular ustida yuqoridagi amallarni bajarish lozim bo‘ladi, ya’ni ko‘riladigan oraliq **key 1** dan **n-1** gacha deb belgilanadi va 2-qadamga o‘tiladi. Aks holda algoritm shu yerda tugaydi.

Bu algoritmga misol ko‘rib chiqaylik. Masalan: Talaba ism va sharifi hamda tartib raqamidan iborat bo‘lgan jadvalni quicksort algoritmi bilan saralab unga nechta o‘rin almashtirish amalga oshirilgani aniqlanadi.

C dasturlash tilida dastur kodini ko‘raylik.

```
#include <cstring>
 #include <iostream>
using namespace std;
struct table { int t;
    string FIO;  }; int q0;
 void qs(table *a,int first,int last) {
   int i  first, j  last;table x a[(first  last) / 2];
     do { wxile (a[i].FIO < x.FIO) i;
         wxile (a[j].FIO > x.FIO) j--;  if(i < j) {
  if (i < j) { swap(a[i], a[j]);q;} i; j--;   }
} wxile (i < j);
  if (i < last) qs(a,i,last);
   if (first < j)  qs(a,first,j);
} int main(int args, char *argv[]){ int n;
 cout<<"n";
 cin>>n;
  table talaba[n];
      for(int i0;i<n;i) { talaba[i].ti1;
        cin>>talaba[i].FIO;
```

```
} qs(talaba,0,n-1);
   for(int i0;
i<n;
i) cout<<talaba[i].t<<" "<<talaba[i].FIO<<endl;
  cout<<"quicksort algoritmi "<<q<<" ta o‘rin alamashtirish saralandi\n";
system("PAUSE");   }
```

Birlashtirish orqali (MergeSort) saralashga doir misollar yechish

MergeSort O(NlogN) da ishlaydigan optimal saralash algoritmlaridan biridir. Eng yaxshi holatda ham eng yomon holatda ham O(NLogN) assimptotikada ishlay oladi. U bunda sonlar ketma-ketligini buzmagan holda barqaror saralay oladi.

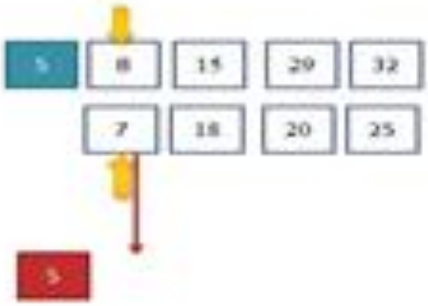
Masalan: sonlardan tashqari ularning kalit sonlari bo‘lsa, kalit soni 3 bo‘lgan 2 soni kalit soni 5 bo‘lgan 2 sonidan oldin kelsa, MergeSortda saralangandan so‘ng ham kalit soni 3 bo‘lgan 2 soni oldin kelaveradi. QuickSort da har doim ham bu shart bajarilmaydi. Bu yerda algoritmning ishlash prinsipi quyidagicha bo‘ladi. Bunda massiv teng ikkiga bo‘linadi. O‘ng tomoni alohida saralanadi, chap tomonida esa alohida saralash o‘tkaziladi. Shundan so‘ng, ikkita saralangan qismni O(n), ya’ni n ta operatsiyada qo‘shib, to‘liq saralangan massiv olinadi.

Aynan shu operatsiya, ya’ni qo‘shish Mergening hisobiga algoritm shunday nom olgan. Endi chap va o‘ng tomonlarini saralash uchun ham ularni ikkiga bo‘lib, xuddi shu ish bajariladi. Massivni bo‘lishni to unda 2 ta element qolguncha davom ettirib boriladi.

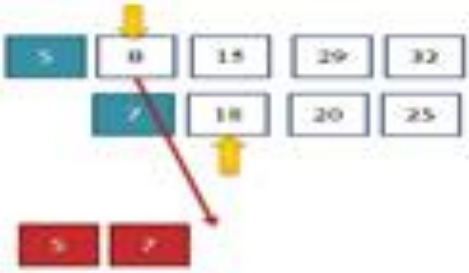
Faraz qilaylik, Merge funksiyasi berilgan bo‘lsin. Bunda a massivning L - Mid qismi saralangan, Mid 1 - R qismi saralanmagan. Shu qismlarni qo‘shib, L – R kesmada saralangan hosil qilish kerak.

Bularni Mid L va R kesma o‘rtasida qaraylik.

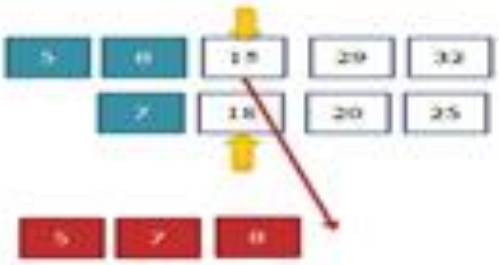
2-rasm. Misol



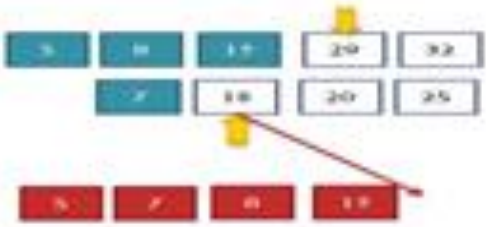
3-rasm. Misol



4-rasm. Misol

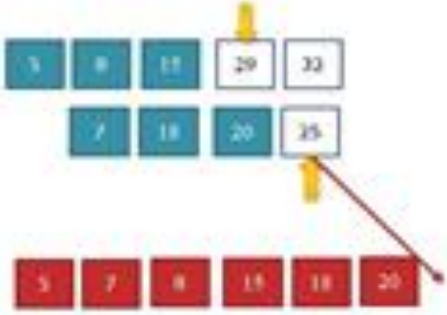


5-rasm. Misol

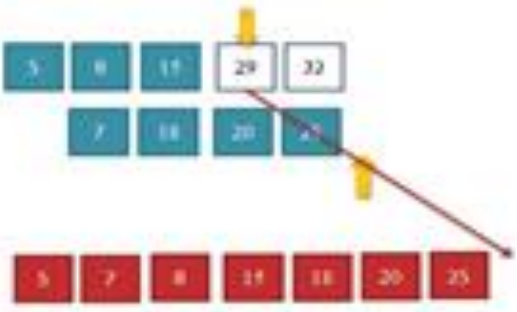


6-rasm. Misol

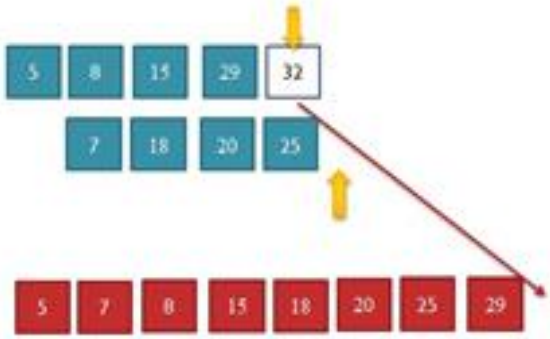
7-rasm. Misol



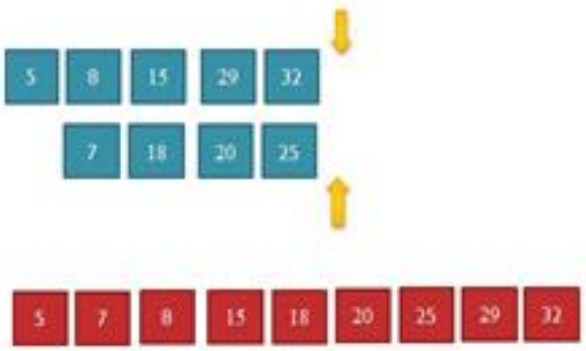
8-rasm. Misol



9-rasm. Misol

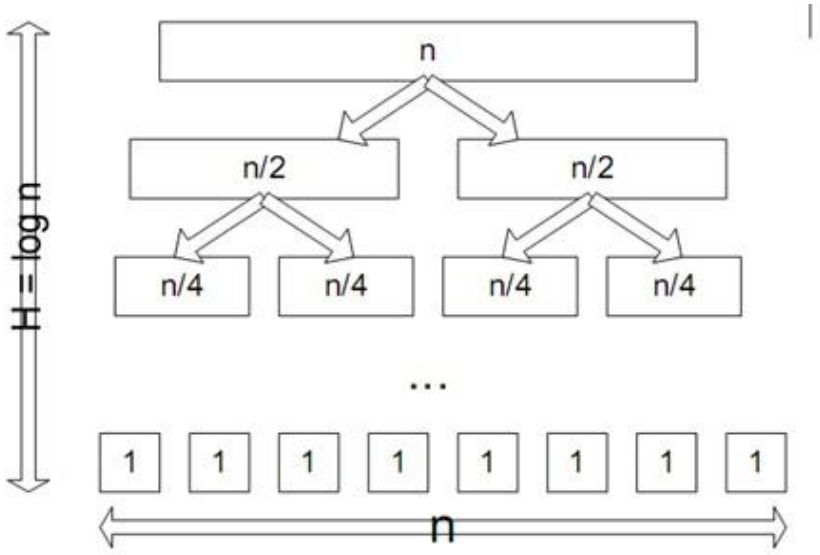


10-rasm. Misol



11-rasm. Misol

[L, R] oraliq m(LR) / 2 o‘rtasi orqali ikkita [L, m] va [m1, R] oraliqqa ajratiladi va ular alohida saralanib boriladi.



12-rasm. Merge funksiya asosida MergeSort

Shunday Merge funksiya asosida MergeSortni quyidagicha yoziladi.

```
#include<iostream>
 using namespace std;
int helper[1000001];
 void Merge(int a[],int left,int middle,int right){
 for(int ileft,jmiddle,kleft;k<right;k)
{
if(imiddle) { helper[k]a[j];continue;
```

```
}
 if(jright ){  helper[k]a[i];continue;
}
 helper[k]( a[i]< a[j])?a[i]: a[j];
 }
for(int kleft;k<right;k)a[k]helper[k];
 }
void MergeSort(int a[],int left,int right)
{
 if(right-left1)return;
 int middle(leftright)>>1;
 MergeSort(a,left,middle);
MergeSort(a,middle,right);
Merge(a,left,middle,right);
 }
int main()
 {
int a[10000], n , i;
cin>>n;
 for(i0;i<n;i) cin>>a[i];
 MergeSort(a,0,n);
for(i0;i<n; i) cout<<a[i]<<" ";
return 0;
}
```

Shell saralashi (qisqarib boruvchi qadamlar orqali saralash)

Shell saralashi – bu qisqarib boruvchi qadamlar orqali saralashga kiradi. To‘g‘ri qo‘shish usulini 1959-yilda D. Shell tomonidan mukammallashtirishda taklif etilgan. Quyidagi rasmda ushbu usul tasvirlangan:

Saralash	44	55	12	42	94	18	6	67
Keyin to'rtlik	44	18	6	42	94	55	12	67
Ikkilik	6	18	12	42	44	55	94	67
Yakkalik	6	12	18	42	44	55	67	94

13-rasm. Shell saralashi usuli

Boshida bir-biridan to‘rtta qadamda joylashgan elementlar o‘zaro guruhlanib, saralash amalga oshiriladi. Bunday jarayon to‘rtlik saralash deb ham ataladi. Birinchi o‘tishdan keyin elementlar qayta guruhlanib, endi har ikki qadamdagi elementlar taqqoslanadi. Bu esa ikkilik saralash deb nomlanadi va nihoyat, uchinchi o‘tishda oddiy yoki yakkalik saralashi amalga oshiriladi.

Bir qarashda mazkur usul bilan saralash amalga oshirilganda saralash jarayoni kamayish o‘rniga ortib boradigandek tuyulsada, elementlarni o‘rin almashtirishlar soni nisbatan kam amalga oshiriladi.

Ko‘rinib turibdiki, bu usul natijasida tartiblangan massiv hosil bo‘lib, har bir o‘tishdan keyin saralashlar kamayib boraveradi. Eng yomon holatida esa oxirgi ishni yakkalik saralash amalga oshiradi.

Baryer usulidan foydalanilganda har bir saralash o‘zining baryeriga ega bo‘lishi lozim bo‘ladi. Bunda dastur uning joyini aniqlashi uchun iloji boricha baryeri osonlashtirishi kerak bo‘ladi. Shuning uchun bu yerda massivni [-h1..N] gacha kengaytirish lozim bo‘ladi.

h[1..t] – qadamlar o‘lchami massivi;

a[1..n] - saralanayotgan massiv;

k – saralash qadami;

x – qo‘shilayotgan element qiymati;

C dasturlash tilida dasturi:

```
#include <iostream>
#include <vector>
// Massivdagi elementlarni almashtirish funksiyasi
```

```
void swap(int& a, int& b)
 {
 int temp  a;
a  b;
     b  temp;
  }
// Shell Sort algoritmini amalga oshirish
 void shellSort(std::vector<int>& arr)
{
int n arr.size();
   std::vector<int> gap  {7, 3, 1};
    for (int i 0; i < gap.size(); i)
 {
  int gapValue gap[i];
   for (int j gapValue; j < n; j)
 {
        int key arr[j];
       int k j - gapValue;
        wxile (k > 0 && arr[k] > key)
{
       arr[k  gapValue]  arr[k];
          k - gapValue;
  }
       arr[k  gapValue]  key;
   }
    }
 }
int main()
{
```

```
 std::vector<int> arr  {12, 34, 54, 2, 3};
  shellSort(arr);
std::cout << "tartiblangan massiv: ";
  for (int num : arr)
{
std::cout << num << " ";
  }
std::cout << std::endl;
 return 0;
}
```

Umuman olganda, tanlagan qadamlardan yaxshi natija olinmagan bo‘lsada, lekin bu qadamlarning biri ikkinchisini ko‘paytuvchisi bo‘lmasligi aniqlangan. D. Knut qadamlarni quyidagicha ketma-ketligini taklif qilgan (teskari tartibda):

1,3,7,15,31,…,ya’ni: $h_{m-1}2$ h_m1, h_t 1, t $[\log_2 n] - 1$. Agar qadamlar ushbu ko‘rinishda aniqlansa, algoritm samaradorligi tartibi **O**($n^{1,2}$) ga teng bo‘ladi.

Mavzuga oid test savollari

1. Saralash samaradorligi qaysi me’zonlar yordamida aniqlanadi?

a) taqqoslashlar va almashtirishlar soni orqali

b) dastur yozishga ketgan vaqt oralig‘ida

c) ishlatilayotgan identifikatorlar soni va turlari yordamida

d) amallar sonini oshirish yordamida

2. Qanday saralash usullari qat’iy usullar deb belgilangan?

a) To‘g‘ridan-to‘g‘ri qo‘shish; to‘g‘ridan-to‘g‘ri tanlash; to‘g‘ridan-to‘g‘ri almashtirish

b) Birlashtirish saralashlari

c) Birlashtirish saralashi; to‘g‘ridan-to‘g‘ri tanlash; to‘g‘ridan-to‘g‘ri almashtirish

d) Tez saralash, to‘g‘ridan-to‘g‘ri tanlash orqali; to‘g‘ridan-to‘g‘ri almashtirish

3. Qanday saralash usullari yaxshilangan usullar deb belgilangan?

a) Tez saralash

b) Shell saralashi

c) Birlashtirish saralashi

d) Tez saralash, to‘g‘ridan-to‘g‘ri tanlash; to‘g‘ridan-to‘g‘ri almashtirish

4. Operativ xotirada bajariladigan saralash qanday ataladi?

a) ichki saralash

b) to‘liq saralash

c) qo‘shish orqali saralash

d) manzillar jadvalini saralash

5. Saralash usullari orasidan noto‘g‘risini toping?

a) dinamik

b) yaxshilangan

c) logarifmik

d) qat’iy

6. Saralashning qaysi usullari, (N^2) kalitlarni taqqoslash tartibiga ega?

a) qat’iy

b) binar

c) yaxshilangan

d) logarifmik

Nazorat savollari

1. Yuqoridagi usullarning bir-biridan farqini aytib bering.

2. Qaysi saralash usuli eng samarali bo‘lib hisoblanadi?

3. Shell usuli qaysi asosiy saralash usuliga tegishli?

Xulosa

Saralash bir necha xil saralash algoritmlari yordamida amalga oshiriladi. Bularga quiksort – tez saralash usuli, birlashtirish orqali (MergeSort) saralash, shell saralash (qisqarib boruvchi qadamlar orqali saralashlari) kiradi. Bunday saralashlar saralashning yaxshilangan usullari hisoblanadi. Bu usullar yordamida ma‘lumotlarni saralash anchagina ma‘lumotlarni qidirishning samaradorligini oshiradi. Bunday qidirishni C++ dasturi yordamida bir nechta misollarda ko‘rib ijobiy natijalar olindi. Shell usuli natijasida tartiblangan massiv hosil bo‘lib, har bir o‘tishdan keyin saralashlar kamayib boraveradi. Eng yomon holatida esa oxirgi ishni yakkalik saralash amalga oshiradi. Baryer usulidan foydalanilganda har bir saralash o‘zining baryeriga ega bo‘lishi lozim bo‘ladi.

2-BOB. CHIZIQLI MA’LUMOTLAR TUZILMALARI

1-§. Chiziqli ma’lumotlar tuzilmalari

1.1. Statik va dinamik massivlar. Chiziqli konteynerlar. Iteratorlar va ularning turlari

Reja:

1. Statik va dinamik massivlar va ularga misollar keltirish.
2. Chiziqli konteynerlar va ularni qo‘llash.
3. Iteratorlar va ularning turlari.

Tayanch iboralar: statik massivlar, dinamik massivlar, chiziqli konteynerlar, iteratorlar.

Statik va dinamik massivlar va ularga misollar keltirish

Massiv tushunchasi. Massiv – bu bir tipli nomerlangan ma’lumotlar jamlanmasidir. Massiv o‘zgaruvchi indeks tushunchasiga mosdir. Massiv ta’riflanganda uning nomi, tipi va indekslar chegarasi namoyon qilinadi. Masalan:

type turidagi length ta elementdan iborat a nomli massiv shu holatda e’lon qilinadi:

type a [length];

Bu maxsus a[0], a[1], ..., a[length -1] nomlarga ega bo‘lgan type turidagi o‘zgaruvchilarning e’lon qilinishiga mos keladi.

Massivning har bir elementi o‘z raqamiga yoki indeksga egadir. Massivning xnchi elementiga murojaat indekslash operatsiyasi yordamida bajariladi:

```
int x...;
   //butun sonli indeks
TYPE valuea[x];
//ch-nchi elementni oshirish a[x]value;
   //x-yxb elementga joylash
```

Indeks sifatida butun tur qiymatini qaytaradigan har qanday ifoda istalgan vaqtda qo‘llanishi mumkindir: char, short, int, long. C dasturlash tilida massiv elementlarining indekslari 0 dan boshlanadi, length elementdan iborat bo‘lgan massivning oxirgi elementining indeksi esa bu length -1. Massivning int z[3] shakldagi ta’rifi, int tipiga tegishli z[0], z[1], z[2] elementlardan iborat massivni anglatadi.

Massiv chegarasidan tashqariga chiqish (ya’ni mavjud bo‘lmagan elementlar ustida amallar bajarish) dastur bajarilishida kutilmagan natijalarga olib kelishi ehtimoli mavjud. Shuni ta’kidlab o‘tish kerakki, bu eng ko‘p tarqalgan xatolardan biri bo‘lib hisoblanadi.

Agar massiv tahrirlanganda elementlar chegarasi ko‘rsatilgan bo‘lsa, u holda ro‘yxatdagi elementlar soni bu chegaradan kam bo‘lishi ham mumkin, lekin ortiq bo‘lishi mumkin emasdir.

Misol uchun int a[5]{2,-2}. Bu holda a[0] va a[1] qiymatlari aniqlangan bo‘lib, mos holda 2 va –2 ga tengdir. Agar massiv uzunligiga qaraganda kamroq element berilgan bo‘lsa, qolgan elementlar 0 hisoblanadi:

```
int a10[10]{1, 2, 3, 4};    //va 6 ta nol
```

Agar nomlangan massivning tavsifida uning o‘lchamlari ko‘rsatilmagan bo‘lsa, kompilyator tomonidan massiv chegarasi avtomatik aniqlanadi:

```
int a3[]{1, 2, 3};
```

Bir o‘lchamli massivlarni funksiya parametrlari sifatida uzatish

Massivdan funksiya parametri sifatida foylalanganda, funksiyaning birinchi elementiga ko‘rsatkich beriladi, ya’ni massiv hamma vaqt manzil bo‘yicha yetkaziladi.

Bunda massivdagi elementlarning miqdori haqidagi axborot unutiladi, shuning uchun massivning o‘lchamlari haqidagi ma’lumotni alohida parametr sifatida uzatish lozim bo‘ladi.

Funksiyaga massiv boshlanishi uchun ko‘rsatkich uzatilgani tufayli (manzil bo‘yicha uzatish), funksiya tanasining operatorlari hisobiga massiv o‘zgarishi ham mumkin bo‘ladi.

Funksiyalarda bir o‘lchovli sonli massivlar argument sifatida ishlatilganda ularning chegarasini ko‘rsatish uncha ham muhim hisoblanmaydi.

Ko‘p o‘lchovli massivlarda ta’rif. Ikki o‘lchovli massivlar matematikada matritsa yoki jadval tushunchasiga mos bo‘ladi.

Jadvallarning tahrirlash qoidasi, ikki o‘lchovli massivning elementlari massivlardan iborat bo‘lgan bir o‘lchovli massiv ta’rifiga asoslangan bo‘ladi.

Misol uchun ikki qator va uch ustundan iborat bo‘lgan haqiqiy tipga tegishli d massiv boshlang‘ich qiymatlari quyidagicha bo‘lishini ko‘rsatish mumkin:

float d[2][3]{(1,-2.5,10),(-5.3,2,14)};

Bu yozuv quyidagi qiymat berish operatorlariga mos bo‘ladi:

d[0][0]1;d[0][1]-2.5;d[0][2]10;

d[1][0]-5.3;d[1][1]2;d[1][2]14;

Bu qiymatlarni bitta ro‘yxat bilan hosil qilish ham mumkin:

float d[2][3]{1,-2.5,10,-5.3,2,14};

Tahrirlash yordamida boshlang‘ich qiymatlar aniqlanganda massivning hamma elementlariga qiymat berish shart bo‘lmaydi.

Masalan: int x[3][3]{(1,-2,3),(1,2),(-4)}.

Bu yozuv quyidagi qiymat berish operatorlariga xosdir:

x[0][0]1;x[0][1]-2;x[0][2]3;

x[1][0]-1;x[1][1]2;x[2][0]-4;

Tahrirlash yordamida boshlang‘ich qiymatlar aniqlanganda massivning birinchi indeks chegarasini ko‘rsatilishi shart emas, lekin qolgan indekslar chegaralari ko‘rsatilishi lozimdir.

Masalan:

double x[][2]{(1.1,1.5),(-1.6,2.5),(3,-4)}

Bu misolda avtomatik ravishda qatorlar soni uchga teng deb hisoblanadi.

Funksiyaga ko‘p o‘lchamli massivlarni uzatishga e’tibor qarataylik. Bunda ko‘p o‘lchamli massivlarni funksiyaga uzatishda barcha o‘lchamlar parametrlar sifatida uzatilishi muhim deb qaraladi. C dasturlash tilida ko‘p o‘lchamli massivlar aniqlanishi bo‘yicha mavjud emas. Agarda bir nechta indeksga ega bo‘lgan massivlarni tavsiflasak (masalan, int mas [3][4]), bu yerda int [4] bir o‘lchamli massivlar ega bo‘lgan elementlari majmuidir.

Misol uchun kvadrat matritsani uzatishni ko‘rib chiqaylik. Agar void transp(int a[][],int n){.....} funksiyasining ismini aniqlasak, bu holda funksiyaga noma’lum o‘lchamdagi massivni uzatish to‘g‘ri keladi.

Aniqlanishiga ko‘ra, massiv har doim bir o‘lchamli bo‘lishi kerak hamda uning elementlari bir xil uzunlikda bo‘lishi lozim. Shu vaqtgacha massivni uzatishda uning elementlari o‘lchamlari haqida biror joyda biron narsa aniq qilib keltirilmagan, shuning uchun ham ko‘p hollarda kompilyatorlar ba’zi masalalarni xato chiqarib beradi.

Bu muammoning eng sodda yechimi funksiyada quyidagicha aniqlik kiritishdir:

void transp(int a[][4],int n){.....}, bu holda har bir satr o‘lchami to‘rt bo‘ladi,

massiv ko‘rsatkichlarining o‘lchami esa hisoblab aniqlanadi.

Ko‘p o‘lchamli massivlar va ko‘rsatkichlar. C dasturlash tilida massivning eng umumiy tushunchasi - bu ko‘rsatkichdir, bunda har xil turdagi ko‘rstakich bo‘lishi tabiiydir, ya’ni massiv har qanday turdagi elementlarga, shu jumladan, massiv bo‘lishi mumkin bo‘lgan ko‘rsatkichlarga ham ega bo‘lishi lozim. O‘z tarkibiga ko‘ra, boshqa massivlarga ham ega bo‘lgan massiv ko‘p o‘lchamli deb hisobga olinadi.

Bunday massivlarni e’lon qilishda kompyuter xotirasida bir nechta turli xildagi ob’ekt yaratiladi.

Ko‘rsatkichlar massivlari. Bu massivlar quyidagicha ta’riflanadi:

<tip> *<nom>[<son>]

Masalan: int *pt[6] ta’rif int tipidagi ob’ektlarga olti elementli massivni joylaydi.

Ko‘rsatkichlar massivlari satrlar va masssivlarini tasvirlash uchun eng maqbul yechimdir.

Masalan: familiyalar ro‘yxatini kiritish uchun ikki o‘lchovli massivdan foydalanish zarur. char fam[][20]{ "Olimov","Raximov","Ergashev"} bu xotirada 60 elementdan iborat bo‘ladi, chunki har bir familiya 20 gacha 0 lar bilan to‘ldirilib boriladi.

Ko‘rsatkichlar massivi yordamida bu massivni quyidagicha ta’riflash mumkin.

char *pf[] { "Olimov","Raximov","Ergashev"}.

Bu holda ro‘yxat xotirada 23 ta elementdan iborat bo‘ladi, chunki har bir familiya oxiriga 0 belgisi qo‘yib boriladi.

Har xil chegarali jadvallar bilan funksiyalardan foydalanishning bir yo‘li bu oldindan kiritiluvchi o‘zgarmaslardan foydalanishdir. Lekin, asosiy yo‘li ko‘rsatkichlar massivlaridan foydalanishdir.

Bir o‘lchovli dinamik massivlar. C dasturlash tilida o‘zgaruvchilar yo o‘zgarmas tarzda kompilyatsiya paytida yoki standart kutubxonadan funksiyalarni chaqirib olish yo‘li bilan o‘zgaruvchanlik tarzda - dasturni bajarish paytida joylashtirilishi lozim.

Asosiy farq ushbu usullarni qo‘llashda bilinadi, bu esa ularning samaradorligi va moslashuvchanligida yaqqol seziladi. Bunda ko‘pincha statik joylashtirish samaraliroq kechadi. Chunki, bunda xotirani ajratish dastur bajarilishidan oldin sodir bo‘ladi. Biroq, bu usulning moslashuvchanligi ancha past ko‘rsatkichni beradi, chunki bunda joylashtirilayotgan ob’ektning turi va o‘lchamlarini avvaldan bilish kerak bo‘ladi.

Masalan, matnli faylning ichidagisini satrlarning statik massivida joylashtirish qiyin kechadi, avvaldan uning o‘lchamlarini bilish kerak bo‘ladi.

Bunda noma’lum sonli elementlarni oldindan saqlash va ishlov berish kerak bo‘lgan masalalar, odatda xotiraning dinamik ajratilishini talab etadi.

Bunda xotirani dinamik va statik ajratishlar o‘rtasidagi asosiy farq quyidagicha kechadi:

- statik ob’ektlar nomlangan o‘zgaruvchilar bilan belgilanadi, hamda ushbu ob’ektlar o‘rtasidagi amallar to‘g‘ridan-to‘g‘ri, ularning nomlaridan foydalangan holda amalga qo‘llaydi. Dinamik ob’ektlar o‘z shaxsiy otlariga ega bo‘lmaydi va ular ustidagi amallar bilvosita, ko‘rsatkichlar yordamida, amalga oshirilib boriladi;
- statik ob’ektlar uchun xotirani ajratish va bo‘shatish kompilyatorlar tomonidan avtomatik tarzda amalda tadbiq qilinadi. Bunda dasturchi bu haqda o‘zi qayg‘urishi kerak emas. Statik ob’ektlar uchun xotirani ajratish va bo‘shatish to‘laligicha dasturchi zimmasiga yuklatiladi, xolos. Bu anchagina qiyin masala va uni yechishda xatoga yo‘l qo‘yish juda ham oson kechadi.

Dinamik tarzda ajratilayotgan xotira ustida turli xatti-harakatlarni amalga oshirish uchun new va delete operatorlaridan foydalaniladi.

Ma’lum bir turdagi elementlardan tashkil topgan berilgan o‘lchamlardagi massivga xotira ajratish uchun new operatoridan foydalanish mumkin bo‘ladi:

int *pianew int[4];

Bu misolda xotira int turidagi to‘rtta elementdan iborat massivga xotiradan joy ajratiladi. Afsuski, new operatorining bu shakli massiv elementlarini nomlantirish imkoniyatiga yo‘l ochib beradi.

Dinamik massivni bo‘shatish uchun delete operatoridan foydalanish lozim: delete[] pia;

Agar ajratilgan xotirani bo‘shatish esdan chiqqudek bo‘lsa, bu xotira bekordan-bekorga sarflana boshlaydi, undan foydalanilmay qolinadi, biroq, agar uning ko‘rsatkichi o‘z qiymatini o‘zgartirgan bo‘lsa, uni tizimga qaytarish mumkin bo‘lmay qoladi.

Bu hodisa xotiraning yo‘qotilishi degan maxsus nom bilan ataladi. Pirovard natijada, ya’ni dasturda xotira yetishmagani tufayli kiritilgan dastur kodi natija bermaydi va shu bilan holat yakunlanadi.

Ikki o‘lchovli dinamik massivlar. Matritsani shakllantirishda oldin bir o‘lchovli massivlarga ko‘rsatuvchi ko‘rsatkichlar massivi uchun xotiradan joy ajratiladi, shundan so‘ng, parametrli ssiklda bir o‘lchovli massivlarga xotiradan joy ajratiladi.

Masalan:

```
int n;
cin>>n; double *matr[100];
 for (i0;i<n;i) matr[i]new int[n];
```

Xotirani bo‘shatish uchun bir o‘lchovli massivlarni bo‘shattiruvchi ssiklni bajarish muhimdir.

```
for(int i0;i<n;i)
delete matr[i];
```

Chiziqli konteynerlar va ularni qo‘llash

Biblioteka yadrosi uchta elementdan iborat bo‘ladi: konteynerlar, algoritmlar va iteratorlar.

Konteynerlar (containers) – bu boshqa elementlarni saqlovchi ob’ektlardir. Masalan, chiziqli ro‘yxat, vektor va to‘plam.

Assotsiativ konteynerlar (associative containers) – kalitlar yordamida ularda saqlanadigan qiymatlarni tezkor olish imkonini yaratib beradi.

Bu yerda har bir sinf konteynerida ular bilan ishlash uchun mo‘ljallangan funksiyalar to‘plami aniqlangan. Masalan, chiqarish, ro‘yxat elementlarni kiritish va qo‘shish funksiyalarni o‘z ichiga qamrab oladi.

Algoritmlar (algorithms) – bu konteyner ichidagilar ustidan amallar bajaradi. Konteyner ichidagilarni tahrirlash, saralash, qidirish va almashtirish uchun maxsus algoritmlar mavjud.

Ko‘p algoritmlar konteyner ichidagi elementlarni, chiziqli ro‘yxatini ifodalaydovchi ketma-ketliklar (sequence) bilan ishlash uchun yaratilgan.

Iteratorlar (iterators) – bu konteynerga nisbatan ko‘rsatkich sifatida ishlaydigan ob’ektdir. Ular massiv elementlariga ruxsat oluvchi ko‘rsatkichlar kabi konteyner ichidagiga ruxsat olish imkonini yaratadi.

Sinf-konteynerlar. STL da quyidagi sinf-konteynerlari aniqlab olingan:

Asosiy konteynerlar:

- **vektor** <vektor.h> dinamik massiv;
- **list** <list.h> chiziqli ro‘yxat;
- **deque** <deque.h> ikki tarafli navbat;
- **set** <set.h> to‘plam;
- **multiset** <set.h> Har bir elementi noyob bo‘lishi shart emas;
- **map** <map.h> kalit/ qiymat juftlikni saqlash uchun assotsiativ ro‘yxat. Bunda har bir kalit bitta qiymat bilan bog‘langandir;
- **multimap** <map.h> har bir kalit bilan ikkita yoki ko‘proq qiymatlar bog‘langan.

Hosila konteynerlar:

- **stack** <stack.h> stek;
- **queue** <queue.h> navbat;
- **priority_queue** <queue.h> bu esa birinchi o‘rindagi navbat.

Konstruktorlar. Ixtiyoriy sinf-konteyner ko‘rsatilmagan holda konstruktor va destruktorni nusxalovchi konstruktorlarga ega.

Ixtiyoriy ob’ekt uchun ko‘rsatilmagan holda konteynerda saqlanuvchi konstruktor mavjud bo‘lishi zarur. Undan tashqari ob’ekt uchun < va operatorlar aniqlanish lozim.

1-jadval

Misol uchun vektor sinf-konteynerning konstruktori va destruktori

vector<elem> c	bitta ham elementga ega bo‘lmagan bo‘sh vektorni yaratish;
vector<elem> c1(c2)	ko‘rsatilgan tipdagi boshqa vektorning nusxasini yaratadi (barcha elementlarni nusxasini oladi);
vector<elem> c(n)	konstruktor orqali ko‘rsatilmagan holda yaratilgan n elementli vektorni yaratiladi;
vector<elem> c(n,x)	x elementning n nusxalari yordamida tahrirlangan vektorni yaratiladi;
~vector<elem>()	barcha elementlarni o‘chiradi va xotirani bo‘shatadi

Iteratorlar va ularning turlari

Iteratorlar bilan ko‘rsatkichlar kabi ishlash mumkin. Ularga *, inkrement, dekrement operatorlarni qo‘llash mumkin bo‘ladi. Iterator tipi sifatida har xil konteynerlarda aniqlangan iterator tip e’lon qilinadi.

Bugungi kunda iteratorlarning beshta tipi mavjud:

1. Kiritish iteratorlar (input_iterator) tenglik, nomini o‘zgartirish va inkrementa operatsiyalarni amalga oshiradi.

, !, *i, i, i, *i

Kiritish iteratsiyasining maxsus holati istream_iteratoridan iborat.

2. Chiqarish iteratorlar (output_iterator) o‘zlashtirish operatorning chap tarafidan imkoni bo‘lgan ismni o‘zgartirish va inkrement operatsiyalari qo‘llanadi. i, i, *it, *it

Chiqarish iteratsiyasining maxsus holati ostream_iteratoridir.

3. Bitta yo‘nalishdagi iteratorlar (forward_iterator) kiritish/chiqarish operatsiyalarning barchasini qo‘llaydi va bundan tashqari chegarasiz o‘zlashtirishning imkoniyatini yaratadi.

, !, , *i, i, i, *i

4. Ikki yo‘nalishdagi iteratorlar (biderectional_iterator) forwarditeratorlarning barcha xususiyatlariga ega bo‘lib, bundan tashqari, konteynerni ikkita yo‘nalishi bo‘yicha o‘tish imkonini beradigan qo‘shimcha dekrement (--i, i--, *i--) operatsiyasiga ega bo‘ladi.

5. Ixtiyoriy ruxsatga ega bo‘lgan iteratorlar (random_access_iterator) biderectional-iteratorlarning barcha xususiyatlariga ega bo‘lib, bundan tashqari solishtirish va manzil arifmetikasi operatsiyalarni qo‘llay oladi.

in, in, i-n, i-n, i1-i2, i[n], i1<i2, i1<i2, i1>i2, i1>i2

Shuningdek, STLda teskari iteratorlar ham (reverse iterators) amalga oshiriladi. Ketma-ketlikni teskari yo‘nalishda o‘tuvchi ikki yo‘nalishli yoki ixtiyoriy ruxsatga ega bo‘lgan iteratorlar teskari iteratorlar bo‘lishi shart.

Xotirani taqsimlovchilar, predikatlar va solishtirish funksiyalari

Konteynerlarga va algoritmlarga hamda STLdagi iteratorlarga qo‘shimcha bir nechta standart komponentalar ham qo‘llaniladi. Ulardan asosiylari esa xotira taqsimlovchilari, predikatlari va solishtirish funksiyalaridan iboratdir.

Har bir konteynerda uning uchun aniqlangan va konteyner uchun xotirani belgilash jarayonini boshqaradigan **xotira taqsimlovchisi** (**allocator**) mavjuddir.

Ko‘rsatilmagan holda esa xotira taqsimlovchisi aloqador sinf ob’ekti bo‘ladi. Xususiy taqsimlovchini tavsiflash mumkin.

Ba’zi bir algoritmlar va konteynerlarda muhim tipdagi predikat ataluvchi funksiyalarda ishlatiladi. Predikatlar unar va binar bo‘lishi ham mumkin. U yoki bu qiymatni olish aniq shartlari dasturchi orqali aniqlanadi. Unar predikatlarning tipi – **UnPred**, binar predikatlarning esa - **BinPred**. Argumentlar tipi konteynerda saqlanuvchi ob’ektlar tipiga xos etib tayinlanadi.

Ikkita elementni solishtirish uchun binar predikatlarning maxsus tipi aniqlangan. U **solishtirish funksiyasi** (comparison function) deb nomlanadi. Agarda birinchi element ikinchidan kichik bo‘lsa, unda funksiya rost qiymatni qaytaradi. **Comp** tip funksiya tipidir. STL da ob’ekt-funksiyalar o‘ziga xos ahamiyatga egadir.

Ob’ekt-funksiyalari – bu sinfda «kichik qavslar» () operatsiyasi aniqlangan sinf nusxalaridir. Ba’zi bir hollarda funksiyalarni ob’ekt-funksiyalarga almashtirish qulay deb hisoblanadi. Ob’ekt-funksiya funksiya sifatida ishlatilsa, unda uni chaqirish uchun operator (), operatori ishlatiladi.

Vector-vektor konteynerlari: STL da vektor dinamik massiv sifatida aniqlanadi. Massiv elementlariga indeks orqali ruxsat berib boriladi.

Vektor sinfida quyidagi konstruktorlar aniqlangan:

• Birinchi shakl bo‘sh vektor konstruktorini tavsiflaydi.

• Konstruktor vektorning ikkinchi shaklida elementlar soni – bu son, har bir elementi esa qiymatiga tengdir. Qiymat parametri ko‘rsatilmagan holdagi qiymat parametri bo‘lishi ham mumkin.

• Konstruktor vektorning uchinchi shakli – bu nusxalash konstruktoridir.

• To‘rtinchi shakli – bosh va oxirgi iteratorlar orqali elementlar diapazonini o‘z ichiga olgan konstruktor vektoridir.

Vektorda saqlanadigan ixtiyoriy ob’ekt ko‘rsatilmagan holda konstruktor aniqlash uchun zarurdir. Bundan tashqari, ob’ekt uchun < va operatorlar aniqlanishi shart.

Vektor sinfi uchun quyidagi solishtirish operatorlari mavjuddir:

, <, <, !, >, >.

Bundan tashqari, vektor sinf uchun [] indeks operatori mavjud.

Ikki yo‘nalishli tartib (Deque)

deque – vektor kabi, ixtiyoriy ruxsat iteratorlarni qo‘llovchi ketma-ketlik mavjuddir. Bundan tashqari, u o‘zgarmas vaqtga boshida yoki oxiriga kiritish va

o‘chirish operatsiyalarni qo‘llaydi. O‘rtada kiritish va o‘chirish chiziqli vaqtni egallaydi. Xotirani boshqarishiga ishlov berish esa vektorlar kabi avtomatik ravishda bajariladi.

Ro‘yxat(List)

Ro‘yxat – ikki yo‘nalishli iteratorlarni qo‘llaydigan hamda kiritish va o‘chirish operatsiyalarni o‘zgarmas vaqtda ketma-ketlikni ixtiyoriy joyida bajaradigan, shuningdek, xotirani boshqarishiga avtomatik ravishda ishlov beruvchi ketma-ketlik ko‘rinishi.

Vektorlar – bu ikki tarafli tartiblangan tartibdan farqi shundaki, elementlar ro‘yxatiga tez va ixtiyoriy ruxsat qo‘llanmaydi, lekin ko‘pgina algoritmlarga esa ketma-ketlik ruxsatlar kerak bo‘ladi.

Assotsiativ konteynerlar (massivlar)

Assotsiativ massiv juft qiymatlardan iborat. Kalit (key) deb atalgan bitta qiymatni bilib (mapped value) aks etuvchi qiymat deb atalgan ikkinchi qiymatga ruxsat olish mumkin. Assotsiativ massivni massiv indekslari butun tiplardan iborat bo‘lmagan massiv sifatida tavsiflash ham mumkin:

V& operator[](const K&) K ga mos keluvchi V ga teng ilovani qaytaradi.

Assotsiativ konteynerlar – bu assotsiativ massivning umumiy tushunchasidir.

map assotsiativ konteyner − bu kalit yordamida qiymatga tez ega bo‘lish imkonini yaratadigan juftlik (kalit, qiymat) ketma-ketligidir. Map konteyneri ikki yo‘nalishli iteratorni tavsif etadi.

Map assotsiativ konteyneri kalit tiplari uchun “<” operatsiyasi mavjudligini talab qiladi. U kalit bo‘yicha saralangan o‘z elementlarini saqlaydi. Saralash almashuvi esa tartib bo‘yicha bajarilib boriladi.

Map sinfida quyidagi konstruktorlar aniqlangan: birinchi shakli bo‘sh assotsiativ konteynerning konstruktorini tavsiflaydi, ikkinchi shakli esa – konstruktor nusxasi, uchinchisi – elementlar diapazonini qamrab olgan assotsiativ konteynerning konstruktoridir.

O‘zgartirish operatsiyasi aniqlangan:

map& **operator**(const map&);

Quyidagi operatsiyalar aniqlangan: , <, <, !, >, >.

mapda kalit, qiymat juftliklar **pair** tiplagi ob’ektlar ko‘rinishida saqlanadi.

Kalit, qiymat juftliklarni faqatgina pair sinf konstruktorlari yordamida, balki pair tipdagi ob’ektlarni yaratuvchi va ma’lumotlar tiplaridan parametrlar sifatida foydalanuvchi **make_pair** funksiya yordamida yaratish mumkin bo‘ladi. Assotsiativ konteyner uchun o‘ziga xos operatsiya – bu ([]) indeksatsiyalash operatsiyasi yordamida assotsiativ qidiruvdir. mapped_type& **operator[]**(const key_type& K); **set** to‘plamini assotsiativ massiv sifatida ko‘rish mumkin. Unda qiymatlar ahamiyatga ega emas, shuning uchun faqat kalitlarni kuzatish mumkin bo‘ladi.

To‘plamlar assotsiativ massiv kabi T tip uchun (<) “kichik” operatsiyani mavjudligini talab etadi. U o‘z elementlarini saralangan holda saqlaydi. Saralash almashuvi esa quyidagi tartibda bajariladi.

Konteyner usullari

Elementlarga ruxsat:

- **front()** birinchi elementga ilova;
- **Back()** oxiri elementga ilova;
- **operator[]**(i) tekshirishsiz indeks bo‘yicha ruxsat;
- **at**(i) indeks bilan tekshirish bo‘yicha ruxsat.
- **front()** ilova birinchi elementga;

Elementlarni kiritish usullari:

- **insert(**p,x**) r** ko‘rsatgan elementdan oldin **x** ni qo‘shish
- **insert(**p,n,x**) r** dan oldin **x** ning **n** nusxalarini qo‘shish usuli
- **insert(**p,first,last**) r** dan oldin **[first:last]** dagi elementlarni qo‘shish

• **push_front(**x**)** yangi birinchi elementni qo‘shish (ikta uchga ega bo‘lgan tartiblar va ro‘yxatlar uchun)

Elementlarni o‘chirish usullari:

• **erase(**p**) r** pozitsiyadagi elementni o‘chirish;

• **erase(**first,last**) [first:last]**dan elementlarni o‘chirish;

• **pop_back()** oxirgi elementni o‘chirish;

• **pop_front()** birinchi elementni o‘chirish (ikkita uchga ega bo‘lgan tartiblar va ro‘yxatlar uchun)

O‘zlashtirish usullari:

• **operator(**x**)** konteynerga **x** konteynerni elementlari o‘zlashtiriladi;

• **assign(**n,x**)** konteynerga **x** elementning **n** nusxasi o‘zlashtiriladi (assotsiativ bo‘lmagan konteynerlar uchun);

• **assign(**first,last**) [first:last]** diapazondagi elementlarni o‘zlashtirish

Assotsiativ usullari:

• **find(elem) elem** qiymatga ega bo‘lgan birinchi elementni pozitsiyasini topadi

• **lower_bound(elem)** element qo‘yish mumkin bo‘lgan birinchi pozitsiya.

• **upper_bound(elem)** element qo‘yish mumkin bo‘lgan oxirgi pozitsiyani topadi

Assotsiativ usullar:

• **operator[](**k**) k** kalitli elementga ruxsat;

• **find(**k**) k** kalitli element pozitsiyasini topadi;

• **lower_bound(**k**) k** kalitli elementning birinchi pozitsiyasini bajaradi;

• **upper_bound(**k**) k**dan katta bo‘lgan kalitli birinchi elementni bajaradi;

• **equal_range(**k**) k** kalitli elementni **lower_bound** (kuyi chegarasini) va **upper_bound** (yuqori chegarasini) bajaradi.

Boshqa usullar:

• **size()** elementlar hajmi;

• **empty()** konteyner bo‘shmi?

• **capacity()** vektor uchun ajratilgan xotira (faqat vektorlar uchun);

• **reserve(**n**) n** elementdan iborat bo‘lgan konteyner uchun xotira ajratadi;

• **swap(x)** ikkita konteynerlarni bir-birini joyini almashtirish;

• , **!**, < solishtirish operatorlari.

Mavzuga oid test savollari

1.Ma’lumotlar tuzilmalari bog‘lanishiga ko‘ra qaysilarga klassifikatsiyalanadi?

a) Bog‘lamli va bog‘lamsiz

b) Statik, yarimstatik va dinamik

c) Chiziqli va chiziqsiz

d) Oddiy va murakkab

2.Ma’lumotlar tuzilmalari vaqt o‘zgaruvchanligi yoki dastur bajarilishi jarayoniga ko‘ra qaysilarga klassifikatsiyalanadi?

a) Statik, yarimstatik va dinamik

b) Chiziqli va chiziqsiz

c) Bog‘lamli va bog‘lamsiz

d) Oddiy va murakkab ko‘rinishlarda

3.**Ma’lumotlar tuzilmalari tartibiga ko‘ra qaysilarga klassifikatsiyalanadi?**

a) Chiziqli va chiziqsiz

b) Statik, yarimstatik va dinamik

c) Bog‘lamli va bog‘lamsiz

d) Oddiy va murakkab

Nazorat savollari

1. Har qanday konteyner qanday konstruktorlarga ega bo‘ladi?

2. Iteratorlar tiplarini ko‘rsatib bering.

3. Assotsiativ massivlar qanday xususiyatlarga ega bo‘ladi?

4. Elementlarga murojaat usullarini ko‘rsatib bering.

5. Elementlarni o‘chirish usullarini ko‘rsatib bering.

Xulosa

Chiziqli ma‘lumotlar tuzilmasini statik va dinamik massivlarsiz tasavvur qilish qiyin. Bu yerda massiv – bu bir tipli nomerlangan ma’lumotlar jamlanmasidir.

Massiv o‘zgaruvchi indeks tushunchasiga mosdir. Massiv ta‘riflanganda uning nomi, tipi va indekslar chegarasi namoyon qilinadi. Massivning har bir elementi o‘z raqamiga yoki indeksiga egadir. Massiv bir va ko‘p o‘lchovli bo‘ladi.

Ko’p o‘lchovli massiv xotiradan kattaroq joy egallaydi. Massivni e’lon qilish bilan kompyuter xotirasidan massiv elementlariga mos joyni ajratadi. Shuning o‘zi dinamik yoki statikka qarab yo’l ochadi.

1.2. Chiziqli bog‘langan ro‘yxatlar. Bog‘langan ro‘yxatlar haqida tushunchalar. Chiziqli bog‘langan ro‘yxatlarni mantiqiy tasvirlash

Reja:

1. Bog‘langan ro‘yxatlar haqida tushunchalar (Dinamik ma’lumotlar).

2. Chiziqli bog‘langan ro‘yxatlarni mantiqiy tasvirlash usullari.

3. Halqasimon bog‘langan ro‘yxatlar va uni C++ dasturda ko‘rish.

Tayanch iboralar: ro‘yxatlar, mantiqiy tasvirlash, klassifikatsiya, dinamik, Lst, ko‘rsatkichlar.

Bog‘langan ro‘yxatlar haqida tushunchalar

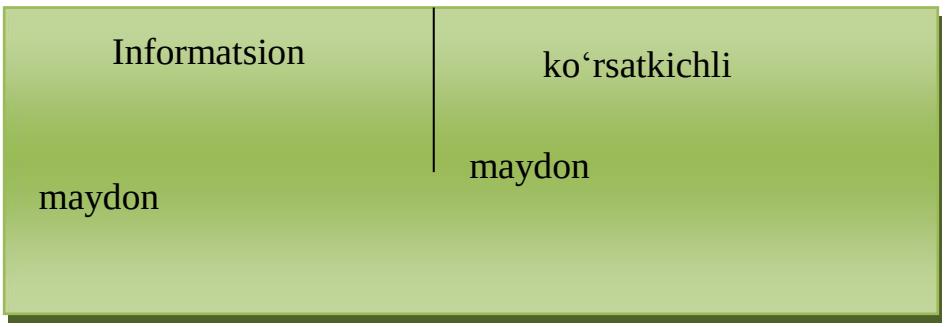
Dinamik ma’lumotlar tuzilmasi. Bu tuzilmada, asosan massivlar (static tuzilmalar) dasturlash tillarida juda foydali va zaruriy tuzilma hisoblanadi. Shunday bo‘lsa-da, uning ikkita kamchiligi ham mavjud:

- uning o‘lchamini dastur bajarilishi mobaynida o‘zgartirib bo‘lmaydi;

- tuzilma orasiga element kiritish uchun qolganlarini surish kerak bo‘ladi.

Bu kamchiliklar bog‘langan ro‘yxatlar bilan ishlanishiga olib keladi. Bu o‘rinda bog‘langan ro‘yxatlar bir xil toifadagi elementlar (tugunlar) ketma-ketligi bo‘lib, ular kompyuter xotirasida turli joylarga joylashtiriladi va o‘zaro bir-biri bilan ko‘rsatkichli maydonlar orqali bog‘lanadi. Ko‘pincha bog‘langan ro‘yxatlarni dasturda turlicha ko‘rinishda amalga oshirish mumkn.

Bog‘langan ro‘yxatlarga elementlar quyidagicha hosil qilinadi.



1-rasm. Bog‘langan ro‘yxatlarning klassifikatsiyasi

Informatsion maydonda foydalanuvchining foydali ma’lumoti yoziladi. Ko‘rsatkichli maydonga keyingi elementning xotiradagi manzili yoziladi. Shunday elementlardan tashkil topadigan tuzilmaga **chiziqli bir bog‘lamli ro‘yxatlar tuzilmasi** deyiladi.

Bog‘langan ro‘yxatlarga massivning kamchiliklari bartaraf qilinganligi sababli **tuzilma uzunligi** va **elementlar orasidagi munosabatlar** dastur bajarilishi mobaynida turli ko‘rinishda o‘zgarib turadi. Bunday o‘zgarish dinamik tuzilma xususiyati hisoblanadi.

Dinamik tuzilma deb, elementlari orasidagi munosabatlar, tuzilma uzunliklari, ya’ni elementlar soni dastur bajarilishi mobaynida o‘zgarib turadigan tuzilmalarga aytiladi.

Dinamik tuzilmalarda elementlar kompyuter xotirasida istalgan joyda joylashishi mumkin. Shu sababli ular orasidagi munosabatlar ko‘rsatkichlar orqali belgilanib olinadi.

Elementlar tuzilmaga kelib qo‘shilgan vaqtda xotiradan bo‘sh joy qidirib topiladi va elementlar o‘sha joyda joylashtiriladi. Aynan shu sababli elementlar xotirada ketma-ket yacheykalarda joylashmagan bo‘lishi ham mumkin. Bu holatda agar fizik xotira tanqisligi sezilmasa, tuzilma uzunligi oshirilishi mumkin bo‘ladi.

Bunday tuzilmalar bilan ishlashning o‘ziga yarasha afzalliklari va kamchiliklari ham mavjud. Afzalligi shundaki, tuzilma uzunligiga oldindan hech qachon chegara qo‘yilmaydi. Unga element kiritish va o‘chirish amallari massivga qaraganda osonroq bo‘ladi. Chunki, elementlar xotiraga istalgan joyga joylashtirilayotgan vaqtda oldin kelib tushgan elementlar joyidan qo‘zg‘atilmay turiladi. Bunda faqat ularning ko‘rsatkichlari to‘g‘rilab qo‘yiladi.

Kamchiligi esa shundan iboratki, oldindan mavjud bo‘lgan tuzilmani massivlarda mavjud bo‘lgan saralash algoritmlari bilan saralash qiyinchilik tug‘diradi, chunki ular elementlarning indekslari bilan bog‘liq tushunchalarga olib keladi.

Elementlarning indeksi degan tushuncha yo‘qligi sababli elementlarga to‘g‘ridan-to‘g‘ri murojaatning iloji yo‘q bo‘ladi, eng so‘nggi holatda oxirgi elementga N ta murojaat orqali yqinlashib boriladi.

Qidiruv amali ham xuddi shunday namoyon bo‘ladi. Yangidan boshlangan vaqtda eng og‘ir holatda oxirgi elementni N ta solishtirishda topish mumkin bo‘ladi.

Bog‘langan ro‘yxatlar bugungi kunda eng ko‘p tarqalgan dinamik tuzilmalardan biri bo‘lib hisoblanadi. Ma’lumotlarni mantiqiy tasvirlash nuqtai nazaridan qaraganda ro‘yxatlar ikkiga ajratilgan bo‘ladi: biri chiziqli, ikkinchisi chiziqsizdir.

Chiziqli ro‘yxatlarda elementlar orasidagi bog‘liqlik qat’iy tartiblangan bo‘ladi, ya’ni element ko‘rsatkichi o‘zidan oldingi yoki o‘zidan keyingi element manzilini saqlab oladi. Chiziqli bog‘langan ro‘yxatlarga bir yoki ikki bog‘lamli

ro‘yxatlar kiradi. Chiziqsiz bog‘langan ro‘yxatlar esa ko‘p bog‘lamli ro‘yxatlardan tashkil topgan bo‘ladi.

Umuman olganda, ro‘yxat elementlari bir yoki bir nechta ko‘rsatkichli maydonlarga ega bo‘ladi, ular har bir ko‘rsatkichi orqali istalgan elementga murojaat qilsa, bunday ro‘yxatlar chiziqsiz bog‘langan ro‘yxatlar deyiladi.

Chiziqli bog‘langan ro‘yxatlarni mantiqiy tasvirlash usullari

Chiziqli bir bog‘lamli ro‘yxatlar va ular ustida bajariladigan amallar algoritmlari. Chiziqli bir bog‘lamli tuzilma elementlari o‘zidan keyingi element bilan bog‘langan bo‘lsa, bunday ro‘yxatga **bir bog‘lamli ro‘yxat** deb nomlanadi.

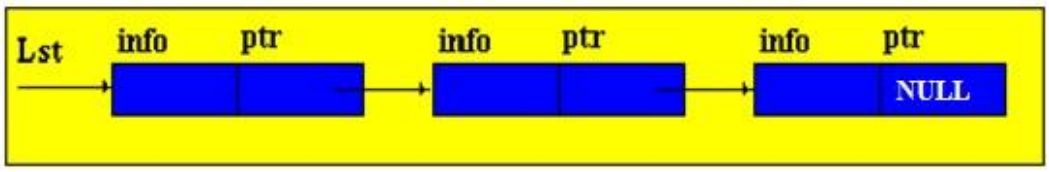
Agar har bir element o‘zidan oldingi va o‘zidan keyingi element bilan bog‘langan bo‘lsa, u holda bunday ro‘yxatlarga **ikki bog‘lamli ro‘yxatlar** deb yuritiladi. Agar oxirgi element birinchi element ko‘rsatkichi bilan bog‘langan bo‘lsa, bunday ro‘yxatga **halqasimon ro‘yxat** deb aytiladi.

Ro‘yxatning har bir elementi shu elementni identifikatsiyalash uchun **kalit**ga ega bo‘ladi. Kalit, odatda butun son yoki satr ko‘rinishida ma’lumotlar maydonining bir qismi sifatida namoyon bo‘ladi.

Ro‘yxatlar ustida quyidagi amallarni bajarish mumkin.

- ro‘yxatni shakllantiriladi (birinchi elementini yaratish);
- ro‘yxat oxiriga yangi elementini qo‘shish;
- berilgan kalitga mos elementlarni o‘qish;
- ro‘yxatning ko‘rsatilgan joyiga element qo‘shish (oldin yoki keyin)
- berilgan kalitga mos elementni o‘chirish;
- kalit bo‘yicha ro‘yxat elementlarini tartibga keltirish.

Ro‘yxatlar bilan ishlashda dasturda boshlang‘ich elementni ko‘rsatuvchi ko‘rsatkichlar talab qilinadi. Chiziqli bir bog‘lamli ro‘yxatlar ustida turlicha amallar bajarish algoritmlari va dasturlarini ko‘rib chiqaylik.



2-rasm. Chiziqli bir bog‘lamli ro‘yxatlar ustida amallar

Mazkur ko‘rinishdagi ro‘yxat elementi ko‘rsatkichining o‘ziga xosligi shundan iboratki, u faqatgina ro‘yxatning navbatdagi (o‘zidan keyin keluvchi) elementi manzilini ko‘rsatadi. Ro‘yxatlarning bir tomonlama yo‘naltirilganida eng so‘nggi element ko‘rsatkichi bo‘sh bo‘ladi, ya’ni u NULL (0) bo‘ladi.

Lst – bu ro‘yxatni bosh ko‘rsatkichidir. U ro‘yxatni yagona bir butun sifatida namoyon qiladi. Ba’zan ro‘yxat bo‘sh bo‘lishi ham mumkin, ya’ni ro‘yxatda bitta ham element bo‘lmasligi ham mumkin.

Bu holda lst NULL (0) bo‘ladi. Agar chiziqli bir bog‘lamli ro‘yxat hosil qilish dasturi ko‘rib chiqiladigan bo‘lsa, u holda foydalanuvchi tomonidan yaratiladigan nostandart toifa yaratib olish zarur bo‘ladi. Uning bir qancha usullari mavjuddir. U klasslar yoki strukturalar bilan amalga oshiriladi. Misol uchun:

```
class Node {
public://klass ma’lumotlariga tashqaridan bo‘ladigan murojaatga ruxsat
berish
   int info; // informatsion maydon
 Node* next;// ko‘rsatkichli maydon
};
```

Bu yerda Node nomli toifa yaratildi va ro‘yxat butun sonlardan iborat edi. Endi ro‘yxat elementlarini shu toifa orqali e’lon qilsa bo‘laveradi, ya’ni

```
  Node *lst NULL;
// ro‘yxat bosh ko‘rsatkichi
  Node *last NULL; // ro‘yxatga oxirgi kelib tushgan elementning
ko‘rsatkichi
```

Endi shu belgilashlar orqali ro‘yxat hosil qilib olinadi.

```
Node * p new Node;
int numb -1;
        cout<<"son kiriting: ";
 cin>>numb;
p->info numb;
p->next NULL;
   if (lst NULL) {
lst  p;
        last p;
}
else{  last->next  p;
        last p; }
```

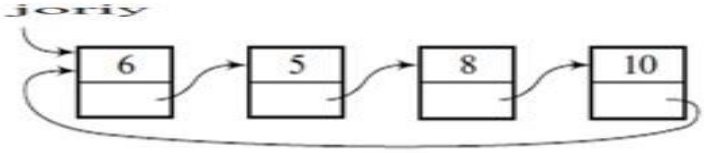
Bu dasturda yangi elementlar ro‘yxatning oxiridan kelib qo‘shiladi.

Bir bog‘lamli ro‘yxatlar ustida quyidagi amallar bajarish algoritmlarini ko‘raylik.

Halqasimon bog‘langan ro‘yxatlar va uni C++ dasturda ko‘rish

Halqasimon bog‘langan ro‘yxatlarda tugunlar halqa ko‘rinishida o‘zaro mustahkam bog‘langan holda bo‘ladi. Ya’ni, oxirgi element ko‘rsatkichi NULL(0) emas, birinchi elementga yo‘naltirilgan bo‘ladi.

Aslida bunday tuzilmalar qachon ishlatiladi, agar bir qancha jarayonlar bir vaqtning o‘zida aynan bir resurslar orqali ishlatilsa, resurslarni teng taqsimlagan holda ishlatilishini ta’minlash zarur bo‘ladi.



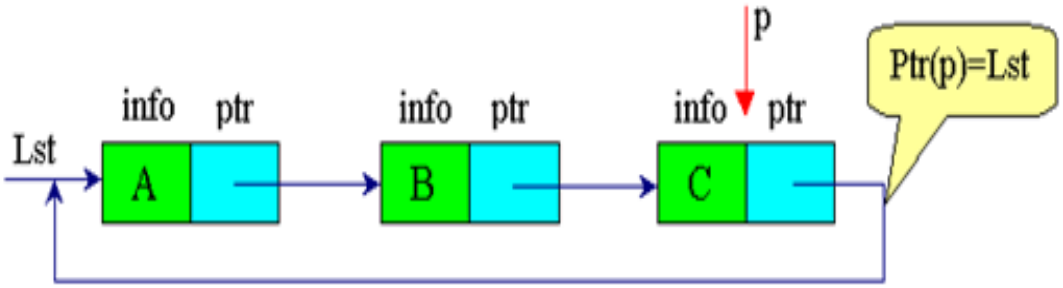
3-rasm. Chiziqli bir bog‘lamli ro‘yxatlar ustida amallar

Tasavvur qilaylik, xuddi yuqoridagi keltirilgan rasmdagidek, o‘sha jarayonlar 6,5,8,10 sonlardan iborat bo‘lsin.

Tuzilmaga current nomli ko‘rsatkich orqali murojaat qilinadi. Bitta elementga murojaat qilib, shu jarayonni bajarilishi ta’minlanadi, ya’ni aktivlashtirish uchun uning qiymati qaytariladi va keyingi elementga o‘tiladi.

Halqasimon ro‘yxatlar ikki xil ko‘rinishda bo‘ladi:

- Halqasimon bir bog‘lamli;
- Halqasimon ikki bog‘lamli.



4-rasm. Chiziqli bir bog‘lamli ro‘yxatlar ustida amallar

Halqasimon ro‘yxatlar ustida bajariladigan amallar

- element qo‘shish;
- element o‘chirish;
- ro‘yxatni ko‘ruvdan o‘tkazish;
- ro‘yxatni o‘chirish;
- bo‘shliqqa tekshirish.

Halqasimon bir bog‘lamli ro‘yxatlar oddiy bir bog‘lamli ro‘yxatda eng so‘nggi element ko‘rsatkichiga ro‘yxat boshi elementi va ko‘rsatkich qiymatini o‘zlashtirishi orqali hosil qilinadi.

Halqasimon bir bog‘lamli ro‘yxatni C dasturlash tilida e’lon qilish usuli bu: struct Node{

int data; //informatsion maydondir

Node *Next;

};

Node *HeadNULL;//ro‘yxatning bosh ko‘rsatkichi

Node *TailNULL;//ro‘yxatning oxirgi ko‘rsatkichi.

Mavzuga oid test savollari

1.**Kompyuter xotirasida binar daraxtini qanday ko‘rinishda tasvirlash qulay hisoblanadi?**

a) bog‘langan chiziqsiz ro‘yxatlar
b) massivlar ko‘rinishida
c) jadvallar ko‘rinishida
d) bog‘langan chiziqli ro‘yxatlar ko‘rinishida

2. Chiziqsiz iyerarxik bog‘langan ma’lumotlar tuzilmasi – bu …?

a) Daraxt
b) Graf
c) Lug‘at
d) Ro‘yxat

3. **Bir bog‘lamli ro‘yxatda nechta ko‘rsatkichdan foydalaniladi?**

a) 1
b) 2
c) 3

4. struct List { int Data; List *Next; }; **Bir bog‘lamli ro‘yxatlarda Next ko‘rsatkichi nima uchun ishlatiladi?**

a) Keyingi elementni ko‘rsatish uchun
b) Oldingi elementni ko‘rsatish uchun
c) Ro‘yxatning boshini ko‘rsatish uchun
d) Ro‘yxatning oxirini ko‘rsatish uchun

Nazorat savollar

1. Chiziqli bir bog‘lamli ro‘yxatlar ustida amal bajarish algoritmlari qanday ko‘rinishda bo‘ladi?
2. Algoritmlarni dastur kodlarini keltirib o‘ting.
3. Bog‘langan ro‘yxatlarda element kiritish qanday amalga oshiriladi?
4. Bog‘langan ro‘yxatlarda element o‘chirish qanday amalga oshiriladi?

Xulosa

Ma’lumki, bog‘langan ro‘yxatlar elementlarning o‘zi ikkiga bo‘linadi. Birinchisi informatsion maydon, ikkinchisi ko‘rsatkichli maydon.

Informatsion maydonda foydalanuvchining foydali ma’lumoti yoziladi, ko‘rsatkichli maydonga keyingi elementning xotiradagi manzili yozib boriladi. Bu har ikkala maydon elementlardan tashkil topadigan tuzilmaga chiziqli bir bog‘lamli ro‘yxatlar tuzilmasi deyiladi. Bularning barchasi bitta bo‘lib, ma’lumotlar ustida tur xil amallarni bajarishga yordam beradi.

Ro‘yxatlar ustida quyidagi amallarni bajarish mumkin:

- ro‘yxatni shakllantirish;
- ro‘yxat oxiriga yangi elementni qo‘shish;
- berilgan kalitga mos elementlarni o‘qish;
- ro‘yxatning ko‘rsatilgan joyiga element qo‘shish;
- berilgan kalitga mos elementni o‘chirish;
- kalit bo‘yicha ro‘yxat elementlarini tartibga keltirish.

1.3. Chiziqli bog‘langan ro‘yxatlar. Ikki bog‘lamli ro‘yxatlar

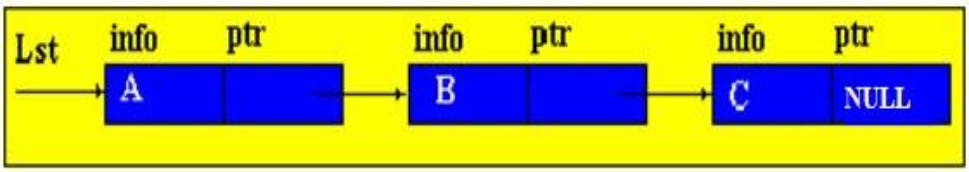
Reja:

1. Bir bog‘lamli ro‘yxatlar va ularning algoritmlari.
2. Elementni ro‘yxatga qo‘shish amali.
3. Halqasimon bir bog‘lamli ro‘yxatlarga element qo‘shish algoritmi.

Tayanch iboralar: informatsion maydon, realizatsiya**,** ro‘yxatlar, mantiqiy tasvirlash, klassifikatsiya, dinamik, lst, ko‘rsatkichlar.

Bir bog‘lamli ro‘yxatlar va ularning algoritmlari

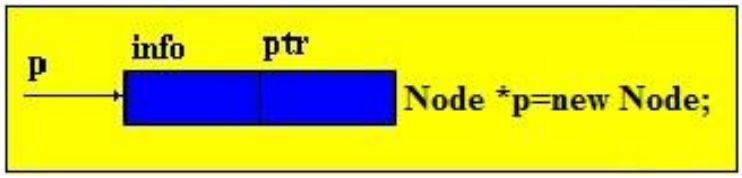
Bir bog‘lamli ro‘yxatning boshiga element qo‘yish quyidagicha bajariladi:



1-rasm. Bir bog‘lamli chiziqli ro‘yxat tuzilishi

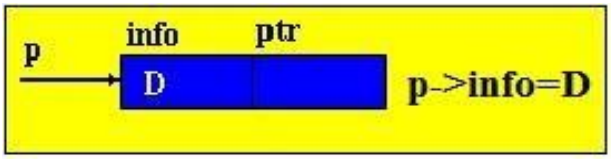
Yuqoridagi ro‘yxatning boshiga, ya’ni informatsion maydon boshiga D o‘zgaruvchi bo‘lgan elementni qo‘yish lozim. Buning uchun ushbu ishni amalga oshirishda quyidagi amallarni bajarish kerak bo‘ladi:

a) p ko‘rsatkichga murojaat qiladigan, bo‘sh elementni yaratish.



2-rasm. Yangi element hosil qilish

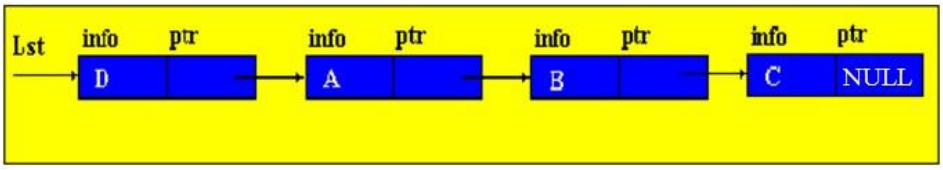
b) Yaratilgan elementning informatsion maydoniga D o‘zgaruvchi qiymatini o‘zlashtirish mumkin.



3-rasm. Yangi element info maydoniga qiymat kiritish

c) Keyingi amal yangi elementni ro‘yxat bilan bog‘lash: p-> ptrlst; (shu holatda yangi element valst– ro‘yxat boshini ko‘rsatyapti)

d) bu esa lst ko‘rsatkichni ro‘yxat boshiga ko‘chirish. lstp; Va nihoyat:



4-rasm. Ro‘yxat boshiga element qo‘shish

Endi shu algoritmni C dasturlash tilida uning realizatsiyasi ko‘rib chiqiladi.

```
Node * p new Node;
int numb -1;
 cout<<"son kiriting: ";
cin>>numb;
p->info numb;
 if (lst NULL){
```

p->next NULL;

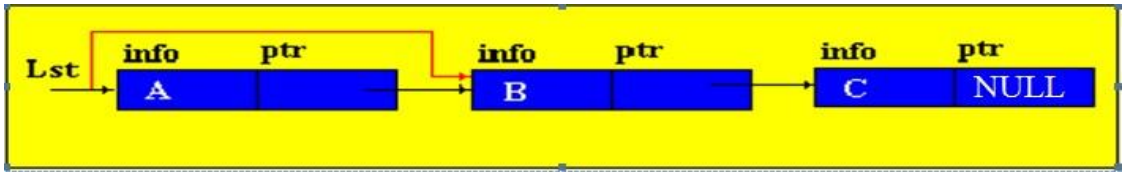
lst p; }

else { p->next lst;

lst p; }

Bir bog‘lamli ro‘yxat boshidan elementlarni o‘chirish

Ro‘yxatda birinchi element info informatsion maydonidagi ma’lumotni esda saqlab qolib, uni ro‘yxatdan o‘chiriladi.



5-rasm. Ro‘yxat boshidagi elementni o‘chirish

Yuqorida aytilganlarni amalga oshirish uchun quyidagi ishlarni bajarish kerak:

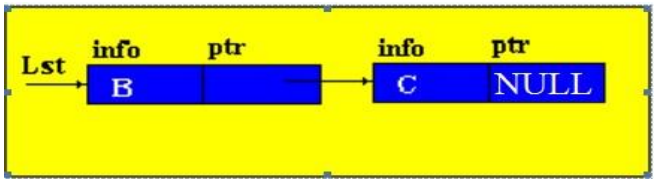
a) o‘chirilayotgan elementni ko‘rsatuvchi p ko‘rsatkichga kiritish: plst;

b) p ko‘rsatkich ko‘rsatayotgan element info maydonini qandaydir x o‘zgaruvchida saqlash lozim: xp->info;

c) lst ko‘rsatkichni yangi ro‘yxat boshiga ko‘chirish shart: lstp->ptr;

d) p ko‘rsatkich ko‘rsatayotgan elementni o‘chirish lozim: delete(p);

Natijada 6-rasmdagi ko‘rinishga ega bo‘ladi.



6-rasm. Ro‘yxatning natijaviy ko‘rinishi

Endi shu algoritmni C dasturlash tilidagi realizatsiyasini ko‘rib chiqaylik.

Node* p new Node;

if (lst NULL){

cout<<"ro‘yxatbo‘sh";

system("pause");

system("CLS");

}

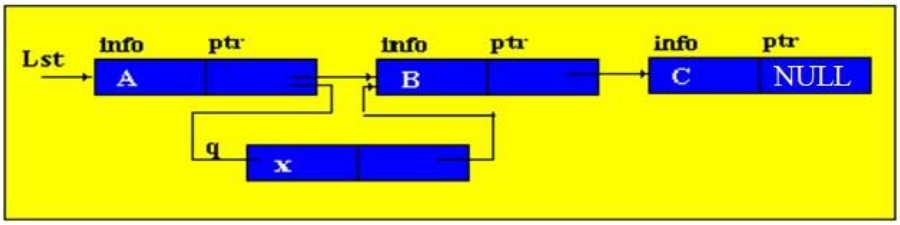
else { p lst;

lst p->next ;

delete(p); }

Elementni ro‘yxatga qo‘shish amali

Berilgan ro‘yxatdagi p ko‘rsatkichni ko‘rsatilayotgan elementdan keyin informatsion maydonga x bo‘lgan element bilan to‘ldiriladi.



7-rasm. Ro‘yxatga yangi element qo‘shish

Aytilganlarni amalga oshirish uchun quyidagi amallarni bajarish shart:

a) q ko‘rsatkich ko‘rsatuvchi bo‘sh elementni yaratish jarayoni: Node *qnew Node;

b) Yaratilgan element informatsion maydoniga x ni kiritish usuli: q->infox;

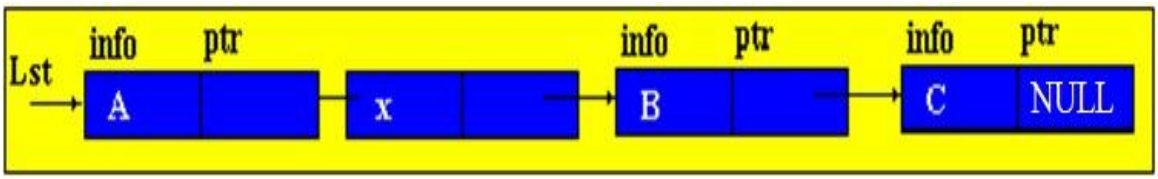
c) q elementni p elementdan keyingi element bilan bog‘lanadi.

q->ptrp->ptr – yaratilgan element ko‘rsatkichiga p element ko‘rsatkichini o‘zlashtiradi.

d) p element bilan q elementni bog‘lash jarayoni.

p->ptrq – bu amal p elementdan keyingi element q ko‘rsatkichga murojaat qilgan element bo‘lishini isbotlaydi.

Natijada quyidagi rasmdagidek holatga ega bo‘ladi.



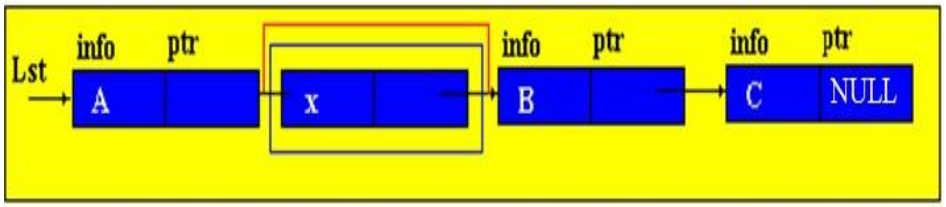
8-rasm. Natijaviy ro‘yxat ko‘rinishi

Endi shu algoritmni C dasturlash tilidagi realizatsiyasini ko‘rib chiqaylik.

```
Node* p lst;
Node* q  new Node;
 int numb -1;
 cout<<"son kiriting: ";
 cin>>numb;
 q->number numb;
int k;
cout<<"nechta elementdan keyin kiritasiz k"; cin>>k;
 for(int i0;i<k-1; pp->next;
  q->next p->next;
p->next q;
```

Bir bog‘lamli ro‘yxatdan elementni o‘chirish amali

Ro‘yxatda p ko‘rsatkich ko‘rsatayotgan elementdan keyingi element o‘chiriladi.



9-rasm. Ro‘yxat o‘rtasidan element o‘chirish amali

Buni ro‘yobga chiqarish uchun quyidagi ishlarni amalga oshirish mumkin:

a) O‘chirilayotgan elementni ko‘rsatuvchi q ko‘rsatkichni kiritish.

```
qp->ptr;
```

b) p elementni q elementdan keyingi element bilan uni bog‘lash.

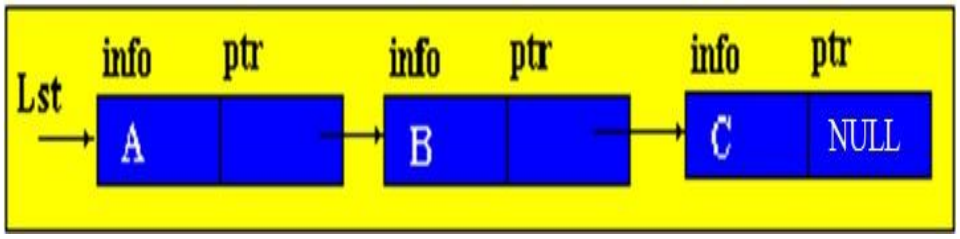
```
p->ptrq->ptr;
```

c) O‘chirilayotgan element info maydonidagi informatsiyasini yodda saqlash

```
(agar zarur bo‘lsa) kq->info;
```

d) q ko‘rsatkich ko‘rsatayotgan elementni o‘chirish amali. delete(q)

Natijada ro‘yxat quyidagi ko‘rinishga ega bo‘ladi:



10-rasm. Natijaviy ro‘yxat ko‘rinishi

Shu algoritmni c dasturlash tilidagi dasturi:

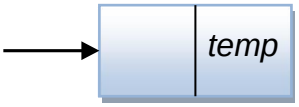
```
Node* p lst;
Node* q new Node;
int k;
cout<<"k";cin>>k;
 for(int i0; i<k-1; i) pp->next;
q  p->next;
 p->next q->next;
  delete(q);
```

Halqasimon bir bog‘lamli ro‘yxatlarga element qo‘shish algoritmi

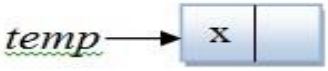
Bo‘sh bo‘lgan ro‘yxatga elementlar rasmdagi kabi kiritilishi kerak bo‘lsin.

1. Yangi elementni e’lon qilinadi:

```
Node  *tempnew Node;
```



2. Yangi elementning informatsion maydoniga x qiymat kiritiladi. temp->datax;

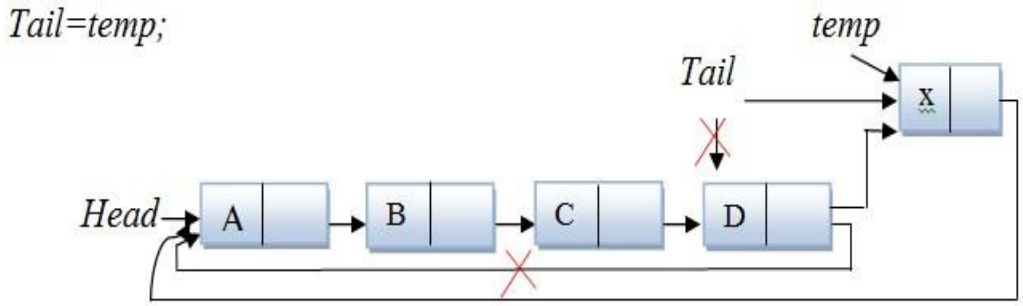


3. Yangi element ko‘rsatkich maydoniga ro‘yxat boshi ko‘rsatkichi o‘zlashtiriladi. temp->next Head;

4. Oxirgi element Tail ni ko‘rsatkichini yangi elementga ulanadi.

```
Tail->nexttemp;
```

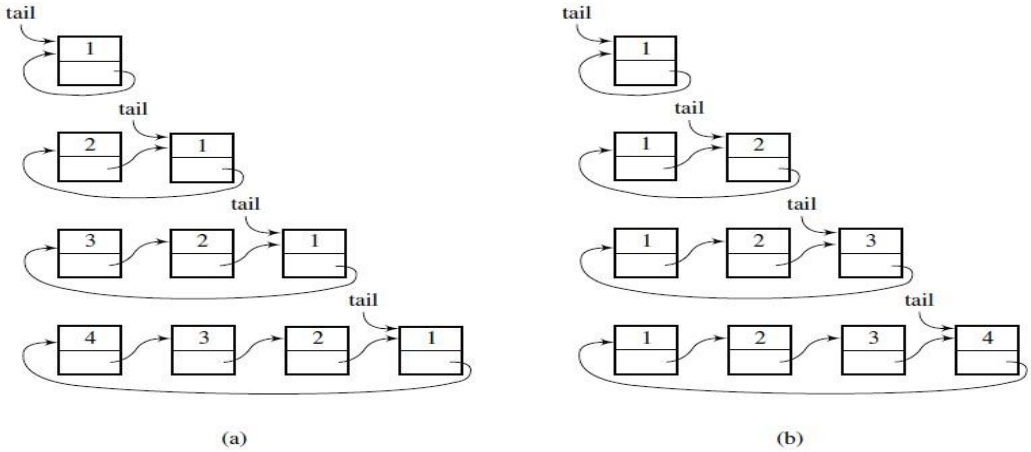
5. Oxirgi elementni ko‘rsatuvchi Tail ni yangi elementga to‘g‘rilab qo‘yiladi. Tailtemp;



11-rasm. Halqasimon bir bog‘lamli ro‘yxatlarga element qo‘shish algoritmi

Ushbu algoritmning dastur kodi esa quyidagi ko‘rinishda bo‘ladi:

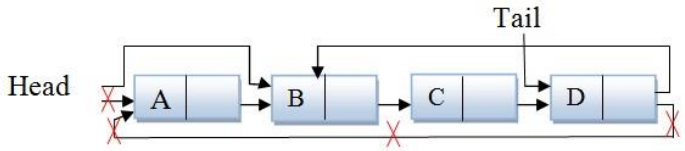
```
void Add(int x)
{
Node  *tempnewNode;
 temp->NextHead;
temp->datax;
  if (Head!NULL) {   Tail->Nexttemp;
  Tailtemp;
 }
else HeadTailtemp;
}
```



12-rasm. Halqasimon bir bog‘lamli ro‘yxatlarga element qo‘shish algoritmi

Halqasimon bir bog‘lamli ro‘yxat boshidan element o‘chirish algoritmi

Quyidagi shaklda ro‘yxat boshidagi elementni o‘chirish amali namoyon qilingan.



13-rasm. Halqasimon bir bog‘lamli ro‘yxat boshidan element o‘chirish algoritmi

- birinchi elementni o‘chirish uchun uni ko‘rsatuvchi Head ni 2-elementga to‘g‘rilanadi.
- oxirgi element, ya’ni Tail ko‘rsatayotgan elementni ko‘rsatkich maydonini ikkinchi elementga to‘g‘rilab qo‘yiladi.
- natijada birinchi elementni xotiradan o‘chirib tashlanadi.

Ushbu algoritmning dastur kodi keltirib o‘tiladi.

```
void Del(){
  Node *delItemHead;
  HeadHead->Next;
  Tail->NextHead;
   delete delItem;
}
```

Halqasimon bir bog‘lamli ro‘yxatni ekranga chiqarish algoritmini ko‘raylik.

```
void Show() {
    Node *pHead;
     wxile (p!Tail)
   {
    cout<<p->data<<" ";
  pp->Next;
     }
      cout<<p->data<<endl;
  }
```

Mavzuga oid test savollari

1. Ikki bog‘lamli ro‘yxatda nechta ko‘rsatkichdan foydalaniladi?

a) 2

b) 1

c) 3

d) 4

2.

```
struct List
{ int Data;
List *Next, *Prev;
};
```

Ikki bog‘lamli ro‘yxatlarda Next va Prev ko‘rsatkichlari nima uchun ishlatiladi?

a) Keyingi va oldingi elementlarini ko‘rsatish uchun

b) Faqat oldingi va undan keyingi elementlarini ko‘rsatish uchun

c) Ro‘yxatning boshini ko‘rsatish uchun

d) Ro‘yxatning oxirini namoyon qilish uchun

3. Halqasimon ikki yo‘nalishli ro‘yxatda qaysi yo‘nalishlar bo‘yicha harakatlanish mumkin?

a) ikkala

b) chapga

c) o‘nga

d) ro‘yxat oxiriga

4. Murrakab ob’ektlarning xususiyati va munosabatlarini aks ettiruvchi chiziqsiz ko‘p bog‘lamli dinamik tuzilmasini ko‘rsating.

a) Graf

b) Lug‘at

c) Daraxt

d) Ro‘yxatlardan iborat

5.Ma’lumotlar tuzilmalari tartibiga ko‘ra quyidagilarga klassifikasiyalanadi?

a) Chiziqli va chiziqsiz

b) Statik, yarimstatik va dinamik

c) Bog‘lamli va bog‘lamsiz

d) Oddiy va murakkab

Nazorat savollari

1. Bog‘langan ro‘yxatlarda element qidirish qanday amalga oshiriladi?
2. Bog‘langan ro‘yxatlarda elementlarini ekranga chop etish qanday amalga oshiriladi?
3. Massivga nisbatan ro‘yxatning kamchiligi nimadan iborat?
4. Chiziqli ro‘yxatlarning halqasimon ro‘yxatlardan farqi nimada?
5. Halqasimon bir bog‘lamli ro‘yxatlar deganda nimani tushunasiz?

Xulosa

Chiziqli bog‘langan ro‘yxatlar bir yoki ikki bog‘lamli ro‘yxatlar ko‘rinishda bo‘ladi. Bunda birinchi yacheykada turgan ma’lumotning manzili ikkinchi yacheykada joylashgan bo‘ladi.

Ma‘lumotlar shu tartibda joylashib boradi va oxirgi ma’lumot manzili nolga aylanib qoladi. Bu yerda ma’lumotlar ustida amallar shu tartibda olib boriladi. Halqasimon bir bog‘lamli ro‘yxatlar ham shu tartibda bajariladi.

3-BOB. STEK, NAVBAT VA DEK

1-§. Stek va navbat

1.1. Stek, navbat va dek. Stek, navbat va deklarni massiv yordamida tasvirlash

Reja:

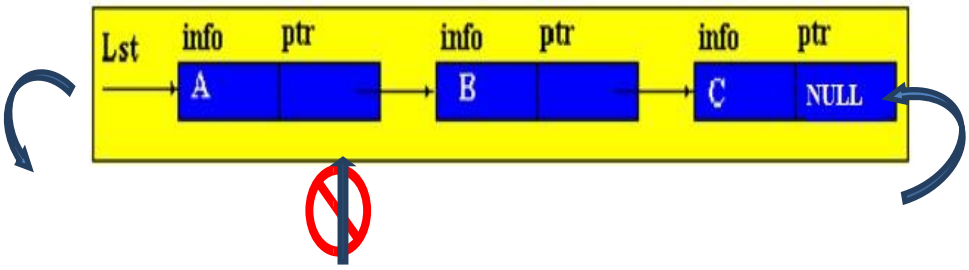
1. Stek, navbat, deklar haqida tushuncha va ularni misollar yordamida tasvirlash.

2. Stek, navbat va deklarni bog‘langan ro‘yxatga oid misol yechish.

Tayanch iboralar: Stek, navbat, deklar, ustuvor navbatlar, massiv, ro‘yxat, lug‘at, bog‘langan ro‘yxat.

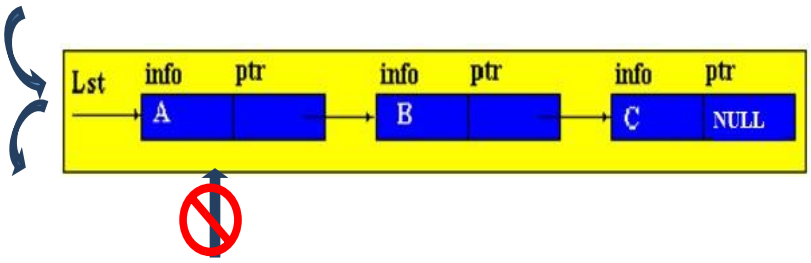
Stek, navbat, deklar haqida tushuncha va ularni misollar yordamida tasvirlash

Oldingi mavzularda stek, navbat, dek tuzilmalari haqida to‘xtalgan edi. Bu tuzilmalar yarimstatik bo‘lganligi sababli ularni dasturda statik yoki dinamik ko‘rinishda ifodalash mumkin. Statik ko‘rinishda ifodalaganda ularni dasturda masalan, massiv shaklda ifodalab, yarimstatiklik shartini buzmagan holda ishlatish mumkin bo‘ladi. Ya’ni, elementlarga murojaat navbatga boshidan, elementlarni kiritish esa oxiridan amalga oshiriladi va o‘rtadan murojaat mumkin bo‘lmaydi. Ana shu shartni qanoatlantirgan holda istalgan ko‘rinishda ifodalash mumkin bo‘ladi. Demak, yarimstatik tuzilmalarni dinamik ko‘rinishda ham ifodalash mumkindir, unda faqat elementlarni kiritish va chiqarishda ularning xususiyatlarini saqlab qolishda ahamiyat berish lozim. Navbatni chiziqli bir bog‘lamli ro‘yxat ko‘rinishida tasvirlashga qaratiladi.



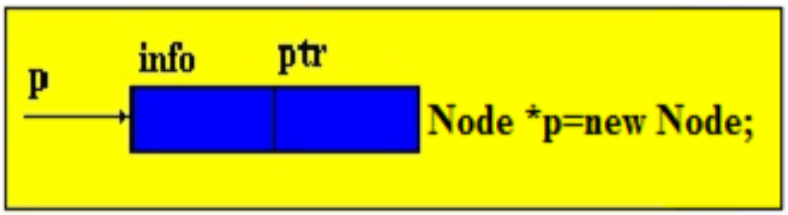
1-rasm. Navbatni chiziqli bir bog‘lamli ro‘yxat ko‘rinishi

Stekni chiziqli bir bog‘lamli ro‘yxat ko‘rinishida tasvirlanadi.



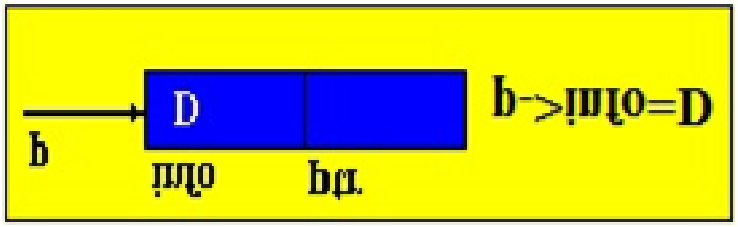
2-rasm. Navbatni chiziqli bir bog‘lamli ro‘yxat ko‘rinishi

Ushbu ko‘rinishda stek tuzilmasini tashkil qilishni va uning ustida amallar bajarish algoritmlari ko‘rib chiqiladi:



3-rasm. Stek tuzilmasini tashkil qilishni va uning ustida amallar bajarish algoritmi

Bulardan biri yangi element yaratish bo‘lsa, ikkinchisi esa uning info maydoniga ma’lumot kiritishdir.

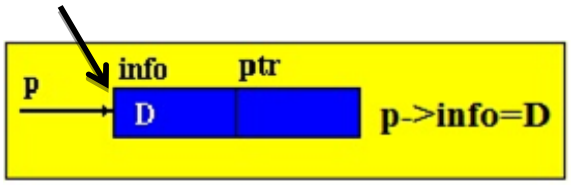


4-rasm. Stek tuzilmasini tashkil qilishni va uning ustida amallar bajarish algoritmi

1. Agar ro‘yxat bo‘sh bo‘lsa, ro‘yxat boshi ko‘rsatkichini ushbu elementga to‘g‘rilanadi va yangi element ptr maydoniga NULL yoziladi, ya’ni **if(LstNULL){ p->ptrNULL; Lstp}** **Lst**

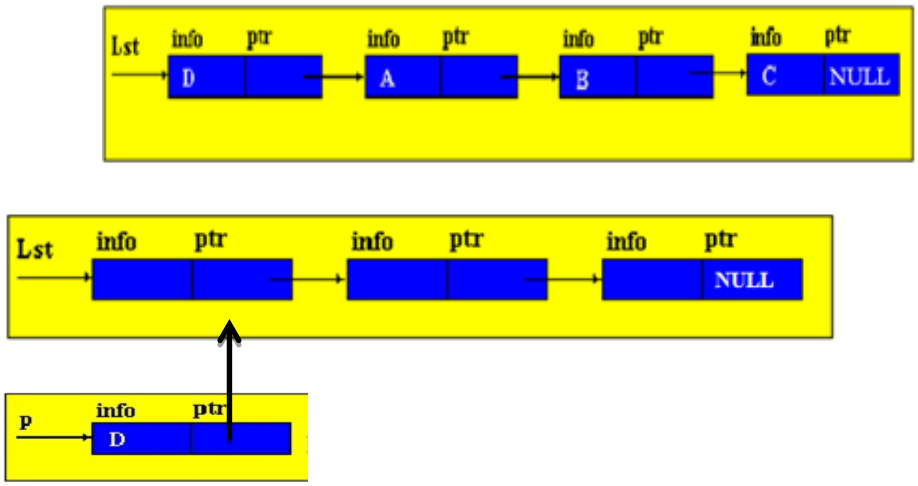
2. Aks holda, ya’ni ro‘yxat bo‘sh bo‘lmasa, yangi yaratilayotgan element ptr maydoniga ro‘yxatning 1-element manzili yoziladi.

p->ptrLst;



5-rasm. Stek tuzilmasini tashkil qilish va uning ustida amallar bajarish algoritmi

3. Ro‘yxat boshi ko‘rsatkichlarini yangi elementga to‘grilanadi.



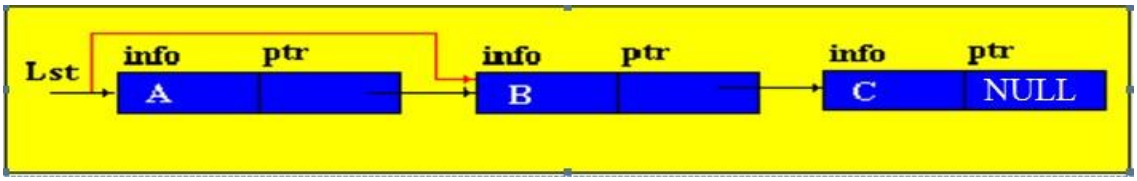
6-rasm. Ro‘yxat boshi ko‘rsatkichlarini yangi elementga to‘g‘rilash

Stekda yangi element kiritish dastur kodi keltirib o‘tiladi.

```
class Node{
public:  int info;
  Node* ptr;  };
int main() {       Node* Lst  NULL;
        Node* p new Node;
int numb;
   cout<<"son kiriting: ";
 cin>>numb;
 p->info  numb;
 p->ptr  Lst;
```

Lst p;

Stekdan element chiqarish amali ham xuddi shu tomondan amalga oshiriladi.



7-rasm. Stekdan element chiqarish amali

Bunda dastur kodini keltiriladi.

Node* p new Node;

if (Lst NULL)

cout<<"ro‘yxat bo‘sh";

else { p lst;

lst p->next ;

delete(p);

}

Stek, navbat va deklarni bog‘langan ro‘yxatga oid misol yechish

C dasturlash tilida navbatni statik, ya’ni bir o‘lchamli massiv ko‘rinishda amalga oshirish jarayonini misol sifatida ko‘rib chiqaylik:

Aytaylik, navbat uchun 10 ta joy ajratilgan bo‘lsin, navbatni butun sonlardan iborat massiv shaklida ifodalanadi. Bunda navbat dastlab bo‘shlig‘i sababli, navbat oxiri ko‘rsatkichi R0 bo‘ladi. Navbatga yangi element qo‘shish va navbatdan elementni chiqarib olish algoritmi, navbat bo‘shlig‘ini va to‘laligini tekshirish algoritmlari quyidagi dasturda keltirib o‘tilgan. Misol uchun, butun sonlardan iborat navbatning juft elementlarini o‘chirish dasturini ko‘rib chiqaylik.

Masalaning algoritmi:

1. Agar navbat to‘lmagan bo‘lsa, unga element kiritiladi, kiritib bo‘lgach keyingi 2-qadamga o‘tish, aks holda navbat to‘lganligini xabar berib, keyingi 2-qadamga o‘tish lozim bo‘lsin.

2. Agar navbat bo‘sh bo‘lmasa, 3-qadamga o‘tiladi, aks holda 4-qadamga o‘tish bajarilsin.

3. Navbatning chiqishiga kelib turgan elementni olib, juftlikka tekshiriladi. Agar element toq bo‘lsa, uni navbatga kiritiladi. 2-qadamga o‘tish.

4. Navbat bo‘sh bo‘lsa, bu haqida xabar berib keyingi 5-qadamga o‘tiladi.

5. Oxirgi amal navbat tarkibini ekranga chiqariladi.

C dasturlash tilida dastur kodini ko‘rib chiqaylik:

```
#include <iostream>
 using namespace std;
int a[10],R0,n;//n  esa navbatga kiritiladigan elementlar soni.
int kiritish(int s){
a[R]s; R;     }
int chiqarish(){
  int ta[0];
  for(int i0;i<R-1;i)
     a[i]a[i1];     R--;
    return t;     }
 bool isEmpty()
{ if(R0) return true;
 else return false;    }
bool isFull(){
 if(R>10)return true; else return false; }
 int print() {   int i;
wxile(i<R){
 int qisqarish();
i; cout<<k<<"  ";
kiritish(k); }
 }
 int main() {  int n,s;
```

```
 cout<<"n"; cin>>n;
   for(int i0;i<n;i) {
  if(!isFull()){cin>>s;
    kiritish(s); }
else{cout<<"navbat to‘ldi"; ni;break; }
      }
    cout<<"\n navbat elementlari: ";
    print();
 for(int i0;i<n;i) {
schiqarish();
  if(s%2!0)kiritish(s);       }
    cout<<"\n natijaviy navbat elementlari: ";
  print();
    system("PAUSE"); }
```

Mavzuga oid test savollari

1.Qanday konteyner yordamida grafda tubiga qarab ko‘rishda qo‘llaniladi?

a) stek

b) navbat

c) ro‘yxat

d) dek

2.Qanday konteyner yordamida grafda eniga qarab ko‘rishda qo‘llaniladi?

a) navbat

b) stek

c) ro‘yxat

d) dek

3. Stek tuzilmasida qanday xizmat ko‘rsatish turi qo‘llaniladi?

a) LIFO

b) FIFO

c) FILO

d) LILO

4. Navbat tuzilmasida qanday xizmat ko‘rsatish turi qo‘llaniladi?

a) FIFO

b) LIFO

c) FILO

d) LILO

5. Stekga yangi element qo‘shish funksiyasi qanday belgilanadi?

a) Push

b) Pop

c) Top

d) Empty

Nazorat savollari

1. Yarimstatik ma’lumotlar tuzilmasi deganda nimani tushunasiz?
2. Stek tuzilmasi haqida gapiring.
3. Navbat tuzilmasi haqida ayting.
4. Dek ma’lumotlar tuzilmasi haqida nimalarni bilasiz?

Xulosa

Oldingi mavzularda stek, navbat, dek tuzilmalari haqida to‘xtalgan edi. Bu tuzilmalar yarim statik bo‘lganligi sababli, ularni dasturda statik yoki dinamik ko‘rinishda ifodalash mumkin.

Statik ko‘rinishda ifodalaganda ularni dasturda, masalan, massiv shaklda ifodalab, yarim statiklik shartini buzmagan holda ishlatish mumkin bo‘ladi. Ya’ni, elementlarga murojaat navbatga boshidan, elementlarni kiritish esa oxiridan amalga oshiriladi va o‘rtadan murojaat esa mumkin bo‘lmaydi.

Ana shu shartni qanoatlantirgan holda istalgan ko‘rinishdagi ma’lumotlarni ifodalash mumkin bo‘ladi.

1.2. Stek, navbat va dek. Stek, navbat va deklarni chiziqli bog‘langan ro‘yxati yordamida tasvirlash

Reja:

1. Stek konteyneri haqida tushuncha va ularga oid misollar yechish.
2. Navbat konteynerlari ustida misollar ko‘rish.

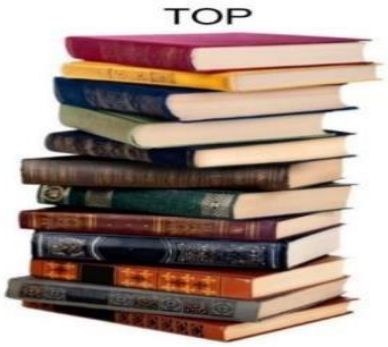
Tayanch iboralar: Stek, navbat, navbat konteynerlari, deklar, ustuvor navbatlar, massiv, ro‘yxat, lug‘at, bog‘langan ro‘yxat.

Stek konteyneri haqida tushuncha va ularga oid misollar yechish

Stek – Stack deb nomlangan inglizcha so‘zdan olingan bo‘lib: uyum, g‘aram, dasta, bog‘lam degan ma’noni bildiradi.

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

Bu juda aniq ta’rif bo‘lib, ammo yangi o‘rganuvchilar uchun uni tushunish kishiga biroz qiyinchilik tug‘diradi. Shuning uchun hayotdagi narsalarning to‘plangan ko‘rinishi haqida to‘xtalib o‘taylik. Eng oddiy tushunchadan boshlaydigan bo‘lsak: bunda aytaylik, kitoblar to‘plami ustma-ust taxlangan bo‘lsin. Bunda eng yuqorida turgan kitob oxirgi marotaba birinchi joylashtirilgan kitob hisoblanadi.



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 ham mumkin, ammo kitoblar bilan misol eng maqbul yechimdir.

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‘lib hisoblanadi. Agar stekka uchta element qo‘shilsa, avval qo‘shilgan oxirgi element o‘chiriladi.

Quyida 2-rasmga e’tibor qaratsangiz siz 6 ta raqamni ko‘rishingiz mumkin: 6, 3, 8, 2, 4, 7. Shunga yaxshiroq e’tibor bersangiz, ular bir xil tartibda chiqariladi.

Masalan, 8 raqamini chiqarish uchun avval 6 va 3 raqamlarini, so‘ngra 1 ni ajratib olish kerak, chunki bu stek, bu raqamlar teskari tartibda qo‘shiladi. Aniqroq qilib aytganda, 7, 4, 2, 8, 3, 6 mana bu holatga keladi.

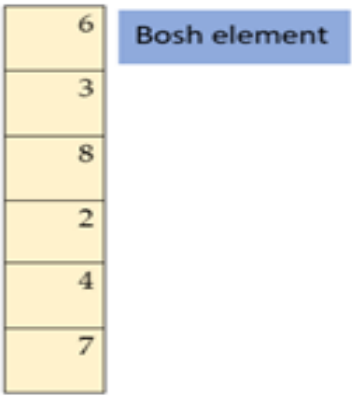
Stekda massivdagi kabi indekslar mavjud emas, demak ma’lum bir elementga murojaat qilib bo‘lmaydi. Buning sababi, stek bog‘langan ro‘yxatlar asosida tuzilgan.

Bu shuni anglatadiki, har bir element (oxirgisidan tashqari qolgan elementlar NULLga ishora qiladi, oddiy so‘zlar bilan aytganda, hech narsaga ishora qilmasa NULL bo‘ladi) keyingi element ko‘rsatgichga ega.

Ammo ko‘rsatgich bo‘lmagan element mavjud - birinchisi (yoki uni bosh element deb ham atashadi).

Shu o‘rinda savol ham paydo bo‘ladi? Nima uchun massivlarni ishlatish mumkin bo‘lganda stekni ishlatiladi, - degan savol juda ko‘p marotaba esga tushadi.

Sababi shundan iboratki, stek to‘plamining asosiy kuchi elementlarni qo‘shish va olib tashlashdan iborat ekanligidadir.



2-rasm. Stek ma’lumotlar strukturasi

Ushbu amallar doimiy vaqt ichida amalga oshiriladi. Ba’zi dasturchilar ko‘pincha massivda stek qilishadi.

C dasturlash tilida stekni realizatsiya qilish jarayoni.

Bu jarayon dastur boshida stek shablonidan foydalanish uchun <stack> kutubxonasini ishga tushurish kerak bo‘ladi. Stek yaratish uchun quyidagi usul orqali ishlash zarur:

stack <ma’lumot_turi> <nom>;

Ya’ni, bunda yangi satrda stack kalit so‘zini yozish kerak. <ma’lumotlar turi> - bu yerda stekda saqlanadigan ma’lumotlar turini yozish kerak bo‘ladi. <nom> - bu stek nomi.

Steklar bilan ishlash metodlari. Metodlar - navbat va stek kabi konteynerlar uchun ishlatiladigan funksiyalardir. Quyida C dasturlash tilida stekda ishlatiladigan metodlarni ko‘rib chiqiladi:

```
#include <iostream>
#include <stack> //stek kutubxonasini ulash
 using namespace std;
int main()    {
stack <int> stek; // Stek yaratish
 int i  0;
 cout << "istalgan oltita son joylang: " << endl;
wxile (i ! 6) {
```

```
 int a; cin >> a;
stek.push(a); // stekka sonni qo‘shish
i; } if (!stek.empty())
cout << "Stek bo‘sh emas"; //Stekni bo‘shlig‘ini tekshirish
cout << "Stekning yuqori elementi: " << stek.top() << endl; // Eng yuqori elementni chop etish
cout << "Eng yuqori elementni yo‘qotish " << endl; // stek.pop(); // yuqori elementni olib tashlash
cout << "endi yangi yuqori element: " << stek.top();
return 0; }
```

Dasturda berilgan push() funksiyasi yordamida stekka element qo‘shib olinadi.

Qavslar ichida qo‘shmoqchi bo‘lgan qiymat bo‘lishi muhim. Dastur kodida stek bo‘shlig‘ini tekshirish uchun empty() metodidan foydalaniladi. Agar bu funksiya natijasi true bo‘lsa, u holda stek bo‘sh bo‘ladi. Agar natija false bo‘lsa, unda stekda element bor bo‘ladi. Eng asosiysi, 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 shart.

Stekning eng yuqori elementini olish uchun top() funksiyasidan foydalaniladi.

peek() funksiyasi. Stack kutubxonasiga yangi peek () funksiyasi qo‘shildi, u yordamida stekning N-elementiga murojaat qilish mumkin bo‘ladi. Shu ko‘rinishda endi stek massivga o‘xshash bo‘ladi. Quyida peek() funksiyasidan foydalanib, uchinchi element chiqariladi. Ushbu funksiya C11 standartidan keyin qo‘shilgan. Bu holatni C dasturlash tilida ko‘rib chiqaylik.

```
#include <iostream>
#include <stack> //stek kutubxonasini birlashtirish
 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;
}
```

Shu vaqtgacha peek() funksiyasidan dasturchilarning kam qismi foydalanib kelgan. Endi esa bu funksiya yaratuvchi dasturchilar uchun ular kutganidek ommalashib ketgani yo‘q.

Navbat konteynerlari ustida misollar ko‘rish

C dasturlash tilida navbatni realizatsiya qilish ham mumkin. Agar C dasturlash tilida navbat shablonidan foydalanilsa, unda avval <queue> kutubxonasini kiritish lozim. Bundan tashqari, navbatni e’lon qilish uchun quyidagi strukturani ishlatish shart.

queue <ma‘lumot turi> <nom>;

Masalan:

queue <int> navbat;

Navbatning metodlariga e’tibor beriladigan bo‘lsa, navbat bilan ishlash uchun funksiyalarni bilish shart: push(), pop(), front(), back(), empty().

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

2. Agar birinchi elementni olib tashlash kerak bo‘lsa, pop() funksiyasi bilan ishlash lozim bo‘ladi. 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 bo‘lib qoladi: empty(),

back() va front().

3. Agar navbatning birinchi elementiga murojaat qilish kerak bo‘lsa, unda front() funksiyasidan foydalanish shartdir.

4. back() funksiyasi navbatdagi oxirgi elementga kirishga yordam qiladi.

5. Navbatning bo‘shlig‘ini bilish uchun empty() funksiyasidan foydalanish eng muhim hisoblanadi.

-Agar navbat bo‘sh bo‘lsa, u true qiymatini qaytara boshlaydi.

-Agar unda biror narsa bo‘lsa, u false qiymatini qaytaradi.

Quyida yuqoridagi metodlarning barchasini qo‘llash orqali C dasturlash tilida dastur tuzib, ular shu dasturda qo‘llab ko‘rildi:

```
#include <iostream>
#include <queue> // Queue kutubxonasini ulash
using namespace std;
 int main()
 {
queue <int> N; // Navbat yaratish
 cout << "Yettita sonni 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;
}
```

Mavzug oid test savollari

1. Stekdan yuqori elementini o‘chirish funksiyasi qanday belgilanadi?

a) Pop

b) Push

c) Top

2. C tilida standart andozalar kutubxonasi yordamida stekni qanday e’lon qilish mumkin?

a) stack <int > S;

b) queue <int > S;

c) deque <int > S;

3. Stek bu ...?

a) chiziqli ma’lumotlar tuzilmasi bo‘lib, ma’lumotlarni kiritish va chiqarish uning bir tomonidan amalga oshiriladi.

b) shunday tuzilmaki, u elementlar qo‘shilishi bilan kengayib boradi va elementlarni faqatgina bir tomondan qabul qiladi.

c) chiziqli ma’lumotlar tuzilmasi bo‘lib, ma’lumotlarni kiritish va chiqarish unda ikki tomonlama amalga oshiriladi.

4. Navbat bu...?

a) shunday tuzilmaki, u elementlar qo‘shilishi bilan kengayib boradi va elementlarni faqatgina bir tomondan qabul qiladi.

b) chiziqli ma’lumotlar tuzilmasi bo‘lib, ma’lumotlarni kiritish va chiqarish uning bir tomonidan amalga oshiriladi.

c) chiziqli ma’lumotlar tuzilmasi bo‘lib, ma’lumotlarni kiritish va chiqarish uning ikki tomonlama amalga oshiriladi.

Nazorat savollari

1. Stek ro‘yxat yordamida qanday tasvirlanadi?
2. Navbat ro‘yxat yordamida qanday tasvirlanadi?
3. Dek ro‘yxat yordamida qanday tasvirlanadi?

Xulosa

Yuqorida stekga aniq ta’rif berib o‘tildi. Bu ta’rif, stek nima ekanligini birdiradi. Lekin, bu stekdan hayotda juda ko‘p hollarda foydalaniladi. Ammo yangi o‘rganuvchilar uchun uni tushunish kishiga biroz qiyinchilik tug‘diradi.

Shuning uchun hayotdagi narsalarning to‘plangan ko‘rinishi haqida tasavvur qilish kerak. Eng oddiy tushunchadan boshlanadigan bo‘lsa, bunda aytaylik, kitoblar to‘plami ustma-ust taxlangan bo‘lsin. Eng yuqorida turgan kitob oxirgi marotaba birinchi joylashtirilgan kitob hisoblanadi. Bu usul bilan hayotda juda ko‘p marotaba foydalaniladi. Aslida, stek har qanday narsaning to‘plami sifatida ifodalanishi mumkin, u daftar, ruchka va shunga o‘xshash narsalar to‘plami bo‘lishi ham mumkin, ammo kitoblar bilan misol eng maqbul yechimdir.

Shunday qilib, 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‘lib hisoblanadi. Agar stekka uchta element qo‘shsangiz, avval qo‘shilgan oxirgi element o‘chiriladi.

Demak, ma‘lumotlar tuzilmalari ustida ham har xil ko’rinishda amallar bajarilganda aynan mana shu holatlarga duch kelinadi. Bundan tashqari, navbat va dek saralash imkoniyatlari mavjud. Bular o‘zaro bir-birlaridan anchagina farq qiladi.

1.3. Stek, navbat va dek. Ustuvor navbatlar. Lug‘atlar va ularni amalga oshirish

Reja:

1. Navbat va deklarlarni misollar va algoritmlar asosida yoritish.
2. Dinamik ma’lumotlar tuzilmasi.
3. Dek va ular ustida misollar yechish.

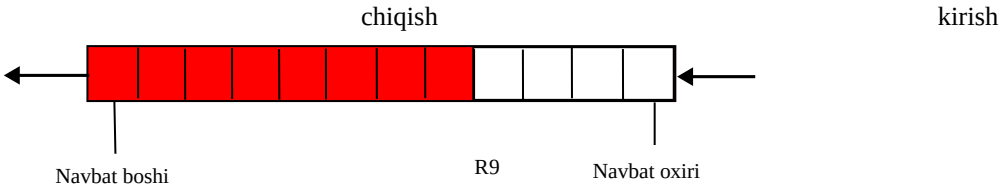
Tayanch iboralar: Stek, navbat, navbat konteynerlari, deklar, dek tuzilmasi, ustuvor navbatlar, massiv, ro‘yxat, lug‘at, bog‘langan ro‘yxat.

Navbat va deklarlarni misollar va algoritmlar asosida yoritish

Navbat bu FIFO (First In - First Out - "birinchi kelgan – birinchi ketadi"), shunday o‘zgaruvchan uzunlikdagi ketma-ketlik bo‘lib, bunday ketma-ketlik ro‘yxati ko‘rinishdagi tuzilmaga elementlar faqat bir tomondan, navbatning oxiridan qo‘shiladi va elementlarni tuzilmadan chiqarish boshqa tomondan, ya’ni navbat boshidan amalga oshiriladi. Navbat ustida bajariladigan asosiy amallar quyidagilar:

- yangi elementlarni qo‘shish;
- elementni chiqarib tashlash;
- uzunligini aniqlash;
- navbatni tozalash.

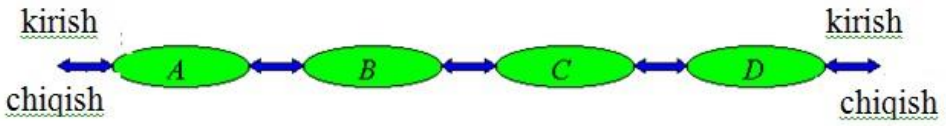
Navbatni statik xotirada vektor ko‘rinishida ifodalashda ikkita parametr, ya’ni navbat boshini (navbatning 1-elementini) va oxirini (navbatning oxirgi elementini) ko‘rsatuvchi ko‘rsatkichlar olinadi(1-rasm).



1-rasm. Navbat tuzilmasi

Deklar haqida tushuncha. Dek so‘zi (DEQ - Double Ended Queue) ingliz tilidan olingan bo‘lib, u ikkita chetga ega navbat, – degan ma’noni

bildiradi. Dekning o‘ziga xos xususiyatlaridan biri shundaki, unga elementlar har ikkala tomondan chapdan va o‘ng tomondan kiritilishi va chiqarilishi lozim bo‘ladi.



2-rasm. Dek tuzilmasi

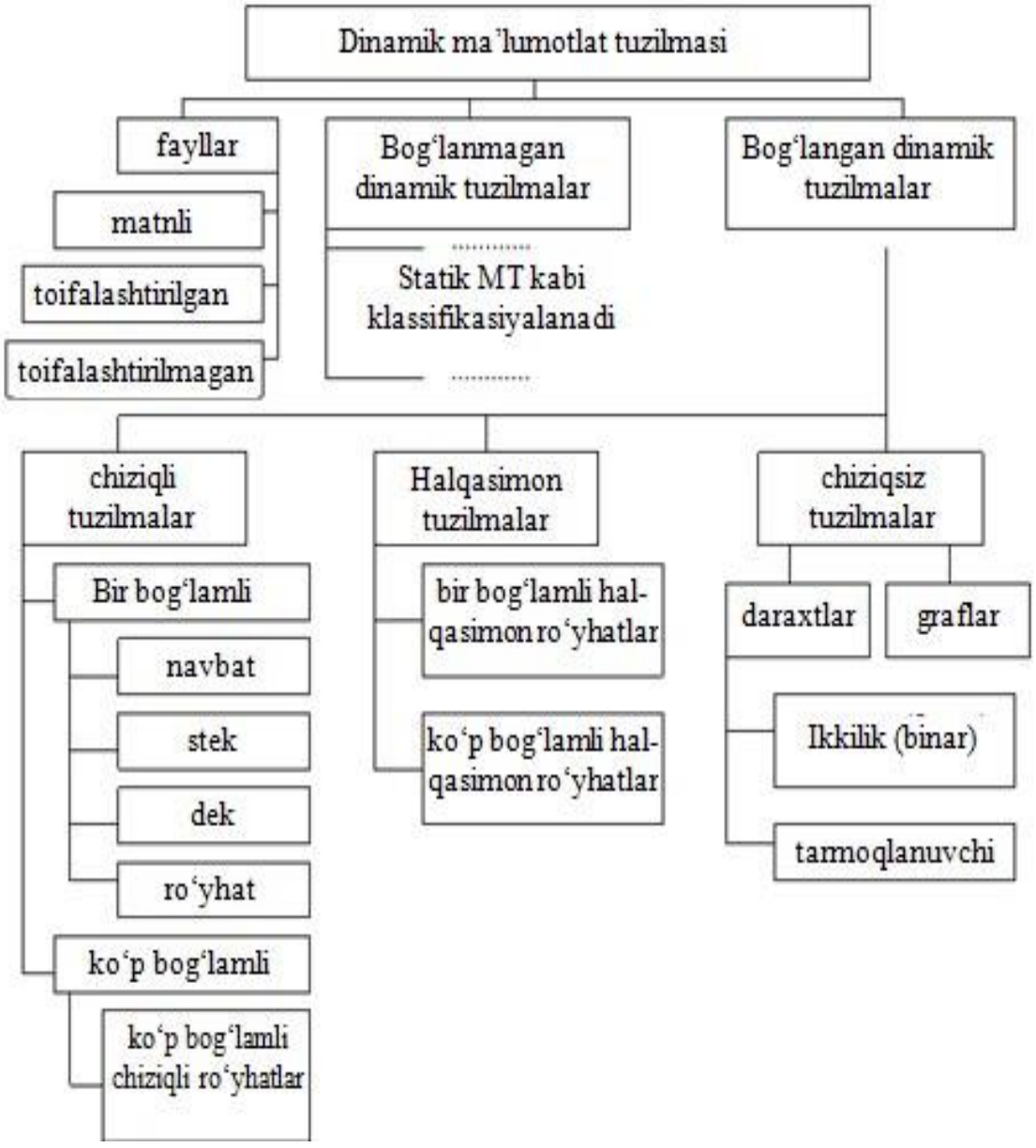
Dinamik ma’lumotlar tuzilmasi

Statik ma’lumotlar tuzilmasi vaqt o‘tishi bilan o‘zining o‘lchamini o‘zgartirmaydi. Har doim dastur kodidagi statik ma’lumotlar tuzilmasiga qarab, ularning o‘lchamini bilish mumkin bo‘ladi. Bunday ma’lumotlarga teskari ravishda dinamik ma’lumotlar tuzilmasi mavjud bo‘lib, bunda dastur bajarilishi davomida dinamik ma’lumotlar tuzilmasi o‘lchamini o‘zgartirishi mumkin**.**

Dinamik ma’lumotlar tuzilmasi – bu qandaydir bir formulalarga asoslanib hosil qilingan, ammo elementlari soni, o‘zaro joylashuvi va o‘zaro aloqasi dastur bajarilishi maboynida shu qonuniyat asosida dinamik o‘zgaruvchan bo‘lgan ma’lumotlar tuzilmasiga aytiladi. Dinamik ma’lumotlar tuzilmasi 3-rasmdagidek ko‘rinishda bo‘ladi.

Dasturlarda dinamik ma’lumotlar tuzilmasidan ko‘pincha chiziqli ro‘yxatlar, steklar, navbatlar va binar daraxtlar ishlatiladi. Bu tuzilmalar bir-biridan elementlarning bog‘lanish yo‘llari va ular ustida bajarilishi lozim bo‘lgan amallari bilan farqlanadi. Dinamik tuzilmalarga e’tibor beriladigan bo‘lsa, ular massiv va yozuvdan farqli ravishda operativ xotirada ketma-ket sohalarda joylashmaganini ko‘rish mumkin. Ixtiyoriy dinamik tuzilma elementi ikkita maydondan tashkil topgan bo‘ladi: tuzilma tashkil etilishiga sabab bo‘layotgan **informatsion maydon** va elementlarning o‘zaro aloqasini ta’minlovchi bu **ko‘rsatkichli maydondir**. Chiziqli ro‘yxatlarda har bir element o‘zidan oldingi yoki keyingisi bilan ham bog‘langan bo‘lishi ham mumkin. Agar har bir element o‘zidan oldingi va o‘zidan keyingi element bilan bog‘langan

bo‘lsa, u holda bunday ro‘yxatlarga ikki **bog‘lamli ro‘yxatlar** deyiladi. Birinchi holatda, ya’ni elementlar o‘zidan keyingi element bilan bog‘langan bo‘lsa, bunday ro‘yxatga **bir bog‘lamli ro‘yxat** deyiladi. Agar oxirgi element birinchi element ko‘rsatkichi bilan bog‘langan bo‘lsa, bunday ro‘yxatga **halqasimon ro‘yxat** deyiladi. Ro‘yxatning har bir elementi shu elementni identifikatsiyalash uchun **kalit**ga ega bo‘ladi. Kalit odatda, butun son yoki satr ko‘rinishida ma’lumotlar maydonining bir qismi sifatida paydo bo‘ladi.



3-rasm. Dinamik ma’lumotlar tuzilmasining ko‘rinishi

Ro‘yxatlar ustida quyidagi amallarni bajarish mumkin:

- ro‘yxatni shakllantirish, ya’ni birinchi elementini yaratish;
- ro‘yxatni oxiriga yangi element qo‘shish;
- berilgan kalitga mos elementni o‘qish;
- ro‘yxatning ko‘rsatilgan joyiga element qo‘shish, ya’ni berilgan kalitga mos elementdan oldin yoki keyin qo‘shish;
- berilgan kalitga mos elementlarni o‘chirish;
- kalit bo‘yicha ro‘yxat elementlarini tartibga keltirish.

Ro‘yxatlar bilan ishlashda dasturda boshlang‘ich elementni ko‘rsatuvchi ko‘rsatkichlar talab etiladi. Chiziqli bir bog‘lamli ro‘yxatlar ustida turli amallarni bajarish algoritmlari va dasturlarini ko‘rib chiqaylik.

Dek va ular ustida misollar yechish

Dek ustida bajariladigan amallar quyidagilardan tashkil topgan:

1. Dekda chapdan element kiritish.
2. Dekda o‘ngdan element kiritish.
3. Dekda chapdan element chiqarish.
4. Dekda o‘ngdan element chiqarish.
5. Dekni bo‘shligini tekshirish.
6. Dekni to‘laligini tekshirish.

C dasturlash tilida dekni statik ko‘rinishda, ya’ni bir o‘lchamli massiv ko‘rinishida amalga oshirishga misol ko‘raylik:

Berilayotgan butun sonlar ketma-ketligining birinchi yarmini dekning chap tomonidan, qolgan yarmini esa dekning o‘ng tomonidan kiritaylik. Dekning elementlarini bir safar chapdan, bir safar esa o‘ngdan juftlikka tekshirib, toq elementlari o‘chirilsin.

Dek bo‘yicha keltirilgan misolning algoritmi.

1. Dekka nechta element kiritilishi aniqlanadi – n, i0.

2. i; agar i<n bo‘lsa va dek to‘lmagan bo‘lsa, 3-qadamga, aks holda 4-qadamga o‘tish jarayoni bajariladi.

3. Agar i<n/2 bo‘lsa, navbatdagi element chapdan, aks holda, ya’ni i>n/2 bo‘lsa, dekning o‘ng tomonidan kiritiladi va shundan so‘ng 2-qadamga o‘tiladi.

4. Agar dek bo‘sh bo‘lmasa, chapdan element chiqarib olinadi. Agar element juft bo‘lsa, b[] massivga joylanadi, so‘ng 5-qadamga o‘tiladi. Agar dek bo‘sh bo‘lsa, 6-qadamga o‘tish jarayoni ko‘rib chiqilsin.

5. Agar dek bo‘sh bo‘lmasa, o‘ngdan element chiqarib olinadi. Agarda element juft bo‘lsa, b[] massivga joylay olsa bo‘ladi. Shundan so‘ng, u 5-qadamga o‘tiladi. Agarda dek bo‘sh bo‘lsa, u holda 6-qadamga o‘tish jarayoni boshlanadi.

6. b[] massiv elementlarini dekka o‘ng tomondan kiritish lozim.

7. Oxirgi amalga dek tarkibi ekranga chiqariladi.

Dekni C dasturlash tilida yozilgan dastur kodini ko‘rib chiqaylik.

```
#include <cstdlib>
#include <iostream>
 using namespace std;
int a[10],n,R0;
 bool isEmpty(){
     if(R0) return true; else return false;
     }
bool isFull(){
    if(R>10) return true; else return false;
     }
int kirit_left(int s){
  if(isFull()){cout<<"\ndek to‘ldi"; nR;return EXIT_SUCCESS;}
    for(int iR;i>0;i--)
  a[i]a[i-1];
     a[0]s;R;
    }
int olish_left(){
```

```
   if(isEmpty()){cout<<"\ndek bo‘sh";return EXIT_SUCCESS;}
   int ta[0];
   for(int i0;i<R-1;i)
 a[i]a[i1];
  R--;
return t;  }
int kirit_right(int s){
  if(isFull()){cout<<"\ndek to‘ldi";nR; return EXIT_SUCCESS;}
  a[R]s;R;  }
int olish_right(){
   if(isEmpty()){cout<<"\ndek bo‘sh";return EXIT_SUCCESS;}
   R--;
    return a[R];   }
int print(){
    cout<<endl<<"dek ele-tlari";
     for(int i0;i<R;i)cout<<a[i]<<"  ";
     cout<<endl; }
int main(int argc, char *argv[])
{ int n,s;cout<<"n"; cin>>n;
for(int i0;i<n;i){
           if(!isFull()){
      cout<<"kirit";cin>>s;
if(i>n/2) kirit_right(s);
         else kirit_left(s);}
       else {cout<<"dekda boshqa joy yo‘q\n";break;}
     }  print();
  int b[n/2],k0,c[n/2],p0;
wxile(!isEmpty()){
            int qolish_left();
```

```
 if(q%20) b[k]q;
 if(isEmpty()) break;
   int polish_right();
     if(p%20) b[k]p;
  }              int i0;
 wxile(i<k){
      kirit_right(b[i]);
        i;  }
 print();
 system("PAUSE");
return EXIT_SUCCESS; }
```

Hozir yana dekda mos bo‘lgan quyidagicha masalani ko‘rib chiqaylik. Bu masalada ro‘yxatning maksimal elementini topish ko‘rib chiqiladi.

Ushbu masalaning algoritmi va dasturiy kodi hamda natijasini quyida keltirilgan.

Masalaning algoritmi:

1. Boshlanishida ekranga menyu chiqariladi; Shundan so‘ng, 1 - element qo‘shish amali bajariladi; Keyingi usul 2 - ro‘yxatni ko‘rishdan iborat; 3 -bo‘lib ro‘yxat maksimali topiladi; 0 - chiqish; oxirida tanlash uchun tanla o‘zgaruvchisiga qiymat so‘raladi; Shundan so‘ng, 2-qadamga o‘tish jarayoni boshlanadi.

2. Agar tanla1 bo‘lsa, 3-qadamga, 2 ga teng bo‘lsa, 4-qadamga, 3 tanlansa, 6-qadamga o‘tish amali bajarilsin, 0 tanlansa dastur yakunlansin.

3. Navbatdagi elementni yaratish p; (p ning info maydoniga qiymat so‘rab olib yozish va ptr maydoniga NULL yozish) agar ro‘yxat boshi ko‘rsatkichi lstNULL bo‘lsa, lstp va lastp; aks holda last – ro‘yxat oxirgi elementi ptr maydoniga p ni yozib, p elementni last qilib belgilab olinadi va 1-qadamga o‘tiladi.

4. Agar lst NULL ga teng bo‘lsa, ro‘yxat bo‘shlig‘ini ekranga chiqarib, 1-qadamga o‘tish kerak. Aks holda, plst va 5-qadamga o‘tilsin.

5. Agar p ning ptr maydoni NULL bo‘lmasa, u holda p ning info maydonini ekranga chiqariladi va keyingi elementga o‘tish qadami bajariladi, ya’ni pp->ptr orqali, 5-qadamga o‘tiladi, aks holda, 1-qadamga qaytiladi.

6. maxlst->info, ya’ni max o‘zgaruvchisiga ro‘yxat 1-elementi info maydoni qiymatini o‘zlashtiriladi va plst orqali 7-qadamga yo‘l olinadi.

7. Agar p NULL ga teng bo‘lmasa, 8-qadamga o‘tishga yo‘l ochiladi, aks holda max ni ekranga chiqariladi va 1-qadamga yana qaytiladi.

8. Agar max< p->info bo‘lsa, maxp->info. Shundan so‘ng, keyingi elementga o‘tiladi, ya’ni pp->ptr. 7-qadamga o‘tiladi.

Shunday qilib, yuqoridagi algoritmga mos C dasturlash tilida dastur kodi ko‘rib chiqiladi:

Dastur kodi

```
#include <iostream>
 using namespace std;
 class Node{
 public:  int number;
          Node* next;
};
int main()
{ Node* head  NULL;
     Node* lastPtr  NULL;
     short action -1;
  wxile (1)
      { cout<<"1. element qo‘shish\n";
         cout<<"2. ro‘yxatni ko‘rish\n";
          cout<<"3. ro‘yxat maksimalini topish\n";
            cout<<"0. chiqish\n\n";
```

```
          cout<<"tanlang: ";
          cin>>action;
       if (action 0) {
  system("CLS");
                  break;}
  if (action 1)
     {       system("CLS");
                Node* ptr new Node;
                   int numb -1;
                   cout<<"son kiriting: ";
                 cin>>numb;
                 ptr->number numb;
                ptr->next NULL;
                   if (head 0)
{      head  ptr;
                       lastPtr ptr;
                     system("CLS");
                        continue;   }
                   lastPtr->next ptr;
          lastPtr ptr;
                 system("CLS");
                  continue; }
      if (action 2){
                   Node* ptr NULL;
                  system("CLS");
         if (head0)
                   {  cout<<"\t!!! ro‘yxatda hech narsa yo‘q !!!\n\n";
                     system("PAUSE");
                    system("CLS");
```

```
                    continue; }
      cout<<"* * * * * ro‘yxat * * * * *\n\n";
                ptr head;
          wxile (1) {
cout<<ptr->number<<" ";
if (ptr->next 0)  break;
             ptr ptr->next;      }
       cout<<"\n\n";
          system("PAUSE");
             system("CLS");
                continue;   }
           if (action 3)
       {      system("CLS");
                 Node* p head;
                 Node* q new Node;
                Node* last new Node;
              int maxp->number; qhead;
          wxile(p){
     if(max<p->number){ maxp->number;}
        pp->next;  }
      system("CLS");
              cout<<"max"<<max;
               system("pause");
            continue;
           }
}
 }
```

Mavzug oid test savollari

1. Dekning vazifasi bu ...?

a) chiziqli ma’lumotlar tuzilmasi bo‘lib, unda ma’lumotlarni kiritish va chiqarish uning ikki tomonlama amalga oshiriladi.

b) bu shunday tuzilmaki, u elementlar qo‘shilishi bilan kengayib boradi va elementlarni faqatgina bir tomondan qabul qiladi.

c) dekda chiziqli ma’lumotlar tuzilmasi bo‘lib, ma’lumotlarni kiritish va chiqarish uning bir tomonidan amalga oshiriladi.

d) dek chiziqli ma’lumotlar tuzilmasi bo‘lib, ma’lumotlarni kiritish va chiqarish uning faqat o‘rtasiga amalga oshiriladi.

2. Chiziqsiz ma’lumotlar tuzilmasiga nimalar kiradi?

a) Daraxtlar va graflardan

b) Stek, dek, navbatlardan

c) Yozuv, jadvallardan

d) Graf va vektorlardan

3. Qanday konteyner yordamida grafning tubiga qarab ko‘rishda qo‘llaniladi?

a) stek konteyneri yordamida

b) navbat konteyneri yordamida

c) ro‘yxat konteyneri yordamida

d) dek konteyneri yordamida

4. Qanday konteyner yordamida grafda eniga qarab ko‘rishda qo‘llaniladi?

a) navbat konteyneri yordamida

b) stek konteyneri yordamida

c) ro‘yxat konteyneri yordamida

d) dek konteyneri yordamida

5. C dasturlash tilida standart andozalar kutubxonasi yordamida dekni qanday e’lon qilish mumkin?

a) deque□int □S;
b) queue□int □S;
c) stack □ int □ S;
d) list □ int □ S;

Nazorat savollari

1. Navbat tuzilmasini tushuntiring.
2. Yarimstatik ma’lumotlar tuzilmasini dasturda e’lon qilishning qanday usullarini bilasiz?
3. Dek nima va navbat tuzilmasidan farqi nimada?

Xulosa

Yuqorida navbatga aniq ta’rif berildi. Bu ta’rif, navbat nima ekanligini bildiradi, lekin bu navbatdan juda ko‘p hollarda foydalaniladi. Ammo, yangi o‘rganuvchilar uchun uni tushunish kishiga biroz noqulaylik tug‘diradi. O‘zgaruvchan uzunlikdagi ketma-ketliklar mavjud bo‘lib, bunday ketma-ketlik ro‘yxatidagi tuzilmaga elementlar faqat bir tomondan, navbatning oxiridan qo‘shiladi va elementlarni tuzilmadan chiqarish boshqa tomondan, ya’ni navbat boshidan amalga oshiriladi.

Navbat ustida yangi elementlarni qo‘shish, elementni chiqarib tashlash, uzunligini aniqlash, navbatni tozalash kabi asosiy amallar bajarilad. Bu yerda dek ro‘yxati ham mavjud bo’lib, bunda ro‘yxatni har ikkala tomondan navbat ham qabul qilinadi, ham chiqariladi. Dek ustida ham bir nechta amallarni bajarish mumkin.

4-BOB. DARAXTSIMON MA’LUMOTLAR TUZILMALARI

1-§. Daraxtsimon ma’lumotlar nazariyasi

1.1. Daraxtsimon ma’lumot tuzilmalari ta’riflari va xususiyatlari. Daraxtlar klassifikatsiyasi. Daraxt ko‘ruvi

Reja:

1. Daraxt haqida tushuncha va uni misollar orqali tasvirlash.
2. Daraxtlarni tasvirlash usullari.
3. Binar daraxtlar haqida tushuncha va ularni tasvirlash.
4. m-o‘lchamli daraxtni binar ko‘rinishga keltirish usuli.
5. Binar daraxtlar ustida amallar bajarish.

Tayanch iboralar: daraxt, klassifikatsiya, binar daraxt, ildiz, tugun, ideal, muvozanat, daraxt ko‘ruvi, qism daraxt.

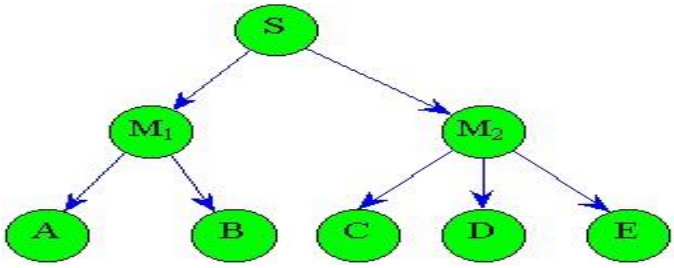
Daraxtlar haqida tushuncha va uni misollar orqali tasvirlash

Daraxt – bu chiziqsiz bog‘langan ma’lumotlar tuzilmasidir.

Daraxt quyidagi belgilar bilan tasniflash mumkin:

- daraxtda shunday bitta yagona element mavjudki, unga boshqa elementlardan umuman murojaat yo‘q. Bunday yagona elementga daraxt ildizi deb aytiladi;
- daraxtda ixtiyoriy elementga chekli sondagi ko‘rsatkichlar yordamida murojaat qilish mumkin;
- daraxtning har bir elementi faqatgina o‘zidan oldingi kelgan faqat bitta element bilan bog‘langan bo‘ladi. Daraxtning har bir tuguni, bu oraliq yoki terminal bo‘ladi.

Quyida chizilgan chizmada M1va M2 – oraliq deb yuritilsa, A, B, C, D, E lar esa uning barglari deyiladi. Bu yerda terminal tugunning o‘ziga xos tasnifi bo‘lib, u daraxtni shoxlari yo‘qligidan darak beradi.



1-rasm. Daraxt

Daraxt balandligi – bu daraxtning bosqichidagi soniga aytiladi. Yuqoridagi chizmadagi daraxtning balandligi bu ikkiga tengdir.

Daraxt tugunlaridan chiqayotgan shoxlar soni bu tugundan chiqish darajasi deb yuritiladi. Yuqorida keltirilgan chizmada M1 uchun chiqish darajasi bu 2 ga tengdir. Bu daraxt M2 uchun esa 3 ga teng deb hisoblanadi. Daraxtlar chiqish darajasi bo‘yicha quyidagi sinflarga ajratiladi:

1) agar daraxning maksimal chiqish darajasi m ga teng bo‘lsa, u holda bunday daraxt m-chi tartibli daraxt deyiladi;

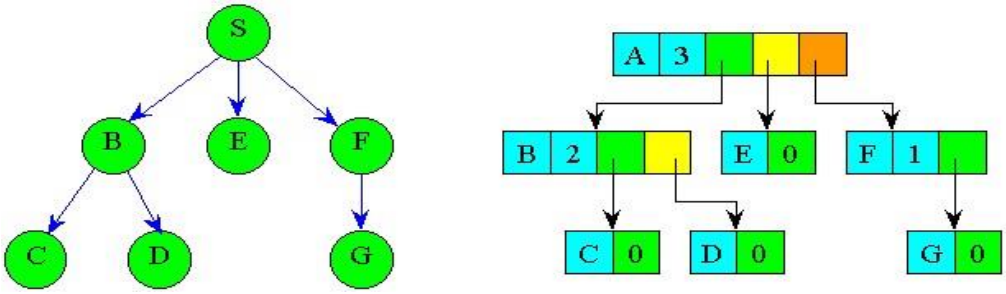
2) agar daraxtning chiqish darajasi 0 yoki m bo‘lsa, u holda to‘liq m-chi tartibli daraxt deb yuritiladi;

3) agar maksimal chiqish darajasi ikki bo‘lsa, u holda bunday noyob daraxt binar daraxt deb hisoblanadi;

4) agar chiqish darajasi 0 yoki 2 bo‘lsa, bunday daraxt to‘liq binar daraxt deyiladi.

Bundan tashqari tugunlar orasidagi bog‘liqlikni tavsiflash uchun yana quyidagicha termindan foydalansa bo‘ladi. Bunda M1 – A va V elementlar uchun “ota”. A va V – esa M1 tugun “o‘g‘illari” deb hisobga olish mumkin.

Daraxtlarni tasvirlash



2-rasm. Daraxtlarni tasvirlash

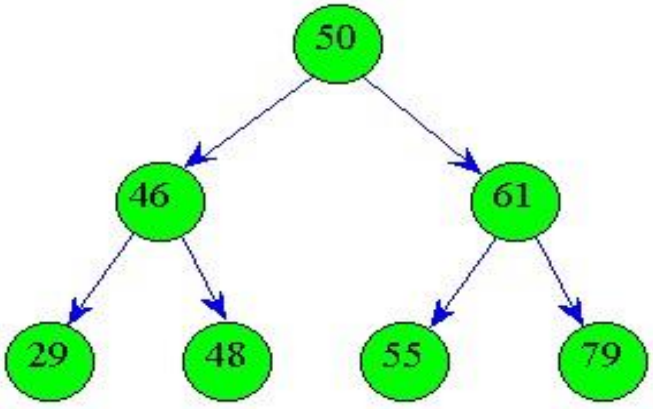
2-rasmda daraxtlarni grafik shakldagi hamda uning chiziqsiz ro‘yxat shaklidagi ko‘rinishlari ko‘rsatilgan.

Elektron hisoblash mashinasi xotirasida daraxtni ifodalashning eng qulay usuli - bu uni bog‘langan ro‘yxatlar ko‘rinishida ifodalashdir. Ro‘yxat elementi tugun qiymati va chiqish darajasini o‘z ichiga oluvchi informatsion maydonga hamda chiqish darajasiga teng bo‘lgan ko‘rsatkichlar maydoniga ega bo‘ladi. Bunga misol qilib, yuqoridagi chizmani keltirsak bo‘ladi, ya’ni elementning har bir ko‘rsatkichi ushbu elementni tugun, ya’ni o‘g‘illari bo‘lgan tugunlardagi yo‘nalishini anglatadi.

Binar daraxtlar haqida tushuncha va uni tasvirlash

Binar daraxti eng ko‘p foydalaniladigan daraxtlar turkumiga kiradi. Daraxtlarni elektron hisoblash mashinasi xotirasida tasvirlanishiga ko‘ra, har bir elementlar to‘rttadan maydonga ega yozuvlar hisoblanadi. Mazkur maydonlar qiymati mos ravishda yozuv kaliti bo‘lib, boshqa elementlarga murojaatni ifodalab beradi, ya’ni chapga-pastga, o‘nga-pastga va yozuv matniga qaratilgan bo‘ladi.

Shuni esda tutish kerakki, daraxt hosil qilinayotganda, otaga nisbatan chap tomondagi o‘g‘il qiymati kichik kalitga, o‘ng tomondagi o‘g‘il esa qiymati katta kalitga mos deb olinadi. Misol uchun quyidagi elementlardan tashkil topgan binar daraxtini qurib ko‘raylik: 50, 46, 61, 48, 29, 55, 79. Bu daraxtni qurganda u esa quyidagi ko‘rinishga ega bo‘ladi:

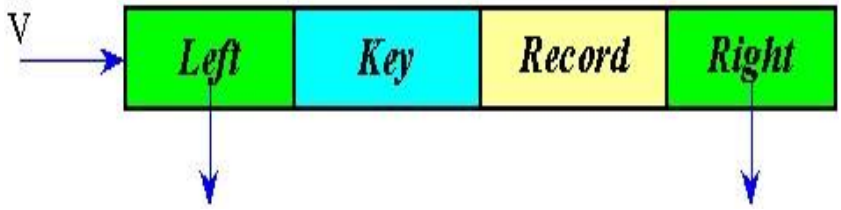


3-rasm. Binar daraxtlarni tasvirlash

Natijada, bunda o‘ng va chap qism daraxtlari bir xil bosqichli tartiblangan binar daraxti ko‘rinishini hosil qiladi.

Ideal muvozanatlangan daraxt deb, agar daraxtning o‘ng va chap qism daraxtlari bosqichlari farqi birdan kichik bo‘lgan daraxtga aytiladi. Yuqorida hosil qilingan binar daraxt ideal muvozanatlangan daraxtga eng ajoyib misol bo‘la oladi.

Binar daraxtni hosil qilish uchun elektron hosoblash mashinasi xotirasida elementlar quyidagi tarzda bo‘lishi shart:



4-rasm. Binar daraxtni elektron hosoblash mashinasi xotirasida tasvirlash

V MakeTree(Key, Rec) amali ikkita ko‘rsatkichli (kalit) va ikkita maydonli (informatsion) element yaratib beradi (daraxt tuguni).

m-o‘lchamli daraxtni binar ko‘rinishga keltirish usuli

m-o‘lchamli daraxtni binar ko‘rinishga keltiraylik.

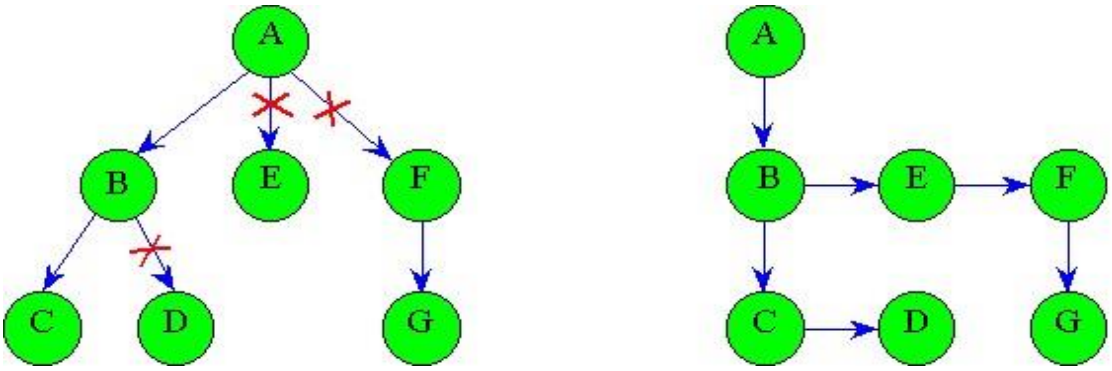
Bunda noformal algoritmdan foydalaniladi:

1. Ko‘riladigan daraxtning har bir tugunida katta o‘g‘ilga mos chetki chap shoxidan tashqari barcha shoxlari kesib chiqarib tashlanadi.

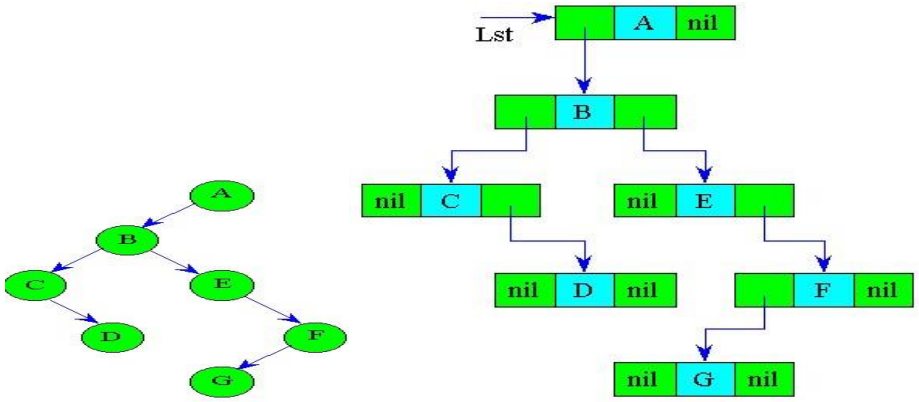
2. Bitta ota barcha o‘g‘illari gorizontal chiziq bilan bog‘lanadi.

3. Agar u mavjud bo‘lsagina, hosil qilingan tuzilmaning har bir tugunida yuqoridagi o‘g‘il mazkur tugunning quyida turgan tugun hisoblanadi.

Bunday daraxtda algoritm amallar ketma-ketligi quyidagi ko‘rinishga keltiriladi.



5-rasm. m-o‘lchamli daraxtni binar ko‘rinishga keltirish usuli



6-rasm. Daraxtda m-o‘lchovli daraxtni binar ko‘rinishga keltirish usuli

Binar daraxtlar ustidagi amallar bajarish

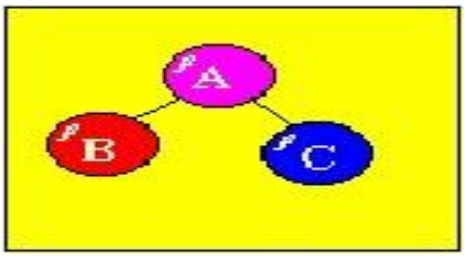
Daraxtlar ustida bajariladigan quyidagi amallar jamlammasi bajariladi:

1. Daraxt ko‘ruvi.
2. Daraxtdan qism daraxtni o‘chirish.
3. Daraxtga qism daraxt qo‘yish.

Daraxt ko‘ruvini amalga oshirish uchun quyidagi uchta protsedurani bajarish kerak:

1. Ildizni qayta ishlash kerak.
2. Chap tarmoq(shox)ni qayta ishlash kerak.
3. O‘ng tarmoq(shox)ni qayta ishlash kerak.

Yuqoridagi protsedura qanday ketma-ketlikda amalga oshirilishiga qarab daraxtga quyidagi ko‘ruv uchta ajoyib ko‘rinishga ajratiladi.



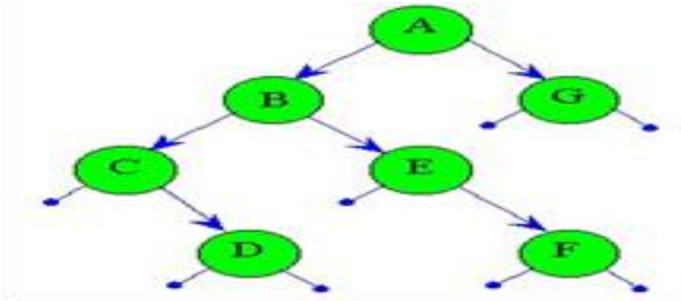
7-rasm. Binar daraxtlar ustidagi amallar bajarish

1. Daraxt uchun yuqoridan quyiga qarab. Bunda protseduar quyidagi ketma-ketlikda amalga oshiriladi. A-B-C.

2. Chapdan o‘ng. Protsedura quyidagi ketma-ketlikda amalga oshiriladi. B-A-C.

3. Quyidan yuqoriga. Protsedura quyidagi ketma-ketlikda amalga oshiriladi. B-C-A.

Masalan, quyidagi daraxt ustida ko‘ruv o‘tkazaylik.



8-rasm. Daraxt ko‘ruvi

Daraxat ko‘ruvi tartibi: Yuqoridan pastga: A,B,C,D,E,F,G.

Pastdan yuqoriga: D,C,F,E,B,G,A.

Daraxt ko‘rigini rekursiv protseduralarini ko‘rib chiqaylik:

```
1.      int pretrave(node *tree){
     if(tree!NULL) {int a0,b0;
    if(tree->left!NULL) atree->left->info;
   if(tree->right!NULL) btree->right->info;
   cout<<tree->info<<"  - chapida "<<a<<"  - o‘ngida "<<b<<"   \n";
                  pretrave(tree->left);
        pretrave(tree->right);
                    }
```

```
        return 0;
        };
2.      int intrave(node *tree){
 if(tree!NULL) {
intrave(tree->left);
                cout<<tree->info;
              intrave(tree->right);
                 }
      return 0;
        };
3.      int postrave(node *tree){
      if(tree!NULL) {
          postrave(tree->left);
                postrave(tree->right);
           cout<<tree->info;
                   }
     return 0;
        };
```

Mavzuga oid test savollari

1. Oddiy sozlangan ma’lumotlar turlariga quyidagilardan qaysilari kiradi?

a) mantiqiy, butun, haqiqiy, belgili, ko‘rsatkichli turlar

b) massiv, yozuv, rekursiv turlar, to‘plami

c) jadval, stek, navbat, ruyxat, deklar

d) daraxtlar va graflar

2. **Sozlangan tuzilmaviy ma’lumotlar tuzilmalariga quyidagilardan qaysilari kiradi?**

a) massiv, yozuv, rekursiv turlar va to‘plam

b) jadval, stek, navbat, ro‘yxat va deklar

c) daraxtlar va graflar

d) mantiqiy, butun, haqiqiy, belgili, ko‘rsatkichli turlar

3. Ro‘yxat elementlarning ro‘yxatlari bo‘lishi mumkin tuzilma qanday nomlanadi?

a) Lug‘atlar

b) Daraxtlar

c) Graflar

d) Ro‘yxatlar

4. Daraxtsimon tuzilmadagi shunday elementga murojaat yo‘qki, u... tugun hisoblanadi?

a) ildizli

b) oraliqli

c) so‘nggi

d) ildiz bo‘lmaganda

5. Daraxtsimon tuzilmada boshqa elementlarga murojaat bo‘lmasa, u... tugun hisoblanadi?

a) barg

b) oraliq

c) ildiz

d) terminal

Nazorat savollari

1. Daraxt nima? Uning o‘ziga xos xususiyatlarini aytib bering.
2. To‘liq daraxt deganda nimani tushunasiz?
3. Daraxt ko‘ruvi nimadan iborat?
4. Har qanday daraxtni binar ko‘rinishga keltirish mumkinmi?
5. Daraxt tuguni qanday hosil qilinadi?
6. Daraxtda qanday amallarni bajarish mumkin?

Xulosa

Ma’lumki, daraxt – bu chiziqsiz bog‘langan ma’lumotlar tuzilmasidir. Unda shunday bitta yagona element mavjudki, bunda boshqa elementlardan umuman murojaat yo‘q. Bunday yagona elementga daraxt ildizi deb aytiladi.

Daraxtda ixtiyoriy elementga chekli sondagi ko‘rsatkichlar yordamida murojaat qilish mumkin. Daraxtning har bir elementi faqatgina o‘zidan oldingi kelgan faqat bitta element bilan bog‘langan bo‘ladi.

Daraxtning har bir tuguni, bu oraliq yoki terminal bo‘ladi. Shunday ekan, ma’lumotlar tuzilmasi va algoritmlari fanida ma’lumotlar ustida ishlaganda bunday holat juda ko’p marotaba uchraydi. Bunda ma’lumotni tez saralash osonroq kechadi.

1.2. Binar qidiruv daraxti. Binar qidiruv daraxtiga element qo‘shish, element o‘chirish va qidiruv algoritmlari

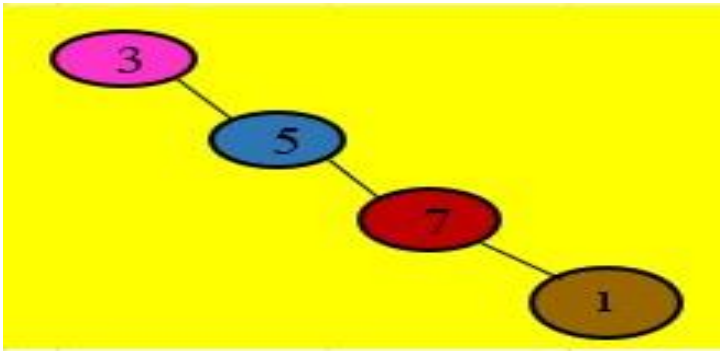
Reja:

1. Binar qidiruv daraxtiga oid misollar yechish.
2. Binar qidiruv daraxtiga element qo‘shish, element o‘chirish va qidiruv algoritmlari.

Tayanch iboralar: daraxt, daraxt ko‘ruvi, binar daraxt, klassifikatsiya, ildiz, tugun, ideal, muvozanat, qism daraxt, ikkilik daraxt.

Binar qidiruv daraxtiga oid misollar yechish

Mazkur protseduraning vazifasi shundan iboratki, u berilgan kalit bo‘yicha daraxtning tugun qidiruvini amalga oshiradi. Bunda qidiruv operatsiyasining davomiyligi daraxt tuzilishiga bog‘liqdir. Haqiqatdan ham agar elementlar daraxtga kalit qiymatlari o‘sish yoki kamayish tartibida kelib tushgan bo‘lsa, u holda bunday daraxt bir tomonga yo‘nalgan ro‘yxat hosil qiladi, ya’ni chiqish darajasi bir bo‘ladi va u yagona shoxga egadir. 1-rasmda binar qidiruv daraxtiga oid misol keltirilgan.



1-rasm. Binar qidiruv daraxtiga oid misol yechish

Bunday holda daraxtda qidiruv vaqti, bir tomonlama yo‘naltirilgan ro‘yxatdagi kabi bo‘lib, o‘rtacha qarab chiqishlar soni N/2 ko‘rinisgda bo‘ladi. Agar daraxt muvozanatlangan bo‘lsa, u holda bunda qidiruv eng samarali natija beradi. Shu zahoti qidiruv $\log_2 N$ dan ko‘p bo‘lmagan elementlarni ko‘rib chiqadi. Endi qidiruv protsedurasini ko‘rib chiqaylik. Bunda search o‘zgaruvchisiga topilgan bo‘g‘in ko‘rsatkichi to‘g‘ridan-to‘g‘ri o‘zlashtiriladi, ya’ni:

```
int search(node *tree, int key) {
 node *next; nexttree;
wxile(next!NULL) {          if (next->infokey){cout<<"Binar daraxtda "<<key<<"
bor bo‘lsa";   return next; }
        if (next->info>key) nextnext->left;
           else  nextnext->right;
    }
cout<<"tuzilmada izlangan element mavjud bo‘lmasa!!!"<<endl;
```

Binar qidiruv daraxtiga element qo‘shish, element o‘chirish va qidiruv algoritmlari

Quriladigan daraxtga yangi element qo‘shish protsedurasi.

Daraxtga biror-bir elementni qo‘shishdan oldin daraxtda berilgan kalit bo‘yicha qidiruvni amalga oshirish kerak bo‘ladi. Agar berilgan kalitga teng

kalit mavjud bo‘lsa, u holda dastur o‘z ishini to‘xtatadi, aks holda daraxtga element qo‘shish jarayoni amalga oshaveradi.

Daraxtga yangi yozuvlarni kiritish uchun, eng avvalo, daraxtni shunday tugunini topish lozimki, natijada mazkur tugunga yangi element qo‘shish imkoni mavjud bo‘lsin.

Kerakli tugunni qidirish algoritmi ham xuddi berilgan kalit bo‘yicha tugunni topish algoritmi kabi amalga oshirilsin. Biroq, berilgan kalit bo‘yicha qidiruv protsedurasidan to‘g‘ri foydalanib bo‘lmaydi, sababi, qidiruv protsedurasida, qaysi tugunda murojaat NIL (search nil) bo‘lgani fiksirlanmay qoladi.

Qidiruv protsedurasini shunday modifikasiya qilish kerakki, unda qo‘shimcha samara sifatida yangi protsedura berilgan kalit turgan tugunni fiksirlasin, ya’ni qidiruv muvofaqiyatli bo‘lsa yoki shunday tugunniki, ushbu tugunni qayta ishlagandan keyin qidiruv yakunlansin va qidiruv muvofaqiyatli amalga oshsin.

Quyida daraxtda qo‘shilayotgan element kalitiga teng kalitli element yo‘q bo‘lgan holda elementni qo‘shish protsedurasini keltirib o‘tildi.

```
Node *qNULL;
 Node *ptree;
wxile(p!NULL){
   qp;
     if(keyp->key) {
       searchp;
       return 0;
          }
      If(key<p->key)  pp->left;
       else pp->right;
    }
{
```

Berilgan kalitga teng tugun topilmadi, lekin unda element qo‘shish talab qilinadi. Ota bo‘lishi mumkin bo‘lgan tugunga q ko‘rsatkich beriladi.} node *qnew node;

Qo‘yilayotgan yangi element chap yoki o‘ng o‘g‘il bo‘lishini aniqlash lozim bo‘ladi.

```
If(key<q->key) q->leftyangi;
   else q->rightyangi;
searchyangi;
return 0;
```

Binar daraxtdan elementni o‘chirish protsedurasini ko‘rib chiqaylik. Daraxtdan tugunni o‘chirib tashlash natijasida daraxtning tartiblanganlik darajasi buzilmasligi kerak.

Daraxtdan tugun o‘chirilayotganda 3 xil usulda bo‘lishi mumkin:

1) Topilgan tugun terminal(barg) bo‘lishi shart. Bunday holatda tugun shunchaki o‘chirib tashlanadi.

2) Topilgan tugun daraxt uchun faqatgina bitta o‘g‘ilga ega bo‘lishi kerak. U holda o‘g‘il ota o‘rniga o‘tkaziladi.

3) O‘chirilayotgan tugun esa ikkita o‘g‘ilga ega bo‘lishi kerak. Bunday holatda shunday qism daraxtlar zvenosini topish lozim bo‘lsinki, uni o‘chirilayotganda tugun o‘rniga qo‘yish mumkin bo‘lsin. Daraxt uchun bunday zveno har doim mavjud bo‘ladi:

- bu yoki chap qism daraxtning eng o‘ng tomondagi elementi, ya’ni ushbu zvenoga erishish uchun keyingi uchiga chap shox orqali o‘tib, navbatdagi uchlariga esa, murojaat NIL bo‘lmaguncha, faqatgina o‘ng shohlari orqali o‘tish kerak bo‘ladi.

- yoki o‘ng qism daraxtning eng chap elementi ya’ni ushbu zvenoga erishish uchun keyingi uchiga o‘ng shoh orqali o‘tib, navbatdagi uchlariga esa murojaat NIL bo‘lmaguncha, faqatgina chap shohlari orqali o‘tish zarur bo‘ladi.

O‘chirlayotgan element chap qism daraxtining eng o‘ngidagi element o‘chirilayotgan element uchun “Predshestvennik” bo‘ladi, ya’ni 12 uchun – 11 bo‘ladi. Bunda merosxo‘r daraxt esa uning o‘ng qism daraxtning eng chapidagi tugunidir (12 uchun - 13).

Daraxtda merosxo‘rni topish algoritmi quyida ishlab chiqilgan:

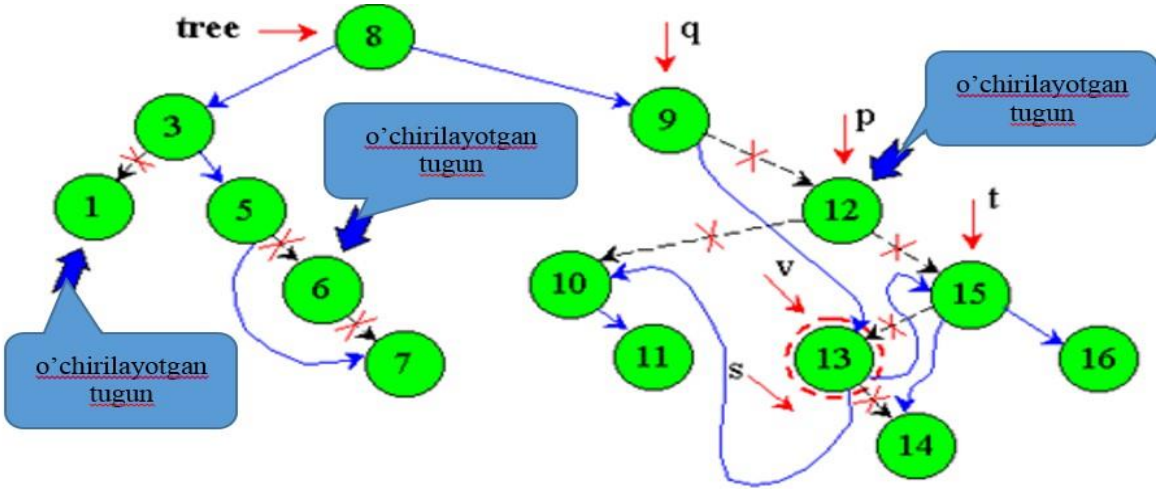
p – bu ishchi ko‘rsatkich;

q - r dan bir qadam orqadagi ko‘rsatkich;

v – o‘chiralayotgan tugun merosxo‘rini ko‘rsatadi;

t - v bir qadam orqada yuradi;

s - v dan bir qadam oldinda yuradi, ya’ni chap o‘g‘ilni yoki bo‘sh joyni ko‘rsatib boradi.



2-rasm. Daraxtda merosxo‘rni topish kemma-ketligi

Yuqoridagi daraxt bo‘yicha qaralganda v ko‘rsatkich 13 tugunni, s esa bo‘sh joyni ko‘rsatishi lozim bo‘ladi.

1) Elementni qidirish protsedurasi orqali o‘chirilayotgan element topiladi. Bunda r ko‘rsatkich o‘chirilayotgan elementni ko‘rsatadi.

2) O‘chiriladigan elementni o‘rniga qaytarib qo‘yadigan tugunga v ko‘rsatkich joylanadi, ya’ni bu C dasturlash tili orqali amalga oshiriladi:

C dasturlash tilida.

```
#include <iostream>

// Ikkilik qidiruv daraxti uchun tugun tuzilishi
```

```
 struct Node {
   int data;
  Node* left;
 Node* right;
};
// Yangi tugun yaratish uchun yordamchi funksiya
Node* newNode(int data) {
   Node* node = new Node;
   node->data = data;
    node->left = node->right = nullptr;
 return node;
}
// Tugunning tartibli vorisini rekursiv topish funksiyasi
Node* findInOrderSuccessor(Node* node) {
Node* current = node->right;
  while (current && current->left) {
        current = current->left;
    }
    return current;
}
// Ikkilik qidiruv daraxtidan tugunni o‘chirish funksiyasi
Node* deleteNode(Node* root, int key) {
    if (root == nullptr) { return root;
 // Tugun topilmadi
    }

    // O‘chirish uchun tugunni takroran qidiring
 if (key < root->data) {
        root->left = deleteNode(root->left, key);
```

```
        } else if (key > root->data) {
            root->right = deleteNode(root->right, key);
        } else {// Node found
            // 1-holat: tugunning vorisi yo‘q
            if (root->left == nullptr && root->right == nullptr) {delete root; return nullptr;
            }
            // 2-holat: Tugun faqat bitta vorisga ega
            if (root->left == nullptr) {
     Node* temp = root->right;
                delete root;
           return temp;
            } else if (root->right == nullptr) {
        Node* temp = root->left;
                delete root;
              return temp;
            }
            // 3-holat: Tugunning ikkita vorisi bor
            Node* inOrderSuccessor = findInOrderSuccessor(root);
          root->data = inOrderSuccessor->data;
        // Joriy tugun ma’lumotlarini navbatdagi voris ma’lumotlari bilan almashtiring
            root->right = deleteNode(root->right, inOrderSuccessor->data);
        //Tartibdagi o‘rinbosarni o‘chiring
        }
        return root;
    }
    // DeleteNode funksiyasini namoyish qilish uchun asosiy funksiya int main() {
```

```
    Node* root = newNode(50);
    root->left = newNode(30);
    root->right = newNode(70);
    root->left->left = newNode(20);
    root->left->right = newNode(40);
    root->right->left = newNode(60);
    root->right->right = newNode(80);
    // 50 tugmasi bilan tugunni
root = deleteNode(root, 50);
    //... (O‘zgartirilgan daraxtda keyingi amallarni bajaring)
    return 0;
}
```

Mavzuga oid test savollari

```
   B
1. / \
  A C
```

Binar daraxt uchun to‘g‘ri (yuqoridan pastga) ko‘ruv amalining natijasini ko‘rsating?

a) BAC

b) ACB

c) ABC

d) CAB

```
   B
2. / \
  A  C
```

Binar daraxt uchun teskari (pastdan yuqoriga) ko‘ruv amalining natijasini ko‘rsating?

a) ACB

b) BAC

c) ABC

d) CAB

B

3 / \

A C **Binar daraxt uchun simmetrik (chapdan o‘nga) ko‘ruv amalining natijasini ko‘rsating?**

a) ABC

b) ACB

c) BAC

d) CAB

4. Kompyuter xotirasida binar daraxtni qanday ko‘rinishda tasvirlash qulay?

a) bog‘langan chiziqsiz ro‘yxatlar

b) massivlar

c) jadvallar

d) bog‘langan chiziqli ro‘yxatlar

5. Agar daraxtni tashkil etuvchi element (tugun)lardan faqat ikkita tugun bilan bog‘langan bo‘lsa, u holda bunday binar daraxt ... deyiladi?

a) to‘liq

b) ikkilik

c) minimal balandlikka ega daraxt

d) muvozanatlangan

Nazorat savollari

1. Binar daraxt qanday daraxt?
2. Daraxtga element qo‘shish qanday amalga ochiriladi?
3. Daraxtdan element o‘chirish qanday amalga oshiriladi?

Xulosa

Bu yerda mazkur jarayonning vazifasi shundan iboratki, u berilgan kalit bo‘yicha daraxtning tugun qidiruvini amalga oshiradi. Bunda qidiruv operatsiyasining davomiyligi daraxt tuzilishiga bog‘liqdir. Haqiqatdan ham agar elementlar daraxtga kalit qiymatlari o‘sish yoki kamayish tartibida kelib tushgan

bo‘lsa, u holda bunday daraxt bir tomonga yo‘nalgan ro‘yxat hosil qiladi, ya’ni chiqish darajasi bir bo‘ladi va u yagona shoxga egadir.

Daraxtga biror-bir elementni qo‘shishdan oldin daraxtda berilgan kalit bo‘yicha qidiruvni amalga oshirish kerak bo‘ladi. Agar berilgan kalitga teng kalit mavjud bo‘lsa, u holda dastur o‘z ishini to‘xtatadi, aks holda daraxtga element qo‘shish jarayoni amalga oshaveradi.

Bu esa ma‘lumotlar tuzilmasidan ma‘lumotni topishni yengillashtiradi. Bunday holatni yuqorida dastur kodini ham yozib natijalar olindi.

1.3. Binar qidiruv daraxti. Muvozanatlangan binar daraxtlar. Muvozanatlash algoritmlari: muvozanatlashning umumiy va xususiy algoritmlari. AVL daraxt

Reja:

1. Muvozanatlangan binar daraxtini C++ dastur yordamida ko‘rish.
2. Muvozanatlangan binar daraxtning balandligi binar daraxti ustida amallar bajarish.
3. AVL daraxti va uni misollar bilan yoritish.

Tayanch iboralar: daraxt, AVL daraxti, binar daraxt, klassifikatsiya, ildiz, tugun, ideal, muvozanatlangan daraxt, qism daraxt, ikkilik daraxt.

Muvozanatlangan binar daraxtini C++ dastur yordamida ko‘rish

Binar daraxti ikki xil ko‘rinishda bo‘lishi mumkin. Biri muvozanatlangan, ikkinchisi AVL-muvozanatlangan ko‘rinishda ega bo‘lishi lozim. Bu yerda daraxtning AVL-muvozanatlangani 1962-yil sovet olimlari Adelson, Velsk Georgiy Maksimovich va Landis Yevgeniya Mihaylovichlar tomonidan taklif qilingan.

Daraxtdagi har bir tugunning chap va o‘ng qismi **daraxtning balandligi** deyiladi. Bunda o‘sha chap va o‘ng qismlarining o‘zaro farqi 1 tadan ko‘p bo‘lmasa maqsadga muvofiqdir.

Berilgan butun sonlar – kalitlar ketma-ketligidan binar daraxt yaratib olinadi va u muvozanatlanadi. Bunda daraxtni muvozanatlashdan maqsad, daraxtga yangi element kiritish yoki daraxtdan elementlarni izlash algoritmining samaradorligini oshirishdan iboratdir, ya’ni bu amallarni bajarishdagi solishtirishlar soni keskin kamayadi.

Aslida binar daraxtning **muvozanatlash algoritmlari** quyidagicha ko‘rinishda bo‘ladi: Bunda birinchi bo‘lib, binar daraxt yaratib olinadi.

So‘ngra binar daraxtni chapdan o‘ngga ko‘rikdan o‘tkaziladi va tugunlarning info maydonlaridan a[..] massiv hosil qilinadi.

Tabiiyki, massiv o‘sish bo‘yicha tartiblangan bo‘lishi lozim. Bunda muvozanatlangan daraxtning tugunlarini belgilash uchun massivni ko‘riladigan oralig‘i start0 va endn-1 ko‘rinishlarida belgilab olinadi.

Shundan so‘ng, massivning ko‘rilayotgan oralig‘i o‘rtasida joylashgan elementni, ya’ni mid(startend)/2 va a[mid] ni muvozanatlangan daraxtning tuguni deb hisoblab olinadi.

Agar ko‘rilayotgan oraliqda bitta ham element qolmagan bo‘lsa, u holda ya’ni start>end bo‘lsa, bajarilishi joriy seansdan keyingisiga o‘tkaziladi.

Ko‘rilayotgan tugunning chap qism daraxtini hosil qilish uchun massivning ko‘rilayotgan oralig‘ining birinchi yarmini olish lozim, ya’ni start0 hamda endmid. Shundan so‘ng, bu yerda yuqoridagi qadamlar birin-keyin takrorlanadi.

Shundn so‘ng, ko‘rilayotgan tugunning o‘ng qism daraxtini hosil qilish uchun massivning ko‘rilayotgan oralig‘ining ikkinchi yarmi hosil qililadi, ya’ni startmid1 va endend deb nomlangan oldingi qadamlarga qaytadi. Bu esa yuqoridagi qadamlarni takrorlashga to‘g‘ri keladi.

Yuqoridagilardan foydalanib, uning dastur kodining bir qismidan namuna keltiriladi.

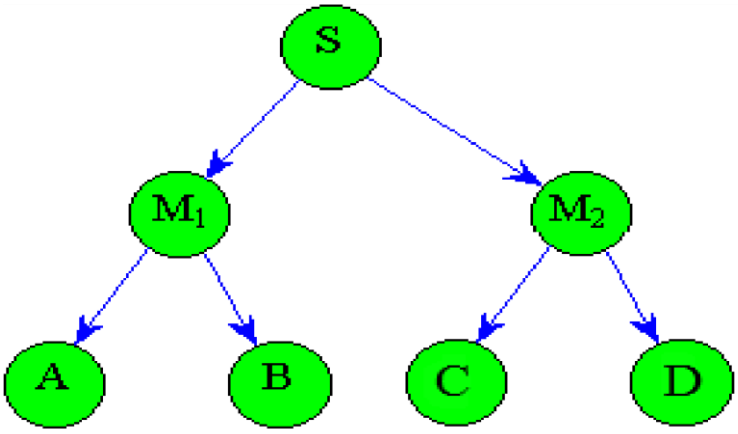
Dastur kodi

```
 node *new_tree(int *arr, int start, int end)
{ if(start>end) return NULL;
else { int mid(startend)/2;
node *treenew node;
  tree->infoarr[mid];
 tree->leftnew_tree(arr,start,mid-1);
tree->rightnew_tree(arr,mid1,end);
return tree;
} }
```

Muvozanatlangan binar daraxtning balandligi binar daraxti ustida amallar bajarish

Binar daraxtning balandligi deb, daraxtning bosqichlari soniga aytiladi. Binar daraxtining balandligini aniqlash uchun uning har bir tuguni chap va o‘ng qism daraxtlari balandliklariga solishtiriladi va unga maksimal qiymat balandlik deb qaraladi.

Masalan quyidagi rasmdagi daraxtning balandligi 2 ga teng.



1-rasm. Binar daraxt balandligi

C dasturlash tilida daraxt balandligini aniqlash dastur kodi keltirib o‘tiladi.

```
 int height(node *tree){
  int h1,h2;
if (treeNULL) return (-1);
```

```
 else {
h1  height(tree->left);h2  height(tree->right);
 if (h1>h2) return (1  h1);
 else return (1  h2);} }
```

Endi binar daraxtni muvozanatlanganligi yoki yo‘qligi tekshirib ko‘riladi:

Yuqorida daraxtning balandligini aniqlashni o‘rgangan edik. Shunga binoan, quyida binar daraxtini muvozanatlanganligini tekshirib chiqaylik. Binar daraxtning muvozanatlanganligini tekshirish uchun uning har ikkala qism daraxti balandliklarini va har bir tugunini hisobga olib, uning o‘rtadagi farqlarini tekshirish muhim hisoblanadi.

Agar bunda har bir qism daraxti balandliklari va tugun farqi **0** yoki **1** ga teng bo‘lsa, u holda bu muvozanatlangan daraxt deb ham hisobga olinadi. Quyida binar daraxtining muvozanatlanganlikka tekshirishning rekursiv funksiyasini qo‘llovchi dastur kodini C dasturlash tilida keltirib o‘tildi.

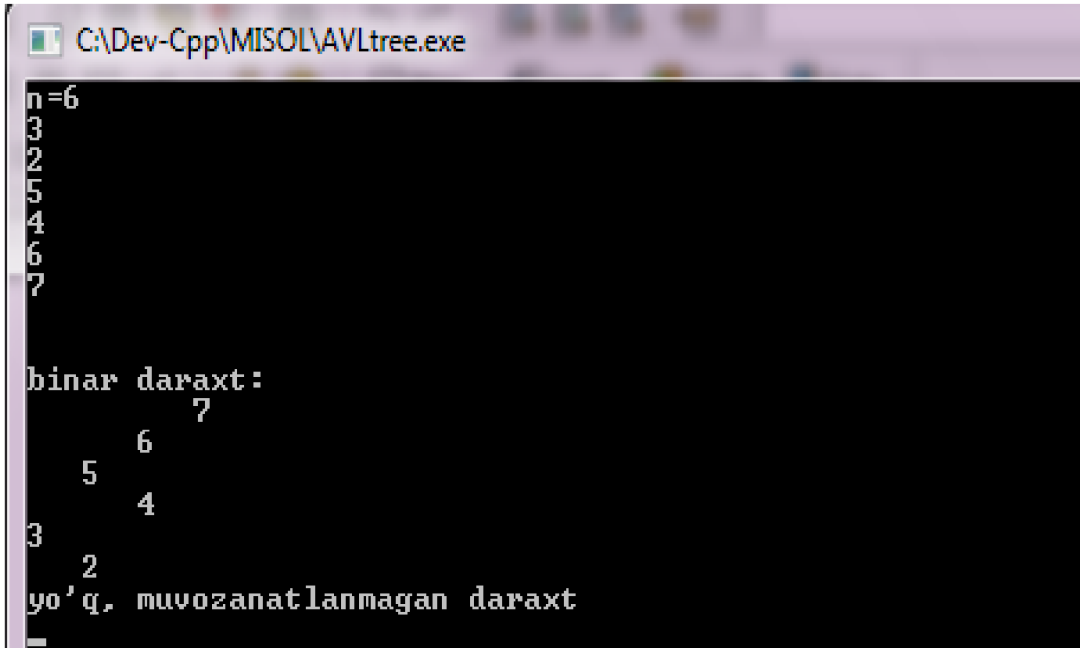
C dasturlsh tilida dastur kodi

```
#include <conio.h>
 #include <iostream>
using namespace std;
 class node{
 public: int info;
node *left;
 node *right; };
  int k0,Flag1;
int height(node *tree){
int h1,h2;
  if (treeNULL) return (-1);
else {
h1  height(tree->left);
  h2  height(tree->right);
```

```
  if (h1>h2) return (1  h1);
else return (1  h2); } }
void vizual(node *tree,int l)
{ int i;
if(tree!NULL) {
  vizual(tree->right,l1);
for (i1; i<l; i) cout<<" ";
 cout<<tree->info<<endl;
 vizual(tree->left,l1);
 }
 }
int AVLtree (node *tree)
{
  int t;
if (tree!NULL)
{
t  height (tree->left) - height (tree->right);
  if ((t<-1) || (t>1)) { Flag  0; return Flag; }
AVLtree (tree->left);  AVLtree (tree->right);
  }
  }
int GetFlag(){return Flag;
}
int main()
{
int n,key,s;  node *treeNULL,*nextNULL;
 cout<<"n"; cin>>n; int arr[n];
  for(int i0; i<n; i)
{
```

```
node *pnew node;
node *lastnew node;
  cin>>s;
  p->infos;
  p->leftNULL; p->rightNULL;
if(i0){treep; nexttree; continue;
 }
nexttree;
 wxile(1)
{
lastnext;
  if(p->info<next->info)nextnext->left;
else nextnext->right;
  if(nextNULL)break;
}
if(p->info<last->info)last->leftp;
  else last->rightp;
}
cout<<endl;
cout<<"\nbinar daraxt:\n";
  vizual(tree,0);
  AVLtree(tree);
if(GetFlag()) cout<<"ha, muvozanatlangan daraxt"; else cout<<"yo‘q,
muvozanatlanmagan daraxt"; cout<<endl;
getch();
}
```

Binar daraxtining muvozanatlanganlikka tekshirishning rekursiv funksiyasini qo‘llovchi dasturning natijasi:



2-rasm. Binar daraxtining muvozanatlanganlikka tekshirish

AVL daraxti va uni misollar bilan yoritish

Binar daraxtini ko‘rikdan o‘tkazayotganda yuqorida har bir tugunni o‘ngida va chapida turgan tugunlarni so‘z bilan ifodalab aytib o‘tildi. Lekin, bu usulda ishlash uchun bir qancha noqulaylikni keltirib chiqaradi. Daraxtni vizual ko‘rinishda ifodalash uning juda ajoyib tarzda tushunishni ifodalaydi. Daraxtni vizuallashtirishning ikki xil ko‘rinishda ifodalash mumkin. Ulardan biri grafik ko‘rinishi bo‘lsa, ikkinchis esa uning konsol oynasida ifodalanishidir.

Endi bu usullardan foydalanib, konsol oynasida daraxtni vizuallashtirishni qarab chiqaylik. Bu usulda ishtirok etadigan sonlar daraxt shaklida joylashtiriladi.

Yuqorida aytilganidek quyida C dasturlash tili muhitida dastur kodi keltirilgan.

```
void vizual(node *tree,int l)
{ int i;
if(tree!NULL) {
```

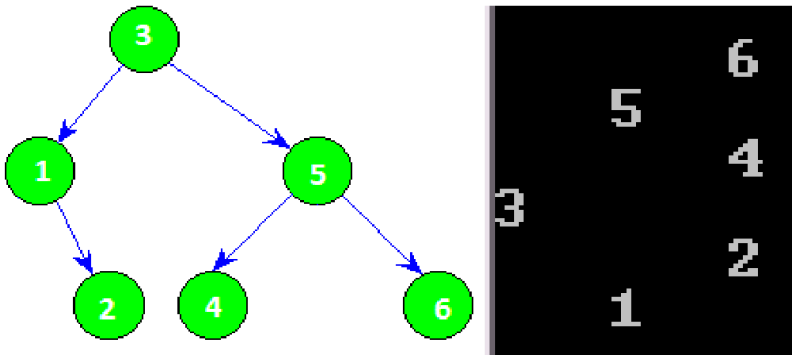
vizual(tree->right,l1);

for (i1; i<l; i) cout<<" ";

cout<<tree->info<<endl;

vizual(tree->left,l1); } }

Quyida dastur kodi a-rasmdagi daraxtni yuqoridagi aytilganidek foydalangan holda konsol ekranida b-rasm ko‘rinishda ifodalab beradi.



a) b)

3-rasm. a - binar daraxt; b - binar daraxtning ekranda namoyon bo‘lishi

Masalan: quyida yuqoridgilarni inobatga olgan holda berilgan binar daraxtning balandligini aniqlab va muvozanatlangan daraxt ekanligini tekshirish kodini yozib chiqaylik.

C dasturlsh tilida dastur kodi keltirilgan:

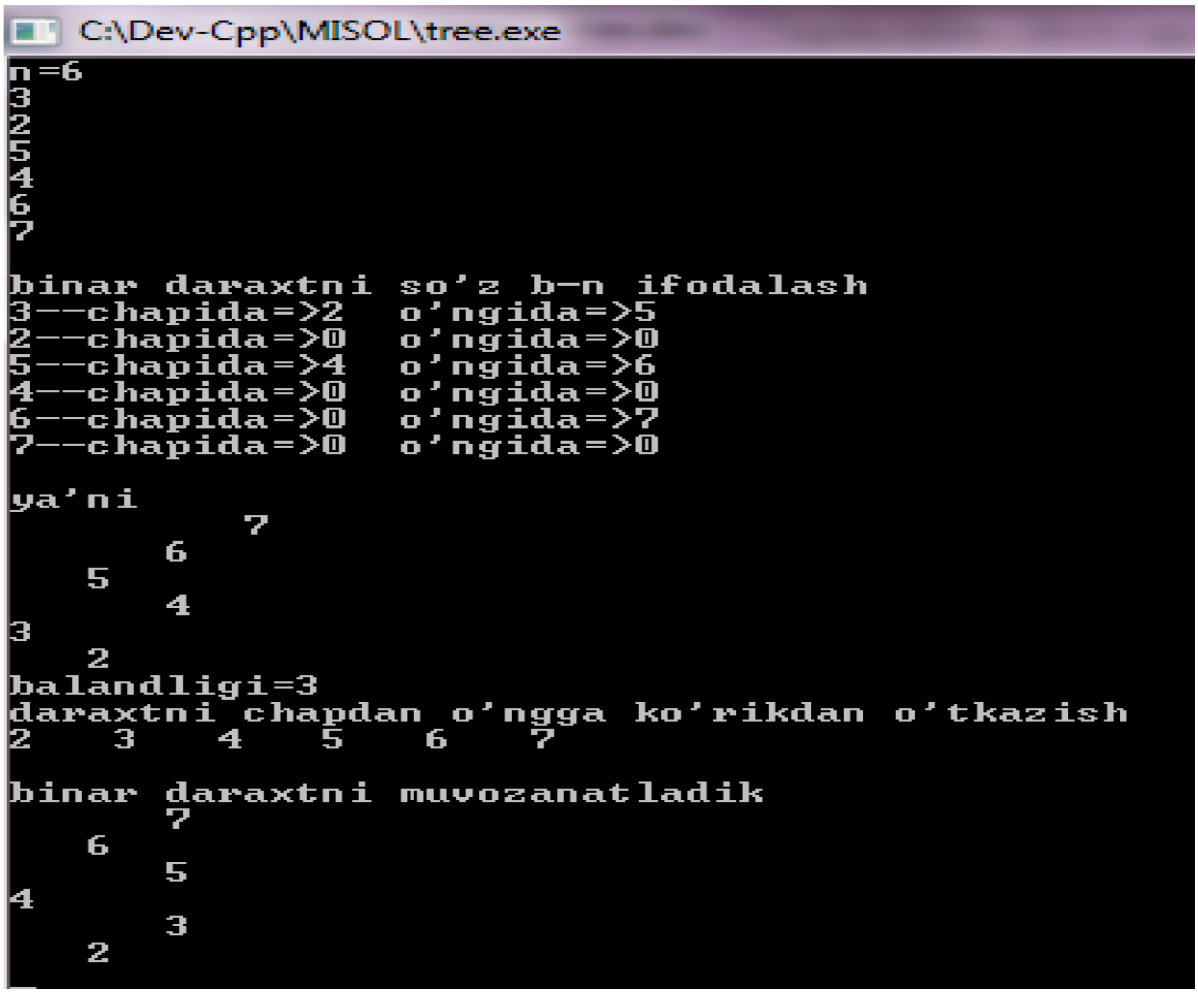
```
#include <conio.h>
 #include <iostream>
 using namespace std;
class node{
public: int info;
 node *left;
 node *right; };
 int k0;
int intrave(node *tree){
 if (tree!NULL){int aNULL, bNULL;
```

```
  if (tree->left!NULL){ atree->left->info; }
 if (tree->right!NULL){ btree->right->info; }
cout<<tree->info<<"--chapida>"<<a<<"
o‘ngida>"<<b<<"\n";  intrave(tree->left);
 intrave(tree->right); }
return 0; }
 int height(node *tree){
  int h1,h2;
if (treeNULL) return (-1);
 else {
h1  height(tree->left);
 h2  height(tree->right);
  if (h1>h2) return (1  h1);
else return (1  h2); } }
 int create_arr(node *tree,int *arr){
if(!tree) return 0;
  else{ create_arr(tree->left,arr);
  arr[k]tree->info;
 create_arr(tree->right,arr); } }
 node *new_tree(int *arr, int start, int end)
{ if(start>end) return NULL;
else {
int mid(startend)/2;
  node *treenew node;
 tree->infoarr[mid];
 tree->leftnew_tree(arr,start,mid-1);
tree->rightnew_tree(arr,mid1,end);
return tree; } }
  void vizual(node *tree,int l)
```

```
{ int i;
if(tree!NULL) {
 vizual(tree->right,l1);
for (i1; i<l; i) cout<<" ";
 cout<<tree->info<<endl;
vizual(tree->left,l1); } }
int main()
{ int n,key,s;
node *treeNULL,*nextNULL;
cout<<"n"; cin>>n; int arr[n];
 for(int i0; i<n; i){
  node *pnew node;
node *lastnew node;
 cin>>s;
  p->infos;
p->leftNULL;
 p->rightNULL;
if(i0){treep; nexttree; continue; }
 nexttree;
  wxile(1)
  { lastnext;
  if(p->info<next->info)nextnext->left;
 else nextnext->right;
 if(nextNULL)break; }
if(p->info<last->info)last->leftp;
 else last->rightp;}
cout<<endl;
intrave(tree);
 cout<<"\nya’ni\n";
```

```
vizual(tree,0);
int hheight(tree);
 cout<<"balandligi"<<h<<endl;
create_arr(tree,arr);
 for(int i0;i<k;i)cout<<arr[i]<<" ";cout<<endl<<endl;
treenew_tree(arr,0,k-1);
vizual(tree,0);
getch(); }
```

Yuqoridagi dasturning natijasi



4-rasm. Binar daraxtining muvozanatlanganlikka tekshirish natijasi

Mavzuga oid test savollari

1.56,34,60,23,40,65 sonlaridan hosil bo‘lgan binar daraxt muvozanatlanganmi yoki yo‘qmi?

a) Xa, muvozanatlangan

b) Yo‘q, muvozanatlanmagan

c) Har ikkalasi ham bo‘lishi mumkin

d) O‘rtacha muvozanatlangan

2. 10,7, 12, 2, 5, 3, 11, 14 sonlaridan hosil qilingan binar daraxtda nechta oraliq tugun mavjud?

a) 4 ta

b) 2 ta

c) 5 ta

d) 8 ta

3. 10,7, 12, 2, 5, 3, 11, 14 sonlaridan hosil qilingan binar daraxtda nechta barg mavjud?

a) 3 ta

b) 2 ta

c) 5 ta

4. 10,7, 12, 2, 5, 3, 11, 14 sonlaridan hosil qilingan binar daraxt balandligi nechaga teng?

a) 5 ta

b) 3 ta

c) 4 ta

Nazorat savollari

1. Muvozanatlangan daraxt deb nimaga aytiladi?
2. AVL algoritmlar nima vazifani bajaradi?
3. Vizuallash nima uchun kerak?
4. Daraxt balandligi qanday aniqlanadi?

Xulosa

Binar daraxtning muvozanatlash algoritmlari quyidagicha koʻrinishda boʻladi: Bunda birinchi boʻlib, binar daraxt yaratib olinadi. Soʻngra binar daraxtni chapdan oʻngga koʻrikdan oʻtkaziladi va tugunlarning info maydonlaridan a[..] massiv hosil qilinadi.

Tabiiyki, massiv oʻsish boʻyicha tartiblangan boʻlishi lozim. Bunda muvozanatlangan daraxtning tugunlarini belgilash uchun massivni koʻriladigan oraligʻi start 0 va endn-1 koʻrinishlarida belgilab olinadi.

Shundan soʻng, massivning koʻrilayotgan oraligʻi oʻrtasida joylashgan elementni, ya’ni mid(startend)/2 va a[mid] ni muvozanatlangan daraxtning tuguni deb hisoblab olinadi.

Binar daraxtni koʻrikdan oʻtkazayotganda har bir tugunni oʻngida va chapida turgan tugunlarini ifodalandi. Lekin, bu usulda ishlash bir qancha noqulaylikni keltirib chiqaradi.

Daraxtni vizual koʻrinishda ifodalash uning juda ajoyib tarzda tushunishni ifodalaydi. Daraxtni vizuallashtirishning ikki xil koʻrinishda ifodalash mumkin. Ulardan biri grafik koʻrinishi boʻlsa, ikkinchis esa uning konsol oynasida ifodalanishidir.

1.4. Heap tree koʻrinishidagi binar daraxtlar. Heap tree tuzilmasi tavsifi. Heap tree ustida amal bajarish algoritmlari. Heap treeni tashkil etish usullari va samaradorligi

Reja

1. Heap tree tuzilmasining tavsifi va uni chizmada tasvirlash.
2. Heap tree ustida amallar bajarish algoritmlari.
3. Heap treeni tashkil etish usullari va uning samaradorligi.

Tayanch iboralar: Heap tree daraxt, tizim, samaradorlik, binar daraxt, klassifikatsiya, ideal, qism daraxt, ikkilik daraxti.

Heap tree tuzilmasini tavsifi va uni chizmada tasvirlash

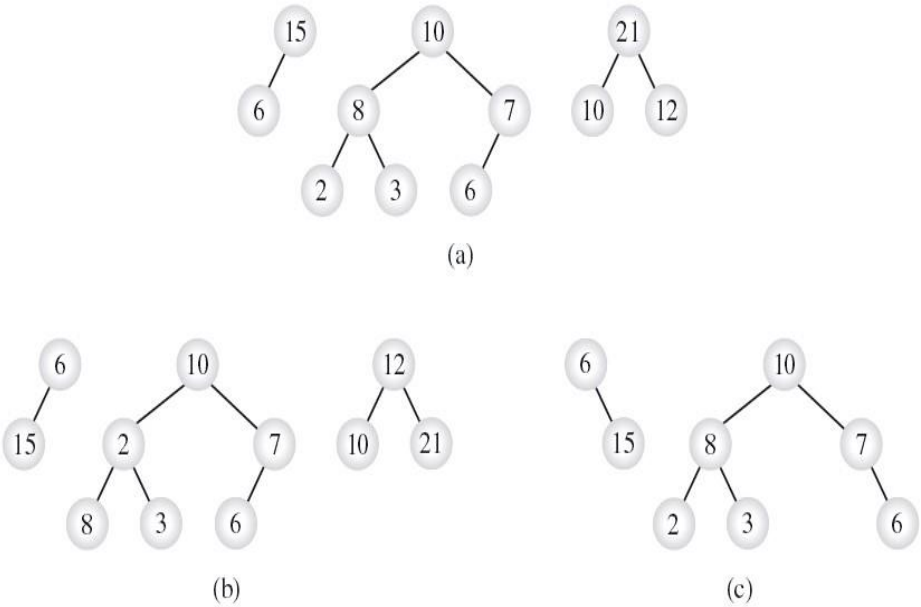
“Heap tree” soʻzi ingliz tilidan olingan boʻlib, u ikkilik uyurma daraxti degan ma’noni bildiradi va binar daraxtning bir turi boʻlib hisoblanadi hamda binar daraxtdan ikkita ajoyib xususiyati bilan ajralib turadi.

- Har bir tugun qiymati uning oʻgʻil tugunlari qiymatidan katta yoki teng hamda kichik qolversa teng boʻlishi ham mumkin;
- Daraxt ideal muvozanatlangan boʻladi yoki daraxt barg tugunlari chapdan oʻngga qarab toʻldirila boshlaydi.

Agar har bir tugun oʻgʻil tugunlardan katta yoki teng boʻlsa, u holda daraxtiga **max heap** toʻgʻri keladi**,** aks holda, ya’ni ota tugun farzandlardan kichik yoki teng boʻlsa, **min heap** deb nomlanadi.

Bu degani, max heap da maksimal element daraxt ildizida joylashadi, min heapda esa bu holat daraxtning ildizida minimal element sifatida oʻrnashadi.

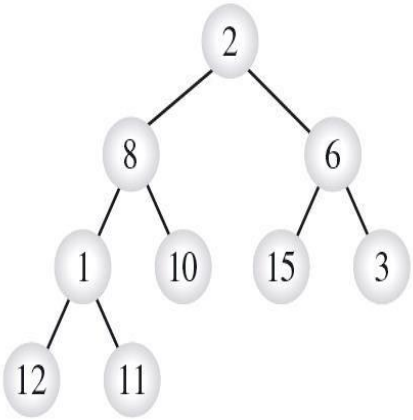
1-rasmda Heap tree (a) va heap tree boʻlmagan daraxtlar koʻrinishi tasvirlangan. Bu yerda rasmdagi a) heap tree va b) hamda c) lar esa heap tree boʻlib hisoblanmaydi. Chunki, bunda b va c rasmda heap treening birinchi va ikkinchi xususiyatlari qaysi tomonlalama mos ravishda buzilgan hisoblanadi.



1-rasm. Heap tree (a) va heap tree bo‘lmagan daraxtlar ko‘rinishi

Qizig‘i shundaki, heap tree ham massiv yordamida tuzilishi mumkin. Misol uchun, data[] {2 8 6 1 10 15 3 12 11} massiv berilgan bo‘lsin. Bu yerda yuqoridan pastga va chapdan o‘ngga elementlarni joylab yangi daraxt (heap tree bo‘lmagan)ni keltirib chiqaradi.

Bu keltirib chiqargan daraxtlarga bosqichlar soni O(lgn) ga teng deb olinadi.



2- rasm. Massivdan daraxt hosil qilish jarayoni

Bu daraxtni heap tree ko‘rinishida qayta tashkil qilish uchun uning uzunligini n ga teng deb olib, **heap** massivini quyidagi shartlarga asoslangan holda tashkil etiladi:

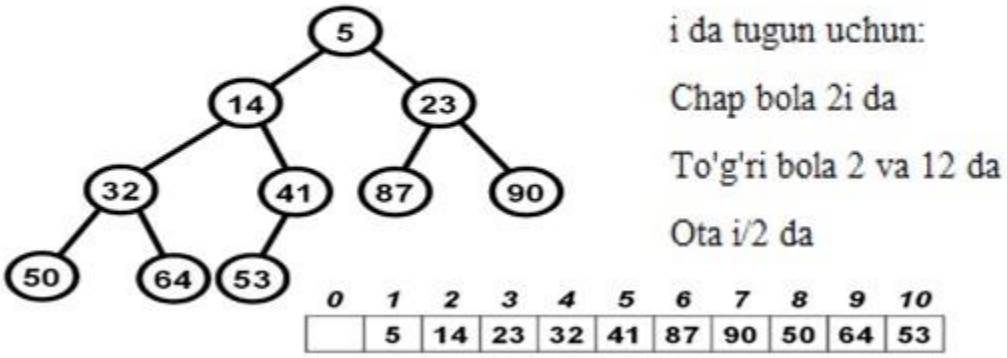
heap[i] ≥ heap[2·i], for $0 \le i < \frac{n}{2}$

va heap[i] ≥ heap[2·i 1], for $0 \le i < \frac{n-1}{2}$

Bu yerda heap tree elementlari tartiblanmagan elementlar hisoblanadi.

Boshqacha qilib aytganda, bunda sonlar ketma-ketligidan heap treeni qurish uchun quyidagi ishlarni amalga oshirish mumkin.

- Birinchi tugun elementni daraxt ildizi deb qabul qilib olinadi.
- Har qanday i-element uchun quyidagilar o‘rinli deb qaraladi:
- Bu yerda quyidgi jarayonni yodda saqlash kerak, ya’ni:
- Uning chap o‘g‘il tuguni 2*i indeksda,
- Uning o‘ng o‘g‘il tuguni esa 2*i1 indeksda
- Uning ota tuguni i/2 indeksga ega bo‘ladi.

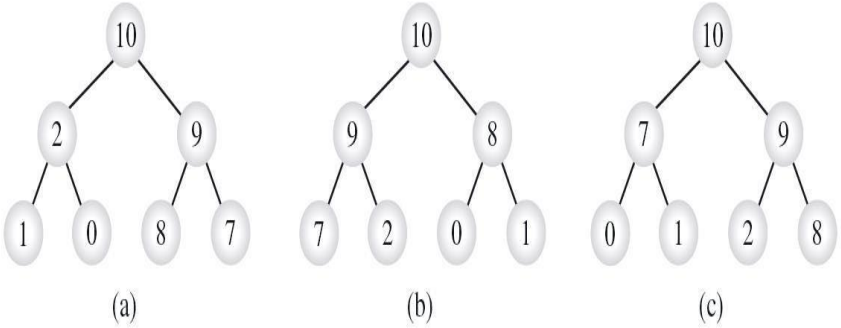


3-rasm. Heap tree ning ko‘rinishi

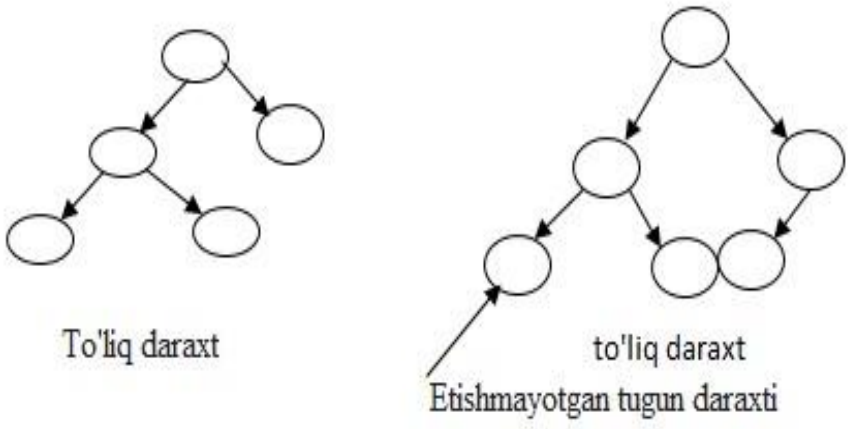
Birinchi elementni ildiz deb belgilab olingandan so‘ng, qolgan elementlarni daraxtning chapdan o‘ngga qarab to‘ldirib chiqiladi. Bu yerda har bir ildiz uchun faqat ikkita o‘g‘il tugun chiqishi lozim bo‘ldi.

Agarda shu usulda elementlarni joylashtirib chiqadigan bo‘lsak, u holda har bir a[i]-o‘rinda turgan ota tugunning chap tomoniga a[2*i]-element hamda o‘ng tomoniga esa mana bu ko‘rinishdagi a[2*i1]elementlar joylashtiriladi.

Bundan ko‘rinadiki, quyida bir xil sonlardan turlicha heap tree hosil qilish mumkin.

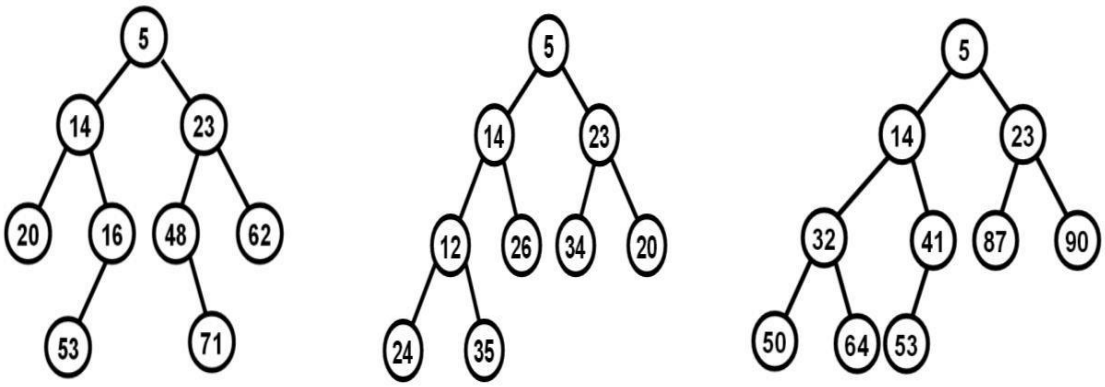


4-rasm. Bir xil sonlardan tashkil topgan heap tree tuzilmasining ko‘rinishi



5-rasm. Heap treeni to‘g‘ri va noto‘g‘ri tashkil etilganligining ko‘rinishi

Quyida yana heap treega mos bo‘lgan rasmlarni ko‘rib chiqaylik. Bu rasmlardan qaysilari Heap tree ekanligiga e’tibor qaratiladi.



6-rasm. Heap treeni ro‘rinishlari

Yuqoridagi rasmdan heap tree ko‘rinishlari bilan tanishib chiqildi.

Bundan ko‘rinadiki, bu yerda heap tree protepli navbatni ifodalashda juda ham mos kelar ekan. Heap treeda eng kerakli element tuzilma uning eng yuqorisida joylashadi. Agar u o‘chirilsa, elementlar qayta joylashtirilishi lozim bo‘ladi. Bu tuzilma graflarda eng qisqa masofani aniqlash masalasini yechishda diskret algoritmlarini samaradorligini oshirishda protepli navbatlardan foydalanilganda eng qulay usul bo‘lib hisoblanadi.

Bundan tashqari heap tree - samaradorligi O(n log n) bo‘lgan piramidani saralash algoritmida ham juda yaxshi natija beradi.

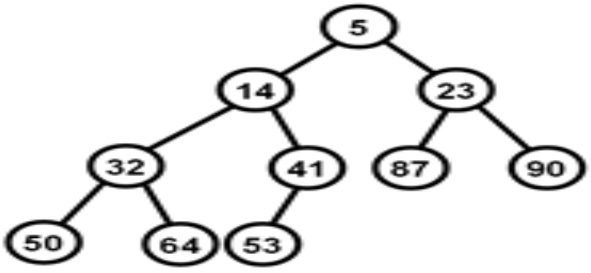
Heap tree ustida amallar bajarish algoritmlari

Heap tree ustida amal bajarish algoritmlarini ko‘rib chiqaylik. Heap tree ustida quyidagi amallarni bajarish mumkin bo‘ladi:

- Heap tree da element qo‘shish;
- Heap tree da element o‘chirish.

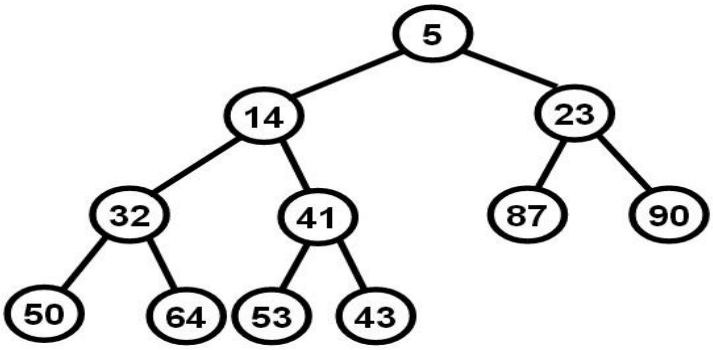
Min-heapga yangi element kiritish algoritmlari:

- Yangi elementni massivning navbatdagi indekslariga o‘rnatish;
- Yangi elementni ota tugun bilan solishtirilib, agar yangi element otasidan kichik bo‘lsa, ularning o‘rni almashtiriladi;
- Bu jarayon bir necha barobar takrorlanadi, toki u o‘z aksini topguncha:
- Yoki yangi elementning otasi kichik yoki teng bo‘lguncha;
- Yoki yangi element ildizga kelguncha, ya’ni massivda 0 indeksga kelguncha. Yuqoridagilardan foydalanib, quyidagi misol ko‘rib chiqaylik. Bu chizmada quyidagi o‘zgartirish kiritiladi.



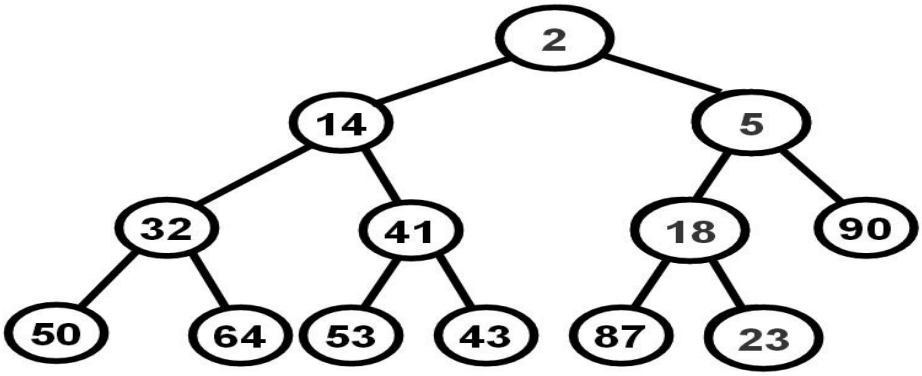
7-rasm. Min-heapga yangi element kiritish algoritmi

Masalan: Min-heap ning elementlari ichida yangi 43 soni kiritiladi. Natijada quyidagi koʻrinish hosil boʻladi.



8-rasm. Min-heapga yangi element kiritish algoritmi

Shu bilan birga min-heapga 2 ga ham 18 soni kiritiladi.



9-rasm. Min-heapga yangi element kiritish algoritmi

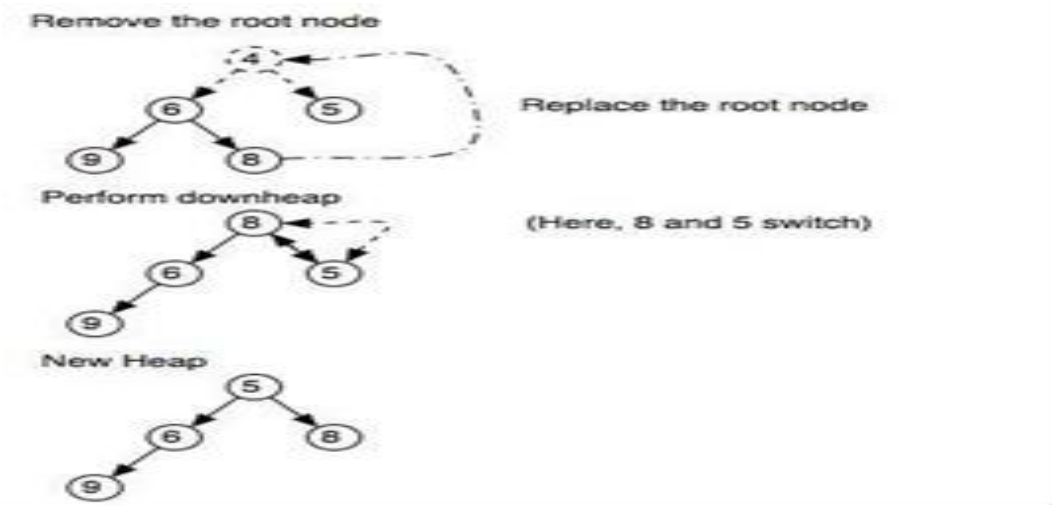
Min-heap treedan elementni oʻchirish algoritmi:

- Oʻchiriladigan element oʻrniga daraxtdagi eng quyi darajada turgan soʻnggi oʻngdagi, ya'ni oxirgi element joylashtiriladi.

- Oʻrni oʻzgargan shu element ikkita oʻgʻil tugunlari bilan solishtiriladi va agar ulardan katta boʻsa, kichik oʻgʻil tugun 1 bilan oʻrin almashtiriladi.

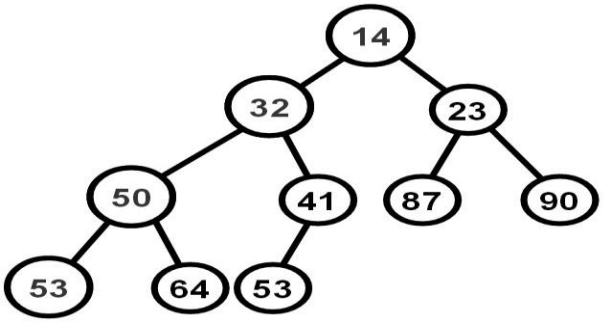
- Oʻrinlashtirishda qatnashgan element ta'sir qiladigan qism daraxtlari tekshirib koʻriladi, buning uchun oxirgi ikkita amal qayta-qayta bajariladi.

-



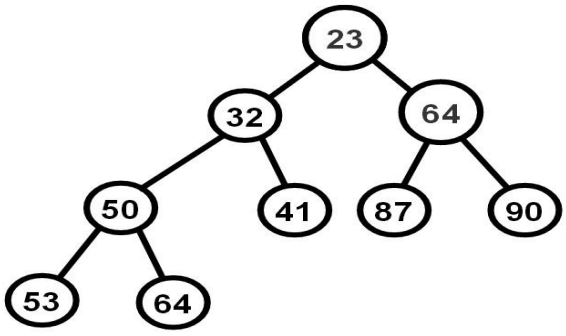
10-rasm. Min-heap treedan elementni o‘chirish algoritmi

Yuqoridagilarni inobatga olgan holda, misol uchun quyidagi rasmdan heap treedan 5 o‘chiriladi. Natijada quyidagi heap tree ko‘rinishi hosil bo‘ladi.



11-rasm. Min-heap treedan elementni o‘chirish algoritmi

Agar bundan 14 o‘chirilsa, pastda yana ko‘rinishi jihatdan boshqacharoq heap tree ko‘rinishga keladi:



12-rasm. Heap tree tuzilmasi bilan ishlashning yuqori samaradorligi

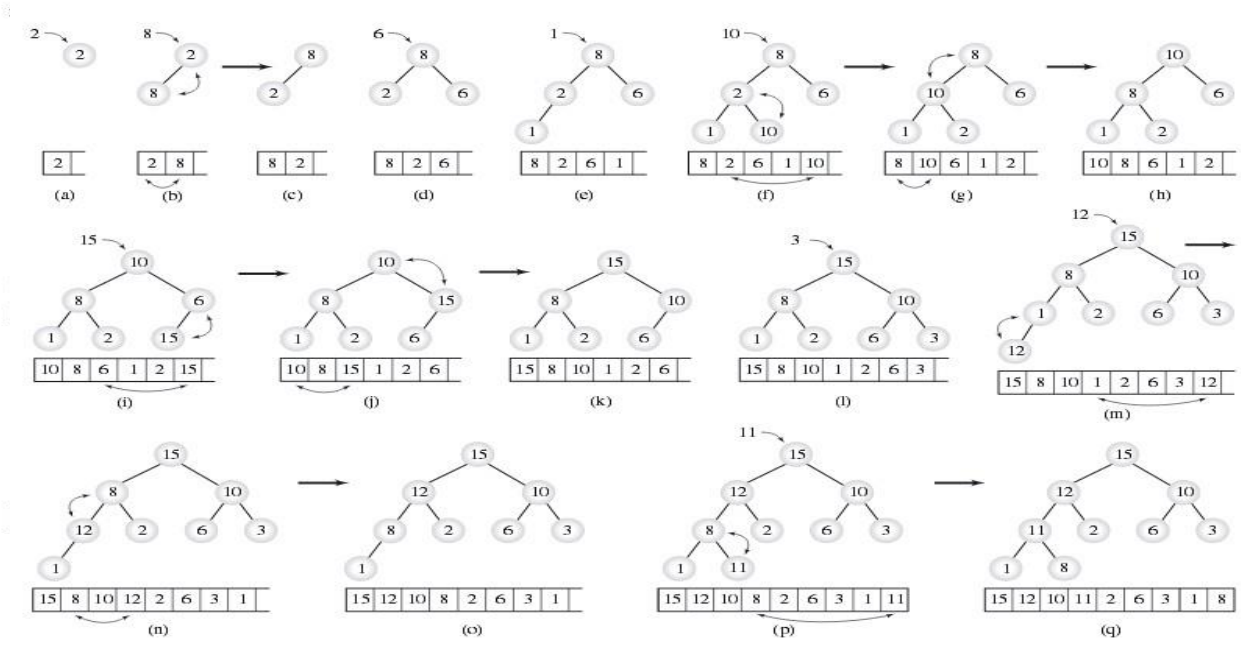
Agar tuzilmada N ta element mavjud bo‘lsa, undagi bosqichlar soni $\log_2(N1)$ ta teng bo‘ladi.

- Yangi element kiritishda 1 ta bosqichda 1 ta elementdan ko‘p bo‘lmagan o‘rinlashtirishni bajarilishi sababli kiritish samaradorligi esa O(lon(n)) ga teng bo‘ladi.

- Element o‘chirishda ham har bir bosqichda faqat 1 ta o‘rinlashtirish bajarilishi mumkinligi sababli o‘chirish samaradorligi O(log(n)) ga teng bo‘ladi.

Heap treeni tashkil etish usullari va uning samaradorligi

Aslida Heap treeni dasturida massiv ko‘rinishida ifodalash mumkin, ya’ni barcha heap tuzilmalarni massiv ko‘rinishida ifodalash qulay bo‘ladi, lekin barcha massivlar heap tree ham bo‘lmaydi. Berilgan massiv elementlarini shunday joylash kerakki, natijada heap tree namoyon bo‘lsin. Buning bir necha usullari mavjuddir. Eng soddasi bo‘sh heap treega ketma-ket elementlarni joylash bilan amalga oshiriladi. Bu “tepadan-pastga” usuli ya’ni elementlar heap treega yuqorida keltirilgan yangi element qo‘shish algoritmi bilan hisoblanadi. Aslida bu usul Jogn Williams tomonidan taklif etilgan. Quyida rasmda “tepadan-pastga” o‘tish algoritmi ifodalangan va dasturlar keltirib o‘tiladi.

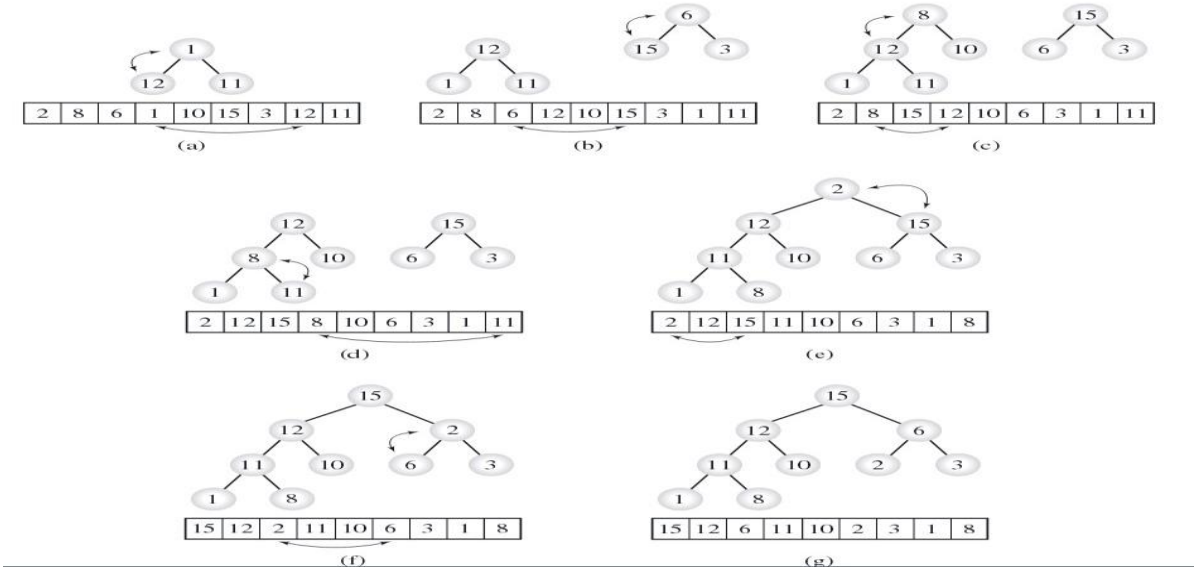


13-rasm. Heap treeni “tepadan-pastga” deb nomlangn usuli bilan tashkil etish

Bu algoritm samaradorligini eng yomon holatlarda tekshiradigan bo‘lsak, unga kiritilgan har bir element ildizgacha yuqoriga harakat qilishi shart bo‘ladi. Bunda k ta elementdan iborat bo‘lgan heapda yangi kiritilgan element yuqoriga, ya’ni harakat qilishi uchun [lg k] ta o‘rin almashtirishlar amalga oshirilishi lozim bo‘ladi. Agar bunda n ta yangi element kiritilsa, eng yomon holatda algoritm bajarilishi quyidagicha o‘rinlashtirishlarni oladi, solishtirishlar ham xuddi shunday tartibda bo‘ladi.

$$\sum_{k=1}^{n} \lfloor \lg k \rfloor \le \sum_{k=1}^{n} \lg k = \lg 1 + \cdots + \lg n = \lg(1 \cdot 2 \cdot \cdots \cdot n) = \lg(n!) = O(n \lg n)$$

Bunday holatni yana Robert Floyd tomonidan ham taklif etilgan boshqa bir algoritmda heap tree “pastdan-yuqoriga” usuli yordamida amalga oshirilganini ko‘rish mumkin. Bu holatda kichik heap qismlar yaratiladi va davriy ravishda kattaroq heap tree larga birlashtiriladi.



14-rasm. Array [2 8 6 1 10 15 3 12 11] massivni “pastdan-yuqoriga” usuli bilan heap treega aylantirish

Bunda elementlarni tekshirishda data[n/2-1] barg tuguni bo‘lmagan elementdan boshlanadi. Agar u o‘g‘il tugunlardan birontasidan kichik bo‘lsa, u holda katta qiymatli o‘g‘il element bilan almashtiriladi va jarayon yuqoridagi

rasmdagi kabi davom ettirilib boriladi. Bu usulda qachonki yangi element tahlil qilinayotganda uning qism daraxti allaqachon heap tree bo‘ladi.

Shunday qilib, aslida heap tree pastdan yuqoriga qarab shakllantiriladi. Heap treeni bunday tashkil qilishda, moveDown() funksiyasi (n1)/2 marta chaqiriladi. Unda har bir barg bo‘lmagan tugun uchun bo‘ladi. Eng yomon holatda moveDown() elementni (n1)/4 ta elementdan iborat bo‘lgan eng quyi bosqichga ko‘chiradi, bunda barg tugunlar bosqichiga yetib kelguncha har bir bosqichda (n1)/4 ta o‘rinlashtirishlarni amalga oshiradi.

Qisqacha qilib aytganda, bu usul samaradorligi O(n) ga tengdir. Shu sababli eng o‘g‘ir holatda Williamning usuli Floydning usulidan samaraliroq bo‘lib hisoblanadi. O‘rta holatda esa ikkala algoritm ham deyarli bir xil ko‘rinishda ishlaydi.

Mavzuga oid test savollari

1. Minimal balandlikka ega bo‘lgan daraxt bu?

a) HEAP TREE
b) BINARY TREE
c) Red Black Tree
d) 2-3 TREE

B
2 / \
A C **Binar daraxt uchun to‘g‘ri (yuqoridan pastga) ko‘ruv amalining natijasini ko‘rsating?**

a) BAC
b) ACB
c) ABC

3. HEAP TREE bu qanday daraxtning elementlari hisoblanadi?

a) Minimal balandlikka ega daraxt
b) Maximal balandlikka ega daraxt

c) Binar daraxtining maksimal balandligi

4. Agar uning chiqish darajasi ikkidan oshmasa, daraxt qanday nomlanadi?

a) Binar

b) Ternar

c) Tetradli

Nazorat savollari

1. Heap tree tuzilmasi nima?

2. Heap tree ustida amal bajarish algoritmlari qanday kechadi?

3. Heap treeni tashkil etishning “pastdan-tepaga” usuli va samaradorligi qanday?

4. Heap treeni tashkil etishning “yuqoridan-pastga” usuli va samaradorligi qanday?

Xulosa

“Heap tree” so‘zi ingliz tilidan olingan bo‘lib, u ikkilik uyurma daraxti degan ma’noni bildiradi va binar daraxtning bir turi bo‘lib hisoblanadi hamda binar daraxtdan ikkita ajoyib xususiyati bilan ajralib turadi. Agar har bir tugun o‘g‘il tugunlardan katta yoki teng bo‘lsa, u holda daraxtiga **max heap** to‘g‘ri keladi, aks holda, ya’ni ota tugun farzandlardan kichik yoki teng bo‘lsa, **min heap** deb nomlanadi. Bu degani, max heap da maksimal element daraxt ildizida joylashadi, min heapda esa bu holat daraxtning ildizida minimal element sifatida o‘rnashadi.

Williamning usuli Floydning usulidan samaraliroq bo‘lib hisoblanadi. O‘rta holatda esa ikkala algoritm ham deyarli bir xil ko‘rinishda ishlaydi.

5-BOB. GRAFLAR BILAN ISHLASH ALGORITMLARI

1-§. Graflar bilan ishlash algoritmlari

1.1. Graflar bilan ishlash algoritmlari. Graflarni tasvirlash usullari: qo‘shma matritsa va munosabat matritsasi

Reja:

1. Graflar nazariyasining asosiy tushunchalari va ularni tasvirlash usullari.
2. Graflarni ifodalash usullari.

Tayanch iboralar: graf, multigraf, psevdograf, yo‘naltirilgan graf, to‘liq graf, to‘yingan graf, siyrak graf, tugun, juftlik, qirra, chekli, cheksiz, yo‘l, ilmoq.

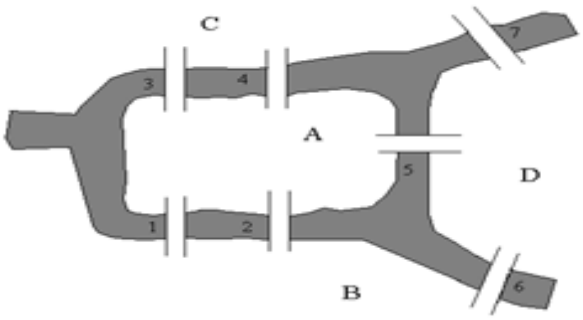
Graflar nazariyasining asosiy tushunchalari va ularni tasvirlash usullari

Matematik nazariyada va informatikada graf – bu tugunlar, ya’ni uchlardan tashkil topgan bo‘lib, u bo‘sh bo‘lmagan to‘plam va tugunlarni birlashtiruvchi yoylarning majmuidir.

Graf – bu murakkab chiziqsiz ko‘p bog‘lamli dinamik tuzilma bo‘lib, murakkab ob’ektlarning xususiyatlari va munosabatlarini o‘zida aks ettiradi. Ob’ektlar graf uzellari yoki tugun ko‘rinishida va uning munosabatlari yoy yoki yo‘naltirilgan qirralar kabi ifodalangan bo‘ladi.

«Graf» atamasini birinchi marotaba 1936-yil Vengriya matematigi Denni Kyonig degan olim kiritgan. Lekin, graflar nazariyasi bo‘yicha buni birinchi ish sifatida Leonard Eylerga tegishli bo‘lgan.

Ba’zi manbalarda bu usulni 1736-yilda bajarilgan edi, - deb yozilgan. Graf tushunchasini XVIII asrda mashhur shvetsariyalik matematik va mexanik hamda fizik Leonard Eyler (1707-1783 yy) Kyonigsberg ko‘prigi haqidagi masalani yechish uchun bu tushunchadan foydalanadi.



1-rasm. Eski Kyonigsberg sharhi sxemasi

Aslida Graflar nazariyasi – bu diskret matematika fanining bir bo‘limi bo‘lib, unda masalalar yechimlari chizmalar shaklida qidiriladi. Keyingi vaqtlarda turli xil diskret xususiyatlarga ega bo‘lgan hisoblash qurilmalarini loyihalashda graflarning ahamiyati yanada kengroq bo‘ldi.

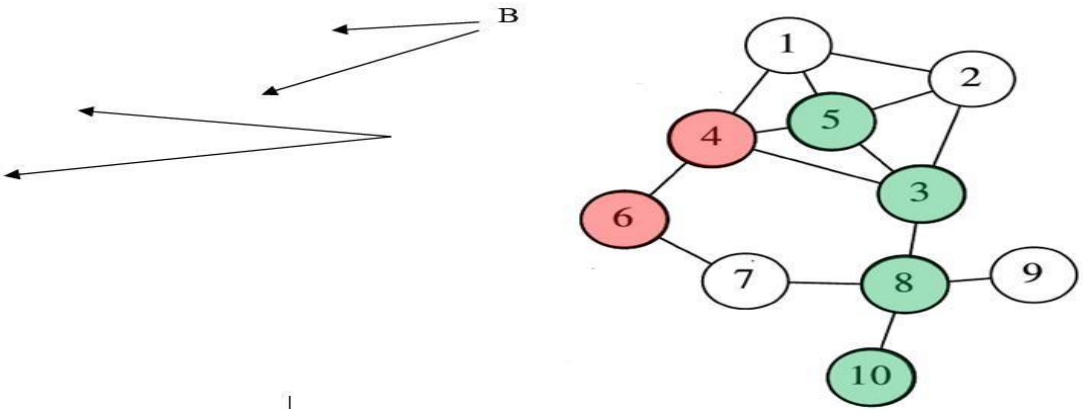
Grafni tasvirlashda bu (V, E) sonlar juftligiga graf deyiladi, bu yerda V – ixtiyoriy bo‘sh bo‘lmagan to‘plam, E esa $V^{(2)}$ning qism to‘plami $(E \subseteq V^{(2)})$, bunda $V^{(2)}$ V to‘plam elementlarining tartiblanmagan juftliklari to‘plamidir.

Grafning qirralari deb, E – to‘plam elementlariga aytiladi.

Grafning uchlari esa V – to‘plam elementlariga qaratiladi.

Agar grafning uchlari (V) chekli bo‘lsa, **chekli graf** deb nomlanadi, aks holda u **cheksiz graf** deb aytiladi.

Yo‘l (path) – bu bironta tugundan boshqa bir tugungacha bo‘lgan yonma-yon joylashtirilgan tugunlar ketma-ketligidan iborat.



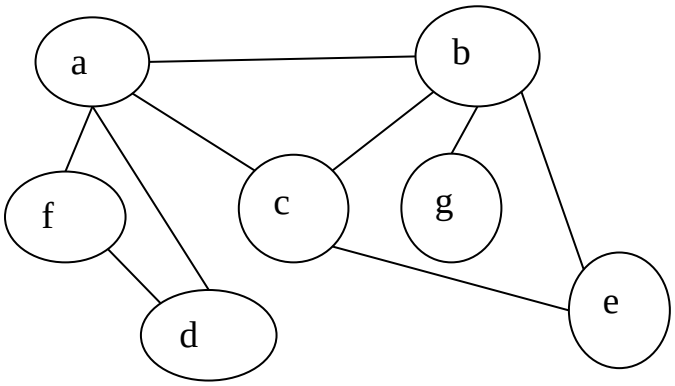
2-rasm. Grafning asosiy alomatlari

Bu yerda grafning uchlarini va qirralari to‘plamini mos ravishda $V(G)$ va $E(G)$ kabi belgilanadi. Sababi $V(G)$ ushbu to‘plamda faqat graf uchlar berilgan bo‘ladi. $E(G)$ to‘plamida esa bu grafning qirallarning qo‘shni uchlar juftligi bilan aniqlanadi.

Misol uchun: V(G) ={a,b,c,d,e, f ,g}

E(G) ={(a,b),(a,c),(a,d),(a, f),(b,c),(b,g),(b,e),(c,e),(d, f)}.

Bu holda grafning grafik ko‘rinishi quyidagicha namoyon bo‘ladi (3-rasm):



3-rasm. Grafga misol

Bunda qirra ikkita uch bilan namoyon bo‘ladi. Bu yerda umumiy uchga ega bo‘lgan ikkita qirra qo‘shni qirra deb hisobga olinadi. Agar bunday holda grafning ikkita uchi qirra bilan tutashtirilgan bo‘lsa, bu uchlar **qo‘shni uchlar** deb nomlanadi. Grafning bir uchdan chiqqan ikki qirrasi esa **qo‘shni qirralar** deb yuritiladi. Agar yasalgan grafda boshi va oxiri bitta tugunda tutashadigan qirra mavjud bo‘lsa, u holda bunday holat **ilmoqli qirra** deyiladi.

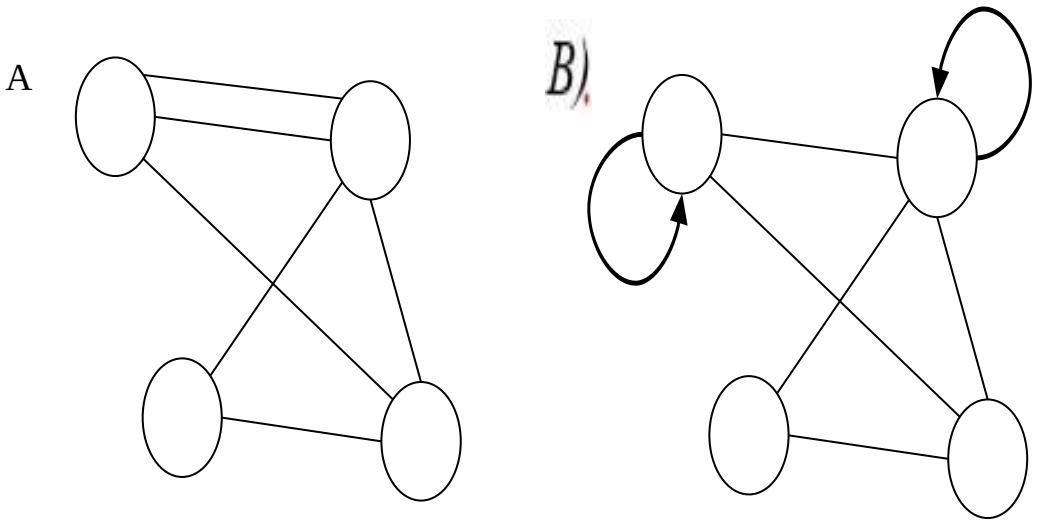
qirra

□□□□□□□1 uch 2 uch

4-rasm. Qirra tushunchasi

Agar grafda takroriy, ya’ni karrali qirralar namoyon bo‘lsa, bunday grafni **multigraf** deb ataladi.

Agar grafda karrali qirralar bilan birga uchni o‘z-o‘zi bilan tutashtiruvchi ilmoqlar ham birga kelsa, u holda bunday graf **psevdograf** deb yuritiladi.



5-rasm:

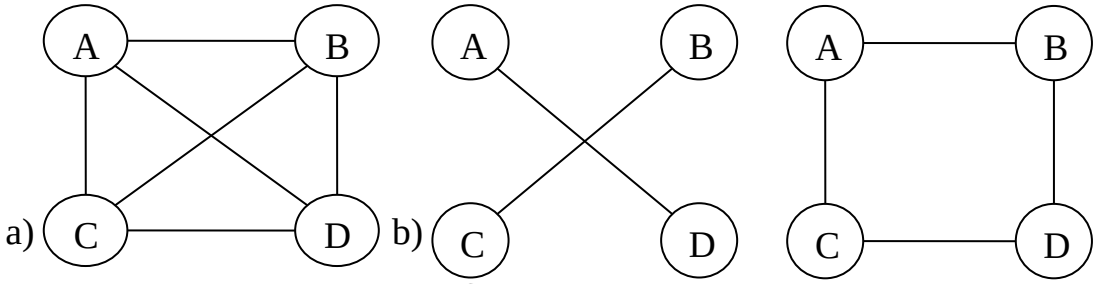
A) multigraf; B) psevdograf

Agar ixtiyoriy tugundan boshqa birorta tugunga murojaat mavjud bo‘lsa, aynan shu murojaat ikki tomonlama ham davom etsa, buday holatdagi grafga **yo‘naltirilmagan graf** deyiladi (5-rasm. A).

Agar graf tugunlari o‘zaro bog‘langan bo‘lib, lekin bu yoylar orqali munosabatlar faqat bir tomonlama bo‘lsa, u holda bunday graflarga **yo‘naltirilgan graflar,** ya’ni oriented graph deb yuritiladi (5-rasm. B).

Istalgan tugunlari qo‘shni bo‘lgan grafga **to‘liq graf** (complete graph) deb yuritiladi, ya’ni bunda barcha tugunlar o‘zaro birlashtirilgan bo‘ladi.

Bundan tashqari grafni to‘ldiruvchisi ham mavjud bo‘lib, u aynan bir tugunlardan va aynan bir qirralardan tashkil topgan bo‘lib, u tayyor grafni to‘liq bo‘lishini **ta’minlovchi graf**ga aytiladi.



6-rasm:

A) to‘liq graf B) graf va uning to‘ldiruvchisi

Eng sosiysi to‘liq yo‘naltirilmagan grafda qirralar soni quyidagi formula (1) orqali hisoblanadi:

$$m = \frac{n(n-1)}{2} \quad (1)$$

Bu yerda n – yoylar, ya’ni tugunlar sonini bildiradi.

Yuqoridagi chizmaga e’tibor qaratiladigan bo‘lsa, D grafning to‘yinganligi deb hisobga olinadi. Ya’ni, u grafning qirralarini tugunlar nisbatiga to‘liqlik munosabat koefitsientini beradi. U quyidagi formula (2) orqali aniqlanadi:

$$D = \frac{2m}{n(n-1)} \quad (2)$$

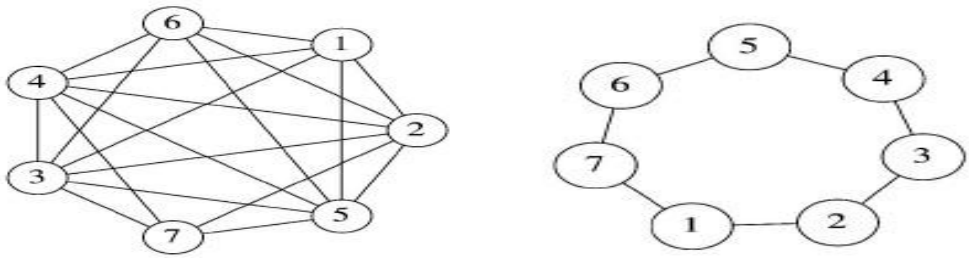
Bu yerda n – grafning tugunlar soni, m – grafning qirralar sonidan iborat.

Graflarning to‘yinganligi koefitsientiga qarab, ikki xil graf ko‘rinishda namoyon qilish mumkin: Ularning biri to‘yingan graf va ikkinchisi siyrak graf deb nomlanadi.

To‘yingan graf deb, qirralar soni eng yuqori bo‘lgan grafga aytiladi.

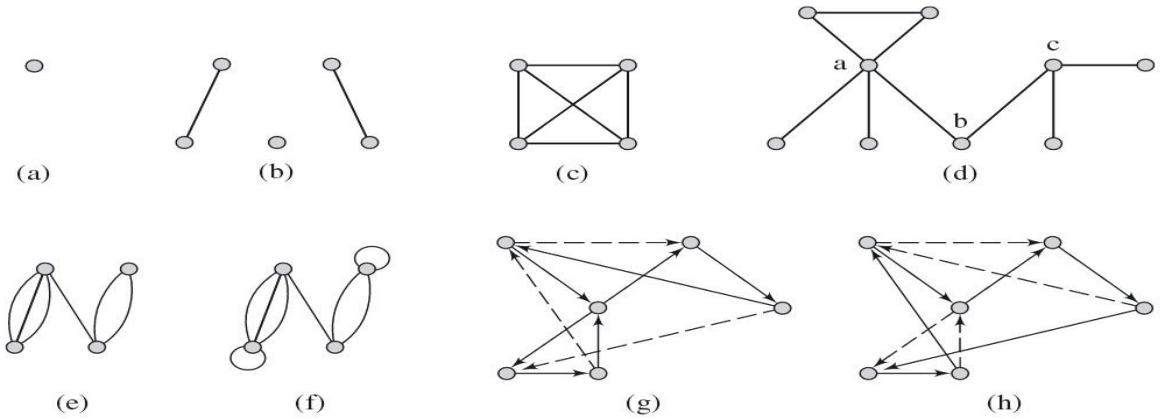
(D>0.5)

Siyrak graf esa bu qirralari soni tugunlar soniga yaqin bo‘lgan grafga qarata aytiladi. (D<0.5)



7-rasm: a) to‘yingan graf (D0.9) b) siyrak graf (D0.33)

Yuqoridagi graflarni inobatga olgan holda quyida turli xil ko‘rinishdagi graflarga misollar keltirildi.



8-rasm: a-d oddiy graf, c-to‘liq graf, e-multigraf, f-psevdograf, g-digrafdagi zanjir, h-digrafda xalqalar

Graflarni ifodalash usullari

Yo‘naltirilmagan va yo‘naltirilgan hamda o‘girlikka ega bo‘lgan graflarni kompyuter dasturlash tillari xotirasida ifodalash, ya’ni xotirada tashkil etish uchun statik tuzilmasi matritsadan yoki dinamik tuzilmasi ro‘yxatlardan foydalanish eng yaxshi usul hisoblanadi. Har qanday masalalarda har bir usulining o‘zining afzalligi va kamchiliklari mavjuddir. Yo‘naltirilmagan va yo‘naltirilgan hamda og‘irlikka ega bo‘lgan graflarni ifodalash uchun har bir usul o‘zining qoidasiga binoan shakllanadi. Shunday ekan, hozir aynan shunday to‘rtta usullarga e’tibor qaratiladi:

- Qo‘shma matritsa (adjacency matrix);
- Qo‘shnilik ro‘yxati (adjacency list);

- Qirralar ro‘yxati (edges list).
- Intsidientlik matritsa (incidence matrix);

Bu yerda G grafning **qo‘shma matritsasi** hisoblanib, u n-o‘lchamli A kvadrat matritsa hisoblandi. Bu graf uchun quyidagilar o‘rinli:

Aij 1 agar i va j tugunlar qirra bilan birlashtirilgan bo‘lsa;

Aij 0 agar i va j tugunlar o‘rtasida qirra mavjud bo‘lmasa;

orgraf uchun:

Aij 1 agar i tugundan j tugunga yoy mavjud bo‘lsa;

Aij 0 agar i va j tugunlarda yoy tugallanmagan bo‘lsa;

vaznga ega graf uchun:

Aij Wij agar i va j tugunlar qirra (yoy) bilan birlashtirilgan bo‘lsa;

Aij ∞ agar i va j tugunlar qirra (yoy) mavjud bo‘lmasa.

Qo‘shma matritsa asosiy diagonaliga semmitrik bo‘ladi, bunda agar yo‘naltirilmagan grafni ifodalasa, orgraflarda esa nosimmetrik hisoblanadi.

Qo‘shma matritsaning qulaylik tomonlari quyidagicha bo‘ladi:

- Grafda qirra qushish va o‘chirish oson;
- Grafda tugunlar qo‘shniligini tekshirish.
- Shu bilan birga qo‘shma matritsaning noqulayliklarini quyidagicha baholash mumkin:
- Grafda tugunlarni kiritish yoki o‘chirish;
- Siyrak graflar bilan ishlash.

G grafning **intsidientlik matritsasi** – bu n-satr(tugunlar) va m-ustunlar (qirralar)dan tashkil topgan B matritsa bo‘lib hisoblanadi, ya’ni unda:

graf uchun:

Bij 1 agar i tugun j qirra bilan to‘qnashgan bo‘lsa.

Bij 0 agar i tugun j qirra bilan to‘qnashmagan bo‘lsa.

orgraf uchun:

Bij -1 agar i tugun j yoyning eng yuqori qismi bo‘lsa.

Bij 0 agar i tugun j yoy bilan bir - biriga qarama-qarshi bo‘lsa.

Bij 1 agar i tugun j yoyning yakunlovchi qismi bo‘lsa.

vaznga ega graf uchun:

Bij ±Wij agar i tugun yoyning boshi (oxiri) bo‘lsa.

Bij 0 agar i tugun j yoy bilan qarama - qarshi bo‘lsa

Intsidientlik matritsaning qulaylik tomonlari quyidagilardan iborat:

- Qirra(yoy) o‘lchamini yoki yo‘nalishini o‘zgartirish;
- Qirra(yoy)larni qo‘shish yoki ularni yo‘qotish;
- To‘qnashuvlar (intsidientlik)ni me’yorida tekshirish.
- Intsidientlik matritsaning noqulayliklari holati ko‘rib chiqiladi:
- Tugunlarni qo‘shish yoki uni yo‘qotish;
- Siyrak graflar bilan ishlash usullari.

Mavzuga oid test savollari

1. **Graf tuzilmasini matematik holatini qanday ifodalash mumkin?**

a) G={V,E}

b) S ={D,R}

c) A={D(1..n)}

2. Agar grafning munosabatlarini tasvirlashda qirralardan foydalanilsa, u holda graf ... deyiladi?

a) Yo‘naltirilmagan graf
b) Yo‘naltirilgan graf
c) Aralash graf
d) Vaznga ega graf

3.Agar grafning munosabatlarini tasvirlashda yoylardan foydalanilsa, u holda graf ... deyiladi?

a) Yo‘naltirilgan graf
b) Yo‘naltirilmagan graf
c) Aralash graf

4. Agar grafning munosabatlarini tasvirlashda yoy va qirralardan foydalanilsa, u holda graf ... deyiladi?

a) Aralash graf

b) Yo‘naltirilmagan graf

c) Yo‘naltirilgan graf

Nazorat savollari

1. Graf deb nimaga aytiladi?
2. Graf qirrasi deb nimaga aytiladi?
3. Grafni qanday tasvirlash usullarini bilasiz, ular haqida so‘zlab bering?

Xulosa

Informatikada graf – bu tugunlar, ya’ni uchlardan tashkil topgan bo‘lib, u bo‘sh bo‘lmagan to‘plam va tugunlarni birlashtiruvchi yoylarning majmuidir.

Graf – bu murakkab chiziqsiz ko‘p bog‘lamli dinamik tuzilma bo‘lib, murakkab ob’ektlarning xususiyatlari va munosabatlarini o‘zida aks ettiradi. Ob’ektlar graf uzellari yoki tugun ko‘rinishida va uning munosabatlari yoy yoki yo‘naltirilgan qirralar kabi ifodalangan keladi. Bugungi kunda turli xil diskret xususiyatlarga ega bo‘lgan hisoblash qurilmalarini loyihalashda graflarning ahamiyati yanada kengroq bo‘ldi.

Yo‘naltirilmagan va yo‘naltirilgan hamda og‘irlikka ega bo‘lgan graflarni kompyuter dasturlash tillari xotirasida ifodalash, ya’ni xotirada tashkil etish uchun statik tuzilmasi matritsadan yoki dinamik tuzilmasi ro‘yxatlardan foydalanish eng yaxshi usul hisoblanadi. Har qanday masalalarda har bir usulining o‘zining afzalligi va kamchiliklari mavjuddir.

1.2 Graflar bilan ishlash algoritmlari. Qo‘shnilik ro‘yxati va yoylar ro‘yxati

Reja:

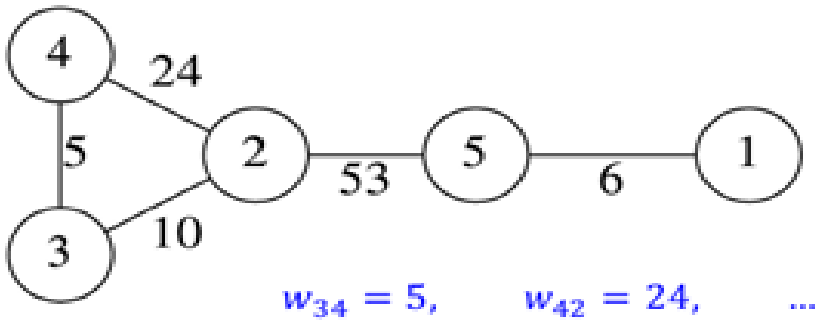
1. Og‘irlikka (vaznga) ega bo‘lgan graf va uni tasvirlash.
2. Qo‘shnilik ro‘yxati va yoylar ro‘yxatlarini jadvallarda tasvirlash.

Tayanch iboralar: graf, vaznli graf, vaznsiz graf, tugun darajasi, halqa, grafning tartibi, qo‘shni tugunlar ro‘yxati, qirralar ro‘yxati.

Og‘irlikka (vaznga) ega bo‘lgan graf va uni tasvirlash

Yo‘naltirilgan va yo‘naltirilmagan hamda o‘girlikka ega bo‘lgan graflarni dasturlash tilida ifodalab, ularni kompyuter xotirasida joylashtirishda, ya’ni xotirada tashkil etish uchun statik tuzilma matritsadan yoki dinamik tuzilma ro‘yxatlardan foydalanish yetarli bo‘ladi.

Bundan tshqari **og‘irlikka (vaznga) ega bo‘lgan graf** (weighted graph) – bu qirralari (yoylari) og‘irliklari bilan berilgan graf hisoblandi. Ya’ni, bu yerda (i,j) qirraning og‘irligi w_{ij} kabi belgilanadi.



1-rasm. Og‘irlikka ega bo‘lgan graf

Grafning tartibi haqida gapiradigan bo‘lsak, agar V to‘plamning quvvati n ga teng bo‘lsa, n soni bu yerda **grafning tartibi** deyiladi. Agar V to‘plamning quvvati n ga teng bo‘lsa, hamda E to‘plamning quvvati m ga teng bo‘lsa, u holda graf **(n, m) graf** deb yuritiladi.

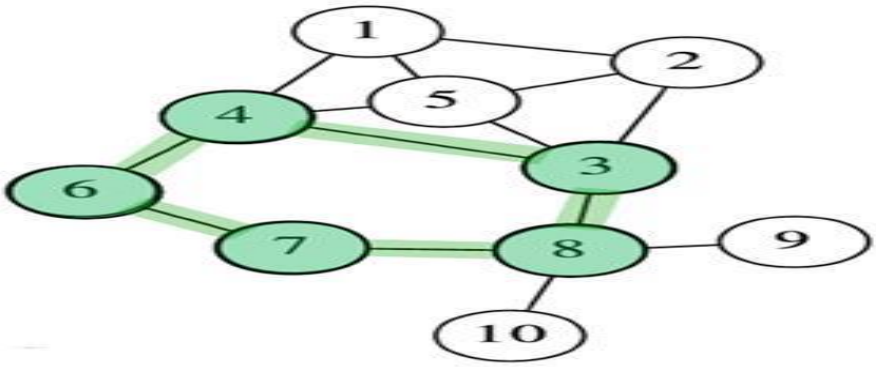
Tugun darajasi (vertex degree) – bu grafda, grafdan chiquvchi yoylar soni hisobga olinadi. deg(7) 2, deg(1) 3;

Halqa (cycle) – buni qisqa qilib aytadigan bo‘lsak, u boshi va oxirini tutashuvchi tugundan iborat yo‘l hisoblanadi: (4, 6, 7, 8, 3, 4) (2-rasm).

G(V, E) grafda uning barcha tugunlari darajalarining yig‘indisidir. Bu yerda – juft, grafning qirralari soni uning ikkilanganiga tengdir.

$$\sum_{i=1}^{n} \deg(V_i) = 2m$$

Bundan kelib chiqadiki, agar ularning darajalari mos ravishda juft yoki toq qiymatga teng bo‘lsa, tugunlar darajasiga nisbatan **juft yoki toq** deyiladi.



2-rasm. Grafning tugun va halqa darajasiga misol

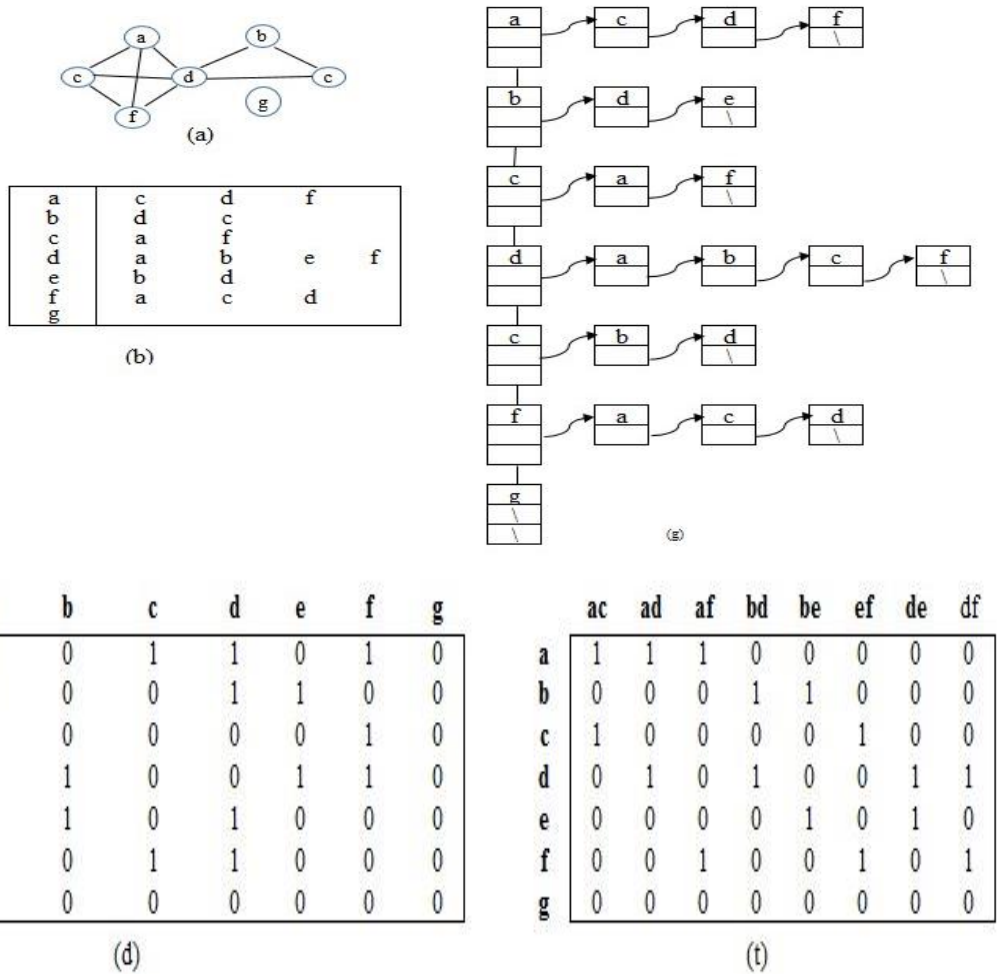
Qo‘shnilik ro‘yxati va yoylar ro‘yxatlarini jadvallarda tasvirlash

Qo‘shnilik (qo‘shni tugunlar) ro‘yxati – bu yerda **A[n]** massiv bo‘lib, u **A[i]** ni har bir elementini, ya’ni **i** tugun bilan qo‘shni tugunlar ro‘yxatlarini o‘zida saqlaydi.

Qo‘shnilik (qo‘shni tugunlar) ro‘yxati qulaylik tomonlari quyidagilardan iborat:

- Oldindan berilgan tugunga qo‘shni tugunni izlash;
- Tugun yoki qirra(yoy)larni birlashtirish;
- Siyrak graflar bilan ishlash usullari.
- Qo‘shnilik (qo‘shni tugunlar) ro‘yxatida noqulayliklari esa quyidagicha ko‘rinishda bo‘ladi:
- Qirra(yoy)ning borligini tekshirish;
- Tugun yoki qirra(yoy)larni olib tashlash.

• Berilgan tugunga intsidiyent qirra (yoy)larni aniqlash jarayoni.



3-rasm. Tugun va qirraning qo‘shniligini aniqlash usullari

Qirralar ro‘yxati – bu qirralarning qo‘shni tugunlar juftliklaridan iborat chiziqli ro‘yxati hisoblanadi.

Bu yerda qo‘shnilik, ya’ni qo‘shni tugunlarning ro‘yxatini qulaylik tomonlari quyidagicha bo‘ladi:

- Qirra (yoy)larni ko‘paytirish yoki ularni olib tashlash;
- Yoylarning yuklanishi bo‘yicha tartibga keltirish;
- Siyrak graflar bilan ishlash jarayonlari.
- Qo‘shnilik, ya’ni qo‘shni tugunlarning ro‘yxatini noqulayliklari esa quyidagi ko‘rinishda bo‘ladi:
- Tugun va qirra (yoy)ning qo‘shniligini aniqlash;

Mavzuga oid test savollari

1. Grafning tartibi nimaga teng?

a) Grafning uchlari soniga

b) Grafning qirralari soniga

c) Grafning qirra va uchlar soniga

d) Grafning ilmoqlari soniga

2. Grafning o‘lchami nimaga teng?

a) Grafni qirralar soniga

b) Grafni uchlari soniga

c) Grafni qirra va uchlar soniga

d) Grafni ilmoqlari soniga

3. Grafning tugun darajasi – bu?

a) undan chiquvchi qirralar soni hisoblanadi.

b) undan chiquvchi tugunlar soni hisoblanadi.

c) undan chiquvchi qirralar o‘rta arifmetik soni hisoblanadi.

4. Grafda nechta va qaysilar ko‘ruv amallarini ifodalaydi?

a) Ikkita (eniga va tubiga)

b) Ikkita (eniga va uzunasiga)

c) Uchta (to‘g‘ri, teskari, akslanuvchi)

5. Agar grafda boshi va oxiri bitta tugunda tutashadigan qirra mavjud bo‘lsa, unga ... deyiladi?

a) Ilmoq

b) Halqa

c) Yo‘l

Nazorat savollari

1. Grafning tugun darajasi deganda nima tushunasiz?

2. Og‘irlikka ega bo‘lgan graf deb qanday grafga aytiladi?

3. Grafda tugun va qirralarni bir-biridan farqi nimada?

Xulosa

Graf - yo‘naltirilgan va yo‘naltirilmagan hamda o‘girlikka ega bo‘lgan graflarni dasturlash tilida ifodalab, ularni kompyuter xotirasida joylashtiradi. Bunda kompyuter xotirasida tashkil etish uchun statik tuzilma matritsadan yoki dinamik tuzilma ro‘yxatlardan foydalanish yetarli bo‘ladi. Shu bilan birga, og‘irlikka ega bo‘lgan graf – bu qirralari og‘irliklari bilan berilgan graf hisoblandi.

Grafning tartibi haqida gapiradigan bo‘lsak, agar V to‘plamning quvvati n ga teng bo‘lsa, n soni bu yerda grafning tartibi deyiladi. Agar V to‘plamning quvvati n ga teng bo‘lsa, hamda E to‘plamning quvvati m ga teng bo‘lsa, u holda graf (n, m) graf deb yuritiladi. Tugun darajasi (vertex degree) – bu grafda, grafdan chiquvchi yoylar soni hisobga olinadi. Halqa (cycle) – buni qisqa qilib aytadigan bo‘lsak, u boshi va oxirini tutashuvchi tugundan iborat yo‘l hisoblanadi.

Grafda uning barcha tugunlari darajalarining yig‘indisidir. Bu yerda – juft, grafning qirralari soni uning ikkilanganiga tengdir. Bundan kelib chiqadiki, agar ularning darajalari mos ravishda juft yoki toq qiymatga teng bo‘lsa, tugunlar darajasiga nisbatan juft yoki toq deyiladi.

1.3 Graflarda ko‘ruv algoritmlari. Eniga (Breadth first search, BFS) va tubiga qarab (Depth-first search, DFS) qidiruv algoritmi

Reja:

1. Graflarda ko‘rik o‘tkazish jarayonlari (tubiga qarab qidirish).
2. Eniga qarab qidirish usullari.

Tayanch iboralar: graf, eniga qarab qidirish, bo‘yiga qarab qidirish, stek, stek tuzilmasi, navbat, navbat tuzilmasi, grafni ko‘rikdan o‘tkazish jarayoni.

Graflarda ko‘rik o‘tkazish jarayonlari (tubiga qarab qidirish).

Grafni ko‘rikdan o‘tkazish (Graph traversal) – bu berilgan tugundan boshlab barcha tugunlarni bir martadan ko‘rib chiqish amali hisoblanadi.

Ko‘rikdan o‘tkazishning ikkita usuli mavjuddir:

Chuqurligiga, ya’ni tubiga qarab izlashdir. (Depth-First Search – DFS)

Kengligiga, ya’ni eniga qarab izlashdir. (Breadth-First Search – BFS)

Bunday usullarda berilgan X tugundan boshlab bironta konteynerni qo‘llagan holda qolgan barcha tugunlarni birma-bir ko‘rib chiqadi.

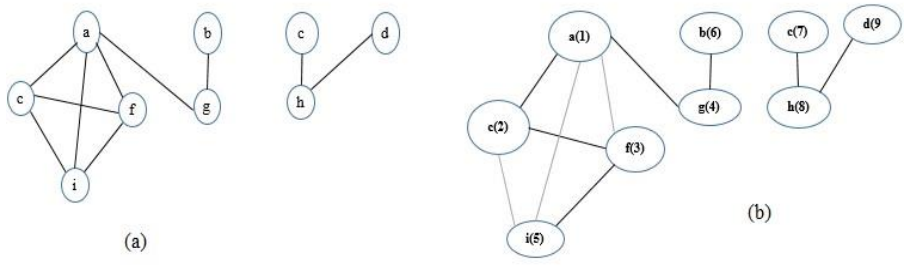
Chuqurlikka qarab qidirishda **stek tuzilmasi** qo‘llaniladi.

Kenglikka qarab ko‘rishda esa **navbat tuzilmasidan** juda yaxshi foyda beradi.

Tubiga qarab ko‘rikni o‘tkazish jarayoni dasturda psevdokod quyidagicha amalga oshiriladi:

```
DFS (v)
    Num (v) = i + + ;
       For all vertices u adjacent to v
If num (u) is 0
     Attach edge (uv) to edges;
   DFS (u);
depthFirstSearch()
        for all vertices v
 num(v) = 0; edges = null; I = l;
while there is a vertex v such that num (v) is 0
DFS(v); Output edges;
```

Yuqoridagilardan xulosa qilgan holda A tugundan boshlab tubiga qarab ko‘rib chiqishni misollarda ko‘raylik.



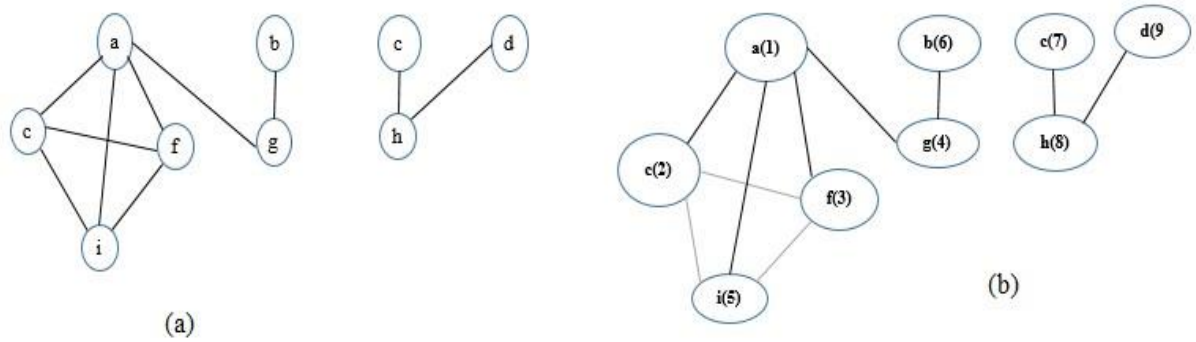
1-rasm. Graflarda ko‘rik o‘tkazish jarayonlari

Masalan eniga qarab ko‘rikni o‘tkazishda psevdokodi quyidagicha amalga oshiriladi:

```
breadthFirstSerch()
     for  all vertices u num (u) = 0;
edges = null; i = 1;
 while there ish  a  vertex v such that num (v) is 0
 num (v) = i + + ; enqueue(v);
   while queue is not empty
 v = dequeue();
            for all vertices u adjacent to v
 if num (u) is 0 num(u) = i + + ; enqueue (u);
                    attach edge (vu) to edges;
        output edges;
```

Eniga qarab qidirish usullari

Endi esa A tugundan boshlab uning eniga qarab ko‘rib chiqish misolini ko‘rib chiqaylik:



2-rasm. Eniga qarab qidirish usullari

Graflarni ko‘rikdan o‘tkazish jarayoni

Grafni ko‘rikdan o‘tkazish (Graph traversal) jarayoni – bu berilgan tugundan boshlab barcha tugunlarni ko‘rib chiqish protsedurasidir.

Uning ikkita usuli mavjud bo‘lib, ular quyidagilardan tashkil topgan:

Ulardan biri bu tubiga qarab (Depth-First Search – DFS)

Ikkinchisi esa eniga qarab (Breadth-First Search – BFS)

Bu usullar berilgan V tugundan boshlab bironta konteynerni qoldirmagan holda barcha tugunlarni koʻrib chiqish imkoniyatini beradi.

Bunda koʻpincha tubiga qarab koʻrishda steklar qoʻllaniladi. Eniga qarab koʻrishda esa navbat ishlatiladi.

Hozirgi kunga kelib, graflarda eng qisqa yoʻlni aniqlash uchun koʻplab algoritmlar yaratilgan boʻlib, ularni amalga oshirish masalaning berilishiga qarab foydalanish imkoniyatini beradi.

Hayotiy masalaga e'tibor beradigan boʻlsak, odatda vaznga ega boʻlgan graf tuzilmalarida eng qisqa yoʻlni aniqlash algoritmlari amalga oshirilib kelgan. Bu yerda vaznga ega boʻlgan graf tuzilmasini kompyuter xotirasiga saqlash uchun quyidagi belgilanishlardan foydalaniladi:

n – tugunlar soni;

m – qirralar soni;

g[n][n] – grafning qoʻshma matritsasi hisoblanadi;

g[n][m] – grafning intsidientlik matritsasi hisoblanadi;

e[m] – grafning qirralar roʻyxati (uchta maydondan iborat (boshlangich va yakunlovchi tugunlar raqami va qirraning ogʻirlik qiymatidan)) hisoblanadi;

w[i][j] – qirraning ogʻirligi ya'ni uning vazni yoki oʻlchami matritsasi hisoblanadi.

Mavzuga oid test savollari

1. Agar grafning toʻyinganligi D darajasi 0.5 dan katta boʻlsa, u holda graf ... hisoblanadi?

a) Toʻyingan

b) Siyrak

c) Ikkilamchi

2. **Agar grafning toʻyinganligi D darajasi 0.5 dan kichik boʻlsa, u holda graf ... hisoblanadi?**

a) Siyrak

b) To‘yingan

c) Ikkilamchi

3. Agar grafning to‘yinganligi D darajasi 1 ga teng bo‘lsa, u holda graf ... hisoblanadi?

a) To‘liq

b) Siyrak

c) To‘yingan

4. **G grafni aks etishda n o‘lchamli A kvadrat matritsasi qanday nomlanadi?**

a) Qo‘shma matrisa

b) Munosabat matrisasi

c) Qo‘shnilik ro‘yxati

Nazorat savollari

1. Graf deb nimaga aytiladi, unga ta’rif bera olasizmi?
2. Graf qirrasi deb nimaga aytiladi?
3. Qo‘shni tugun deb nimaga aytiladi?

Xulosa

Grafni ko‘rikdan o‘tkazish jarayoni – bu berilgan tugundan boshlab barcha tugunlarni bir martadan ko‘rib chiqish amali hisoblanadi. Bunda ko‘rikdan o‘tkazishning ikkita usuli mavjud bo’lib ular chuqurligiga, ya’ni tubiga qarab va kengligiga qarab izlashdir. Bunday usullarda berilgan tugundan boshlab bironta konteynerni qo‘llagan holda qolgan barcha tugunlarni birma-bir ko‘rib chiqish jarayoni amalga oshiriladi. Bu yerda chuqurlikka qarab qidirishda stek tuzilmasi qo‘llaniladi. Kenglikka qarab ko‘rishda esa navbat tuzilmasidan juda yaxshi foyda beradi.

Grafni ko‘rikdan o‘tkazish **(**Graph traversal) – bu berilgan tugundan boshlab barcha tugunlarni bir martadan ko‘rib chiqish amali hisoblanadi.

Ko‘rikdan o‘tkazishning ikkita usuli mavjuddir:

Chuqurligiga, ya’ni tubiga qarab izlashdir. (Depth-First Search – DFS)

Kengligiga, ya’ni eniga qarab izlashdir. (Breadth-First Search – BFS)

Bunday usullarda berilgan tugundan boshlab bironta konteynerni qo‘llagan holda qolgan barcha tugunlarni birma-bir ko‘rib chiqadi. Chuqurlikka qarab qidirishda stek tuzilmasi qo‘llaniladi. Kenglikka qarab ko‘rishda esa navbat tuzilmasidan juda yaxshi foyda beradi.

1.4 Graflarda eng qisqa yo‘lni aniqlash algoritmlari. Graflarda eng qisqa yo‘lni aniqlash masalalari. Graflarda eng qisqa yo‘lni aniqlash algoritmlar tahlili. Floyd-Uorshell algoritmi

Reja:

1. Graflarda eng qisqa yo‘lni aniqlash usullari va ularni tahlil qilish.
2. Floyd-Uorshell algoritmi hamda uning vazifasi.
3. Misollar yechish.

Tayanch iboralar: graf, eniga qarab qidirish, bo‘yiga qarab qidirish, multigraf, psevdograf, yo‘naltirilgan graf, to‘liq graf, to‘yingan graf, siyrak graf, tugun, chekli, cheksiz, juftlik, qirra.

Graflarda eng qisqa yo‘lni aniqlash usullari va ularni tahlil qilish

Hozirgi kunda graflar nazariysida eng qisqa yo‘lni aniqlash muhim klassik masalalaridan biri deb tan olingan. Uni hisoblash va yechimlarni topish uchun bir qancha algoritmlari hozirgi vaqtda mavjud.

Bunda olib qaraydigan bo‘lsak, eng qisqa yo‘l masalasi bu – ingliz tilida shortest path problem, ya’ni yoylar vaznlarining yig‘indisi minimal qiymatga ega bo‘lsagina, bu grafning ikkita tugun orasidagi eng kichik yo‘l (masofa, zanjir, marshrut) topish masalasi hisoblanadi.

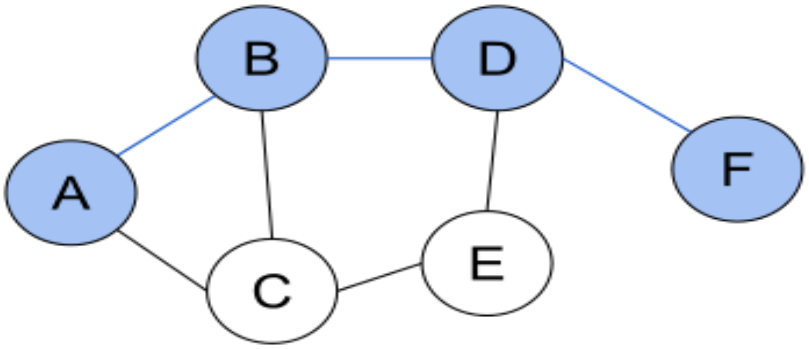
Bundan tashqari qisqa (oddiy) zanjir geodezik zanjir ham deb yuritiladi.

Ushbu masalani adabiyotlarda bir nechta boshqacha nomlanishlari ham keltirib o‘tildi. Masalan: minimal masofa masalasi, dilijans masalasi, qisqa masofa masalasi va boshqalar ko‘rinishida keladi.

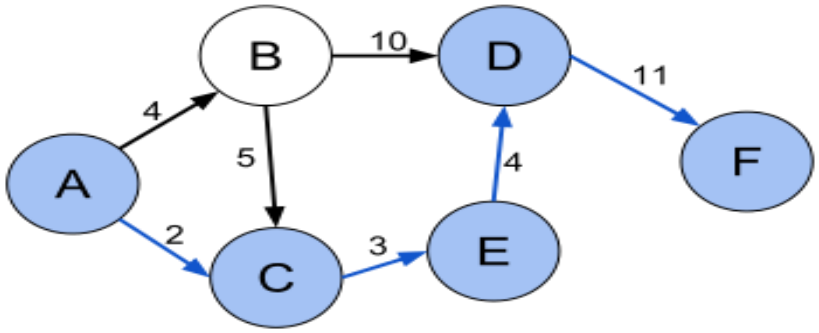
Grafda eng qisqa masofani topish masalasi yo‘naltirilgan bo‘ladi.

Unda yo‘naltirilmagan va aralash graflarda yechimini aniqlash oson kechadi. Buni 1-rasm misolida ko‘rish mumkin.

A)



B)



1-rasm. Grafda eng qisqa masofani topish masalasi:

A) yo‘naltirilmagan grafda, B) yo‘naltirilgan grafda.

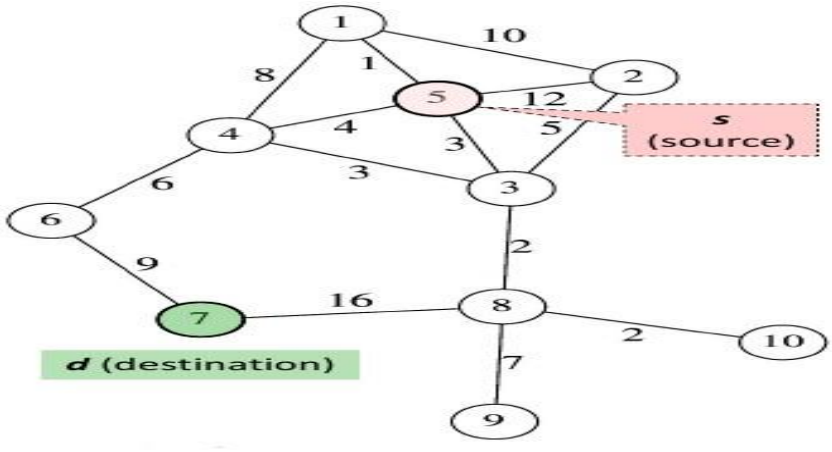
Masalani formal qo‘yilishiga e’tibor beraylik:

G (V, E) yuklanishga ega bo‘lgan graf berilgan bo‘lsin. Bu yerda E (i, j) har bir yoyning og‘irligi berilgan - $w_{ij.}$ Undan tashqari boshlang‘ich tugun s∈V va oxirgi tugun d∈V berilgan bo‘lsin. Natijada ular orasidagi qisqa masofali yo‘lni aniqlash talab etiladi.

Yo‘l uzunligi (path length, path cost, path weight) – unga kiruvchi yoylar yuklanishlari yig‘indisiga teng bo‘lsin (1 formula).

Masofani topish masalasining formal qo‘yilishi:

$$\mathbf{m}\, L = \sum w_{ij} \qquad (1)$$



2-rasm. Grafda eng qisqa masofali yo‘lni aniqlash

Eng qisqa yo‘lni aniqlash masalasi har xil masalalarda berilishiga qarab quyidagicha talaffuzlarga ega bo‘ladi: Misol uchun Li(to‘lqinli) algoritmi, A star (A*) algoritmi, Djonson algoritmi, Viterbi algoritmi, Cherkasskiy algoritmi va boshqalar ko‘rinishida bo‘ladi.

Floyd-Uorshell algoritmi va uning vazifasi

Floyd-Uorshell algoritmi aslida istalgan tugunlardan boshlab qolgan barcha tugunlargacha bo‘lgan masofalarni hisoblash uchun amalda qo‘llaniladi. Algoritm samaradorligi amallar bajarilishi bo‘yicha $\mathbf{n^3}$ tartibli ko‘rinishda hisoblanadi. Bu yerda qirralar o‘girlik qiymatlari manfiy ham bo‘lishi mumkin, ammo manfiy qiymatga ega bo‘lgan qirralar halqa ko‘rinishida berilmagan bo‘lishi shart, chunki bunda algoritm teskari bo‘lib qolishi ham mumkin.

Algoritm g‘oyasi:

d[0 . n–1][0 . n–1] masofalar matritsasi har i-chi qadamda javobini saqlash uchun ishlatiladi va har keyingi qadamda i–1-dan kichik bo‘lgan tugunlarga o‘tish orqali kerakli masofani hisoblay boshlaydi. Masalan, i-chi qadamni amalga oshiraylik. i 1 tugungacha masofalarni yangilash uchun i–1 tugun masofasi tanlanadi va masofalar shart orqali tekshirib olinadi. Agar masofa kichikroq bo‘lsa, u holda barcha qadamlar soni n 1 qiymatga teng bo‘lgunicha bu masofa qiymatini yangilanadi. Oxirgi qadamdan so‘ng bir tugundan boshqa tugunlarga qisqa masofalar qiymati aniqlanadi va u d matritsasida saqlanadi.

Algoritmning psevdokodi:

g grafni o‘qiladi

// g[0 ... n - 1][0 ... n - 1] - qirallar og‘irliklari matritsasi, g[i][j] 2000000000,

agar i va j tugunlar orasida qirra mavjud bo‘lmasa d g

for i 1 ... n 1

for j 0 ... n – 1

for k 0 ... n - 1

if d[j][k] > d[j][i - 1] d[i - 1][k]

d[j][k] d[j][i - 1] d[i - 1][k]

d matritsa natijasini ekranga chiqarish

Algoritmning c dasturida dastur kodini ko‘rib chiqaylik:

```
#include <fstream.h>
#include <algorithm.h>
 int main()
{
   int n, a[101][101];
  ifstream ifs ("input.txt");
 ifs>> n;  for(inti1;i<n;i)
for(int j1;j<n;j)
   ifs>> a[i][j];
  ifs.close();
for(int k1;k<n;k)
 for(inti1;i<n;i)
 for(int j1;j<n;j)
  a[i][j]min(a[i][j],a[i][k]a[k][j]);
    ofstream ofs("output.txt");
for(inti1;i<n;i)
{
```

```
for(int j1;j<n;j)
  ofs<< a[i][j]<<" ";
    ofs<<‘\n’;
 }
 ofs.close();
  return0;
 }
```

Mavzuga oid test savollari

1. G grafni aks etishda A[n] massiv bo‘lib, massivning har bir elementi tugun bilan qo‘shni tugunlar ro‘yxati qanday nomlanadi?

a) Qo‘shnilik ro‘yxati
b) Qo‘shma matrisa
c) Munosabat matrisasi
d) Qirralar ro‘yxati

2. G grafni aks etishda qo‘shni tugunlar qirralaridan iborat chiziqli ro‘yxati qanday nomlanadi?

a) Qirralar ro‘yxati
b) Qo‘shnilik ro‘yxati
c) Qo‘shma matrisa
d) Munosabat matrisasi

3. Grafning D to‘yinganlik darajasi nimaga teng?

a) $D = \frac{2m}{n(n-1)}$

b) $D = \frac{n(n-1)}{2m}$

c) $D = \frac{n}{m}$

d) $D = \frac{m}{n}$

4. To‘liq grafning qirralar soni qanday formula orqali hisoblanadi?

a) $m = \frac{n(n-1)}{2}$

b) $m = n^2$

c) $m = n!$

d) $m = \sqrt{n}$

5. Murrakab ob’ektlarning xususiyati va munosabatlarini aks ettiruvchi chiziqsiz ko‘p bog‘lamli dinamik tuzilmasi?

a) Graf

b) Lug‘at

c) Daraxt

d) Ro‘yxat

Nazorat savollari

1. Graf nima va uning turlarini aytib bering?
2. Graflarni ko‘rikdan o‘tkazish algoritmlari deganda nima tushunasiz?
3. Qisqa masofani aniqlashning Deykstra algoritmi qanday bo‘ladi?
4. Floyd-Uorshell algoritmi haqida ma’lumot bering.

Xulosa

Graflarda eng qisqa yo‘lni aniqlash muhim masalalaridan biri sanaladi. Uni hisoblash va yechimlarni topish uchun algoritmlari hozirgi vaqtda mavjuddir. Bunda eng qisqa yo‘l masalasi – bu yoylarning vaznilarining yig‘indisi minimal qiymatga ega bo‘lsagina, bu grafning ikkita tugun orasidagi eng kichik topish masalasi hisoblanadi. Bundan tashqari qisqa zanjir geodezik zanjir deb ham ataladi.

Floyd-Uorshell algoritmi aslida istalgan tugunlardan boshlab qolgan barcha tugunlargacha bo‘lgan masofalarni hisoblash uchun amalda qo‘llaniladi. Algoritm samaradorligi amallar bajarilishi bo‘yicha $\mathbf{n^3}$ tartibli ko‘rinishda hisoblanadi. Bu yerda qirralar og‘irlik qiymatlari manfiy ham bo‘lishi mumkin, ammo manfiy qiymatga ega bo‘lgan qirralar halqa ko‘rinishida berilmagan bo‘lishi shart, chunki bunda algoritm teskari bo‘lib qolishi ham mumkin.

1.5 Graflarda eng qisqa yo‘lni aniqlash algoritmlari. Graflarda eng qisqa yo‘lni aniqlashning Ford-Belmann va Deykstra algoritmlari

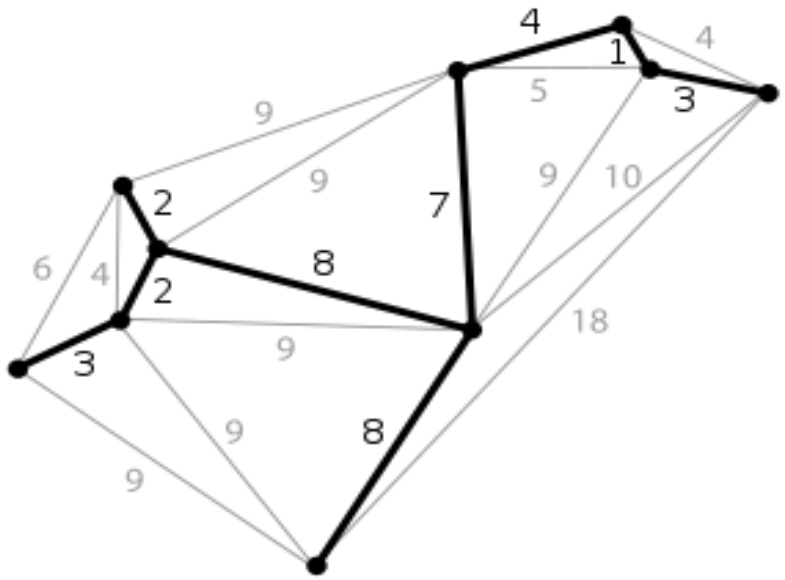
Reja:

1. Minimal narxli daraxtlar skeleti haqida umumiy tushuncha.
2. Primo algoritmi va uni misollar yordamida yoritish.
3. Ford-Belmann algoritmi unga oid misollar yechish.
4. Deykstra algoritmiga doir misollar ko‘rish.

Tayanch iboralar: graf, Primo algoritmi, Kraskala algoritmi, Boruvka algoritmi, Deykstra algoritmi, multigraf, psevdograf, siyrak graf, tugun, chekli, cheksiz.

Minimal narxli daraxtlar skeleti haqida umumiy tushuncha

Minimal narxlisini topish masalasi juda ko‘p holatlarda quyidagicha ko‘rinishda uchraydi: masalan, n ta shahar mavjud va ular orasiga yo‘llarni shunaqa qurish kerakki, istalgan shahardan ixtiyoriy boshqasiga bevosita yoki bilvosita borish imkoniyati mavjud bo‘lsin va ular orasiga yo‘l qurish sarf harajatlari aniq bo‘lsin. Yo‘llarni qurishni shunday rejalashtirish kerakki, natijada sarf harajat eng kam ishlatilgan bo‘lsin.



1-rasm. Vaznga ega bo‘lgan yo‘naltirilmagan graf

Bu masala graflar nazariyasida minimal narxli daraxt skeleti (ostovnoye derevo) deb nomlanadi.

Ushbu masalani yechishning bir qator algoritmlari mavjud:

1. Primo algoritmi;

2. Kraskala (Kruskala) algoritmi;

3. Boruvka algoritmi.

Primo algoritmi va uni misollar yordamida yoritish

Ushbu algoritm birinchi bo‘lib, 1930-yilda matematik Voysex Yarnik tomonidan ilgari surilgan.

Keyin Robert Primo tomonidan 1957-yilda qayta ko‘rib chiqilgan va ulardan tashqari E. Deykstra tomonidan 1959-yilda ishlab chiqilgan.

Algoritmda kirishiga yo‘naltirilmagan og‘irlikka ega bo‘lgan graf o‘zlashtiriladi.

1. Boshida istalgan tugun olinadi va unga tegishli bo‘lgan eng kam vaznga ega insident yoy topib olinadi. Topilgan yoy va unga tegishli tugunlar daraxtni shakllantira boshlaydi.

2. Yana bir uchi bilan daraxtga tegishli bo‘lgan tugunlariga tutashgan va ikkinchi uchi esa tutashmagan boshqa yoylar bilan ham tekshiriladi. Ularning ichidan vaznga ko‘ra, og‘irligi kam bo‘lgani tanlab olinadi.

3. Har bir qadamdagi yoy daraxtga qo‘shilib boraveradi. Daraxtning balandligi ham 1 taga oshib ketaveradi.

Grafning barcha tugunlari ko‘rib chiqilmaguncha algoritm to‘xtatilmaydi.

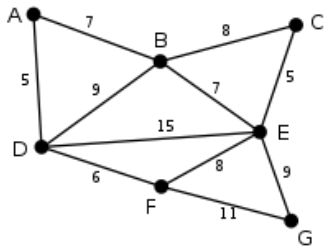
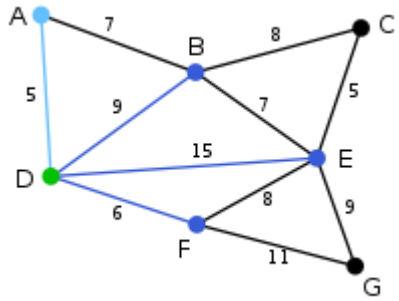
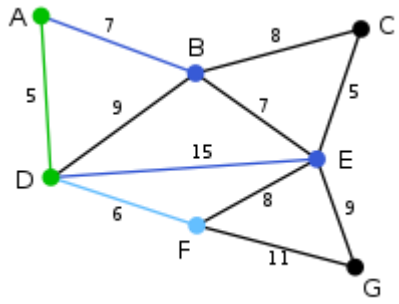
Bu yerda algoritm natijasi bo‘lib, minimal narxli daraxt skeleti hisoblanadi.

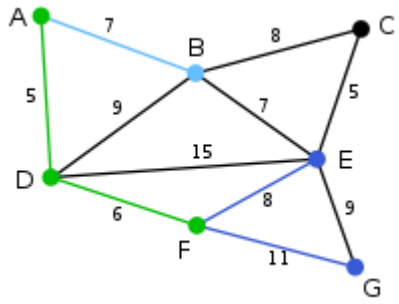
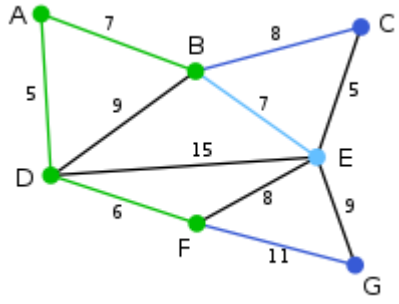
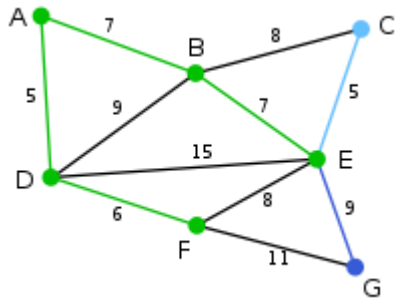
Bunda e’tibor qaratish lozimki, minimal narxli daraxtlarni hosil qilishda halqa paydo bo‘lishiga aslo yo‘l qo‘ymaslik kerak.

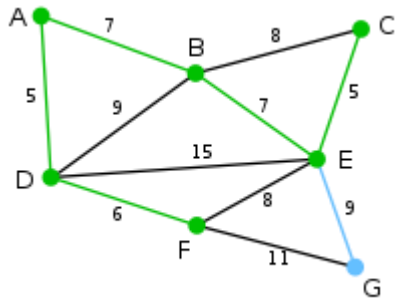
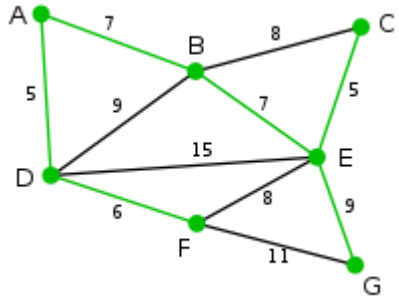
Yuqoridagilarga doir misollar keltiraylik.

1-jadval

Primo algoritmiga doir misollar

Tasvir	U tanlangan tugunlar to‘plami	(u,v) yoylar	v\u tanlanma-gan tugunlar	izox
	{}		{A,B,C, D, E,F,G}	Dastlab darax tugunlar to‘plami bo‘sh
	{D}	(D,A) 5 **V** (D,B) 9 (D,E) 15 (D,F) 6	{A,B,C, E,F,G}	Ilk tugun sifatida D olindi. Unga tegishli yoylar A,B,E,F tugunlarga olib boradi. Ularning minimal yoyligi tanlanadi, ya’ni A
	{A,D}	(D,B) 9 (D,E) 15 (D,F) 6 **V** (A,B) 7	{B,C,E,F ,G}	

	{A,D,F}	(D,B) 9 (D,E) 15 (A,B) 7 **V** (F,E) 8 (F,G) 11	{B,C,E, G}	
	{A,B,D,F }	(B,C) 8 (B,E) 7 **V** (D,B) 9 sikl (D,E) 15 (F,E) 8 sikl (F,G) 11	{C,E,G}	(D,B) yoy tanlansa halqa hosil bo‘ladi. Shuning uchun u tanlaymaydi.
	{A,B,D,E, F}	(B,C) 8 (D,B) 9 sikl (D,E) 15 sikl (E,C) 5 **V** (E,G) 9 (F,E) 8 sikl	{C,G}	

		(F,G) 11		
	{A,B,C, D,E,F}	(B,C) 8 sikl (D,B) 9 sikl (D,E) 15 sikl (E,G) 9 **V** (F,E) 8 sikl (F,G) 11	{G}	
	{A,B,C, D,E,F}	(B,C) 8 sikl (D,B) 9 sikl (D,E) 15 sikl (F,E) 8 sikl (F,G) 11 sikl	{}	Grafdagi barcha tugunlarni tekshirib bo‘ldi.

Ford-Belmann algoritmi va unga doir misollar yechish

Bu algoritm berilgan tugundan (uni 0 deb belgilanadi) barcha boshqa tugunlarga bo‘lgan qisqa masofalarni hisoblash uchun amalda tadbiq qilinadi. Algoritm samaradorligi amallar bajarilishi bo‘yicha **n*m** tartibli bo‘ladi. Bu

algoritmda ham qirralar o‘girlik qiymatlari manfiy bo‘lishi mumkin va halqa ko‘rinishida berilmagan bo‘lishi shart.

Algoritm g‘oyasi: Bu g‘oya d [0 . n–1] masofalar massivi har i-chi qadamda javobini saqlash uchun qo‘llaniladi va har bir qadamda i-dan oshmagan qirralar soni ishtirokida masofa hisoblash uchun undan foydalaniladi. Agar j-tugunga yo‘l mavjud bo‘lmasa, u holda d[j] 2000000000 (ya’ni cheksiz qiymatga teng deb qabul qilinadi).

Birinchi qadamda d massiv cheksiz qiymatlar bilan to‘ldiriladi, keyingi qadamda qirralar ko‘rib chiqiladi va masofani yangilash uchun qayta-qayta tekshiriladi. Agarda qirradan ushbu tugunga marshrutlar mavjud bo‘lsa, u holda masofalar solishtirib ko‘riladi. Bunda yangi qiymat kichik bo‘lsa, u holda massiv yangilanadi, aks holda o‘z holatida qoldiriladi. Shuni ta’kidlash lozimki, qisqa masofani aniqlashda halqaning ishtirokidan umuman foydalanilmaydi.

Algoritmning psevdokodi: Bunda birinchi bo‘lib, g graf o‘qib olinadi: ya’ni e qirralar ro‘yxati shakllantiriladi.

// e[0 ... m - 1] – qirralar ro‘yxati massivi, qaysida (first, second - tugunlar, value – qirra o‘g‘irligi)

for i 0 ... n - 1

d[i] 2000000000

d[0] 0 // 0 – tanlangan tugun

for i 1 ... n

for j 0 ... m – 1

if d[e[j].second] > d[e[j].first] e[j].value

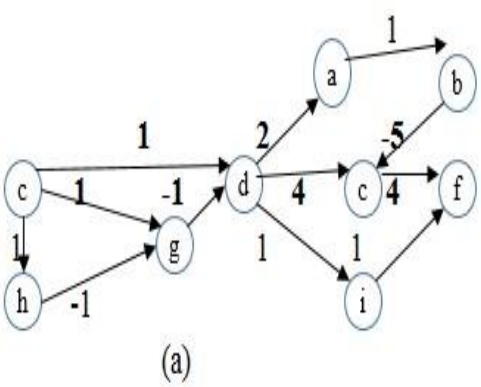
d[e[j].second] d[e[j].first] e[j].value

if d[e[j].first] > d[e[j].second] e[j].value

d[e[j].first] d[e[j].second] e[j].value

d massiv natijasi ekranda chiqariladi.

Ford-uorshell algoritmiga natija olish qadamlari



(a)

Qirralarning tartibi: ab be cd cg ch da de di ef gd hg if

standartlrni hsoblsh:

	tip	1		2	3	4
a	∞	3		2	1	
b	∞			4	3	2
c	0					
d	∞	1	0	-1		
e	∞	5		-1	-2	-3
f	∞	9	3	2	1	
g	∞	1	0			
h	∞	1				
i	∞	2		1	0	

(b)

2-rasm. Ford-Belmann algoritmiga doir misol

C dasturlash tilida algoritmning dastur kodini ko‘rib chiqaylik:

```
#include <fstream.h>
#include <algorithm.h>
int main()
    {
   int n, a[101][101];
  ifstream ifs ("input.txt");
   ifs>> n;
  for(inti1;i<n;i)
  for(int j1; j<n;j)
   ifs>> a[i][j];
    ifs.close();
  for(int k1;k<n;k)
  for(inti1; i<n;
for(int j1; j<n;j)
```

```
a[i][j]min(a[i][j],a[i][k]a[k][j]);
ofstream ofs("output.txt");
 for(inti1;i<n; i)
    {
 for(int j1;j<n;j)
 ofs<< a[i][j]<<" ";
      ofs<<‘\n’;
    }
 ofs.close();
  return0;
}
```

Deykstra algoritmiga doir misollar yechish

Bunda berilgan tugundan (uni 0 deb belgilanadi) boshqa barcha tugunlargacha bo‘lgan qisqa masofalarni hisoblab chiqishda bu algoritm qo‘llaniladi. Algoritm samaradorligi esa amallar bajarilishi bo‘yicha $\mathbf{n^2}$ tartibi bilan hsobga olinadi. Bu algoritmning eng muhim jihati shundaki, unda qirralar og‘irlik qiymatlari faqatgina musbat bo‘lishi shart.

Buning algoritm g‘oyasini ko‘rib chiqaylik: Qisqa masofalarni aniqlash uchun tugunlardan qaysi birini masofasini yaqinlashtirishni osonlashtirish uchun qo‘shimcha mantiqiy elementlardan tashkil topgan mark[0..n–1] massiv kiritiladi. Bunda agar ushbu tugundan o‘tilgan (belgilangan) bo‘lsa, elementlar qiymati **true** qiymatga teng bo‘ladi, aks holda, ya’ni o‘tilmagan (belgilanmagan) bo‘lsa, **false** qiymatni qabul qiladi. d[0..n–1] masofalar massivi har i-chi qadamda javobini saqlash uchun ishlatiladi va har qadamda i-dan oshmagan qirralar soni ishtirokida masofa hisoblash uchun undan foydalaniladi.

Boshlang‘ich qadamda 0 tuguni belgilanadi, u d[i]x (qirra og‘irligiga)dan iborat bo‘lsin, bunda ham agar 0 va i tugunlar orasida qirra mavjud bo‘lsa, u d[i] 2000000000 (cheksiz qiymatga), aks holda agar qirra 0 va i tugunlar o‘rtasida bo‘lmasa jarayon to‘xtatiladi. Bundan keyingi qadamda belgilanmagan tugunlar

o‘rtasidan eng minimal qiymatga ega bo‘lgan tugun tanlanadi, unga v deb belgilash kiritiladi. U holda d[v] v tugun uchun javobi deb hisoblanadi. Keyin v - tugun belgilab olinadi va d qiymatlari qayta hisoblab chiqiladi. Natijada bu amallarni barcha tugunlar belgilanmaguncha davom ettiraveriladi.

Algoritmning psevdokodi:

Bunda g graf o‘qiladi

// g[0 ... n - 1][0 ... n - 1] - qirallar og‘irliklari matritsasi, g[i][j] 2000000000, agar i va j tugunlar orasida qirra mavjud bo‘lmasa

d[0] 0 //0 -tanlangan tugun

d g[0]

mark[0] True

for i 1.. n - 1

mark[i] False

for i 1 .. n - 1

v -1

for i 0 .. n – 1

if (not mark[i]) and ((v -1) or (d[v] > d[i]))

v i

mark[v] True

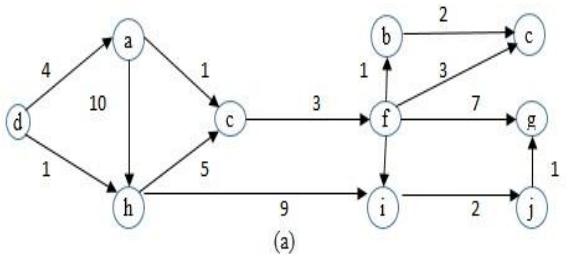
for i 0 ... n – 1

if d[i] > d[v] g[v][i]

d[i] d[v] g[v][i]

d massiv natijasini ekranga chiqarish jarayoni;

Deykstra algoritmiga natija olish qadamlari



(a)

standartlrni hsoblsh:	tip	1	2	3	4	5	6	7	8	9	10
eng faol chuqqi		d	h	a	e	f	b	i	c	j	g
a	∞	4	4								
b	∞	∞	∞	∞	∞	9					
c	∞	∞	∞	∞	∞	11	11	11			
d	0										
e	∞	∞	6	5							
f	∞	∞	∞	∞	8						
g	∞	∞	∞	∞	∞	15	15	15	15	12	
h	∞	∞									
i	∞	∞	10	10	10	9	9				
j	∞	∞	∞	∞	∞	∞	∞	11	11		

(b)

3-rasm. Deykstra algoritmiga doir misol

Algoritmning c dsturlash tilida dastur kodi:

```
#include <fstream.h>
#include <algorithm.h>
#include <vector.h>
#include <iostream.h>
 int main()   {
   int a[500][500],d[500]{0},n,s,f,flag[500],l,min1100000000,nmin0;
for(inti0;i<500;i) flag[i]1;
    ifstream ifs ("input.txt");
   ifs>> n >> s >> f;
  for(inti1;i<n;i)
    for(int j1;j<n;j)     {
  ifs>> a[i][j];
```

```
 if(a[i][j]-1&&i!j) a[i][j]32000;    }
 ifs.close();
 ls;
for(inti1; i<n; i) d[i]a[l][i];
   flag[l]0;
   for(inti1;i<n-1;i)  {
      min1100000000;
       nminl;
     for(int j1;j<n;j)
 if(flag[j]!0&& min1>d[j])   {
min1d[j];
          nminj;       }
    lnmin;
flag[l]0;
   for(int j1;j<n;j)
  if(flag[j]!0)
d[j]min(d[j], a[l][j]d[l]);   }
    ofstream ofs("output.txt");
   if(d[f]32000) ofs<<"-1";
      else ofs<< d[f];
 ofs.close();
  return 0;  }
```

Mavzuga oid test savollari

1. Bironta tugundan boshqa bir tugungacha bo‘lgan yonma-yon joylashgan tugunlar ketma-ketligi nima deyiladi?

a) Yo‘l

b) Halqa

c) Ilmoq

d) Daraja

2. ... – bu boshi va oxiri tutashuvchi tugundan iborat yo‘l?

a) Halqa

b) Yo‘l

c) Ilmoq

d) Daraja

3. **Berilgan tugundan boshlab barcha tugunlarni ko‘rib chiqish protsedurasi qanday nomlanadi?**

a) Obxodom

b) Siklom

c) Putem

d) Stepenyu

4.

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

yo‘naltirilmagan grafning qo‘shma matritsasi berilgan. Grafning tartibi nechiga teng, shuni ko‘rsating?

a) 5

b) 4

c) 7

d) 6

Nazorat savollari

1. Graflarni ko‘rikdan o‘tkazish algoritmlari deganda nimani tushunasiz?
2. Graflarda eng qisqa masofani aniqlash masalasi haqida ma’lumot bering?
3. Ford-Belmann algoritmi – bu qanday algoritm?
4. Algoritmlar samaradorligi deganda nimani tushunasiz?

Xulosa

Hozirgi vaqtda eng kichik narxlisini topish masalasi juda koʻp holatlarda uchraydi. Sababi shundaki, har tomonlam inobatga olingan bu iqtisodiy masalada qisqa masofa va kam xarajat birinchi oʻrinda turadi. Bunda ayniqsa, shaharlar orasidagi masofalar, ularni bogʻlash va boshqa jarayonlar alohida eʻtibor talab qiladi. Bunday hollarda masalani toʻgʻri taqsimlash uchun Primo algoritmi, Ford-Belmann algoritmi, Deykstra algoritmlari yaqindan yordam beradi.

Algoritmda kirishga yoʻnaltirilmagan ogʻirlikka ega boʻlgan graf oʻzlashtilriladi:

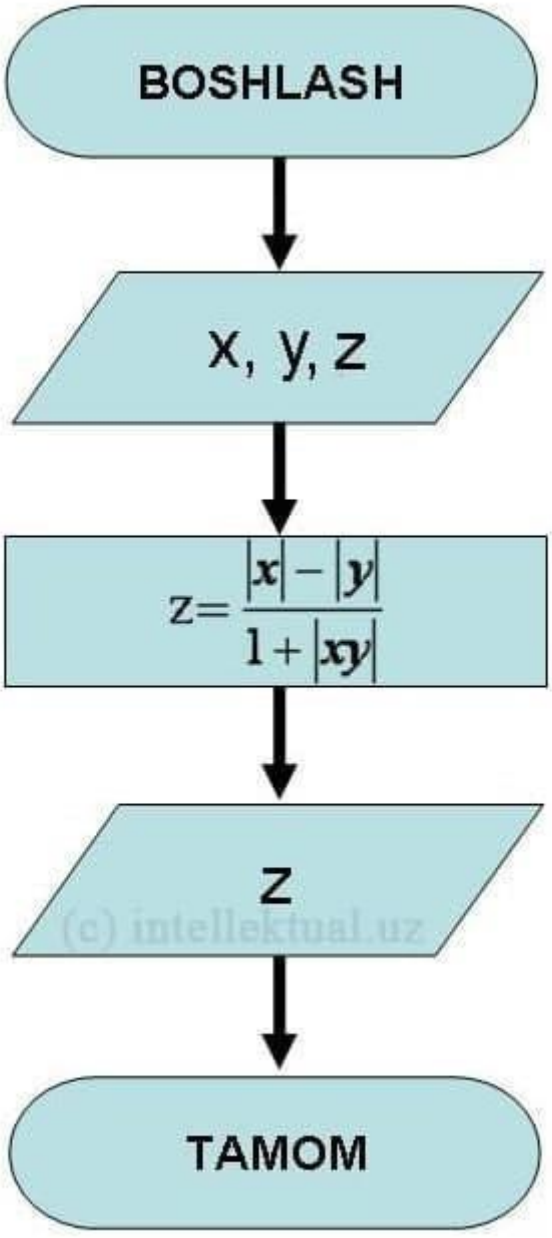
1. Boshida istalgan tugun olinadi va unga tegishli boʻlgan eng kam vaznga ega insident yoy topib olinadi. Topilgan yoy va unga tegishli tugunlar daraxtni shakllantira boshlaydi.

2. Yana bir uchi bilan daraxtga tegishli boʻlgan tugunlariga tutashgan va ikkinchi uchi esa tutashmagan boshqa yoylar bilan ham tekshiriladi. Ularning ichidan vaznga koʻra, ogʻirligi kam boʻlgani tanlab olinadi.

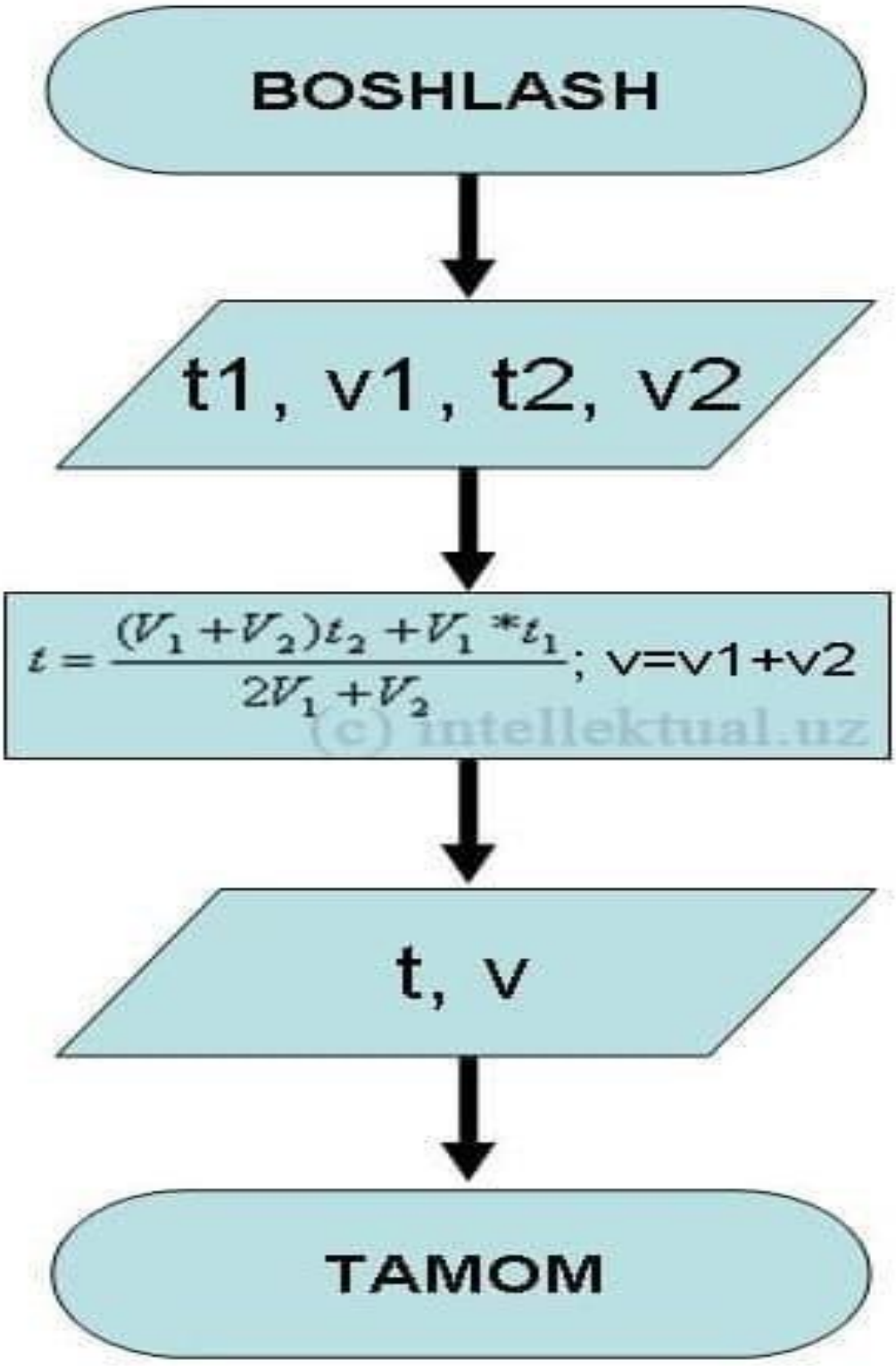
3. Har bir qadamdagi yoy daraxtga qoʻshilib boraveradi. Daraxtning balandligi ham 1 taga oshib ketaveradi.

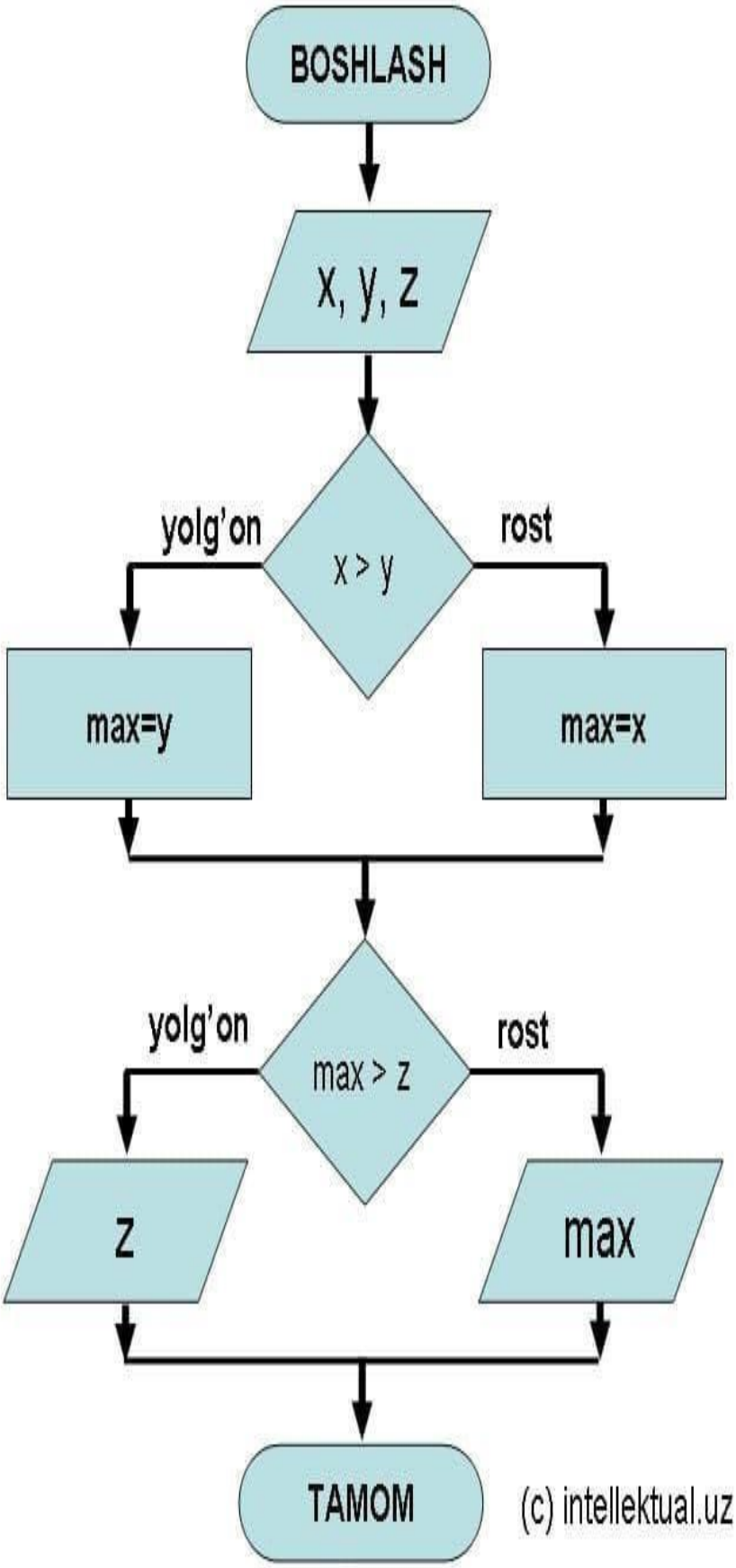
Grafning barcha tugunlari koʻrib chiqilmaguncha algoritm toʻxtatilmaydi. Bu yerda algoritm natijasi boʻlib, minimal narxli daraxt skeleti hisoblanadi. Bunda e’tibor qaratish lozimki, minimal narxli daraxtlarni hosil qilishda halqa paydo boʻlishiga aslo yoʻl qoʻymaslik kerak.

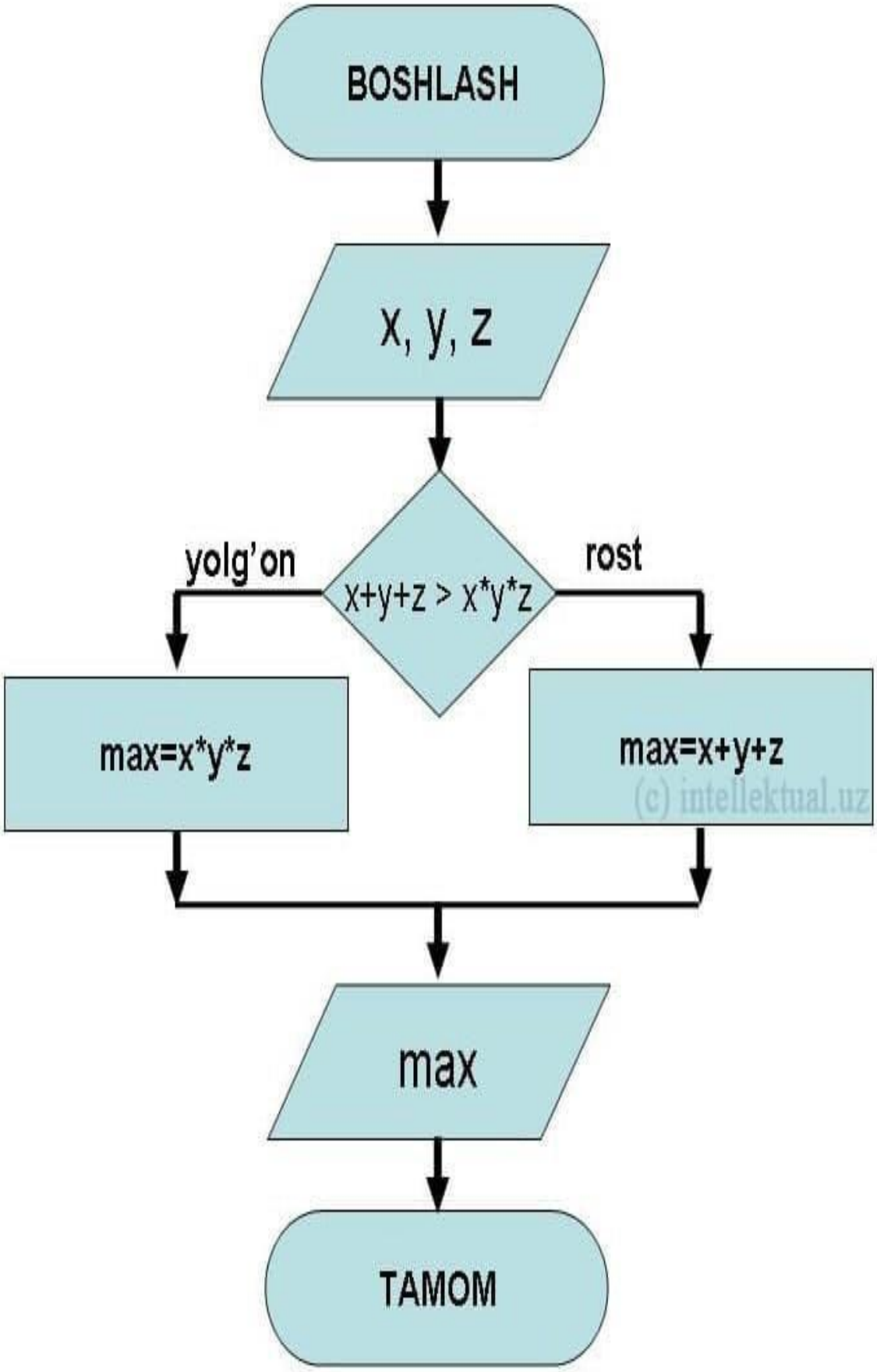
ILOVALAR

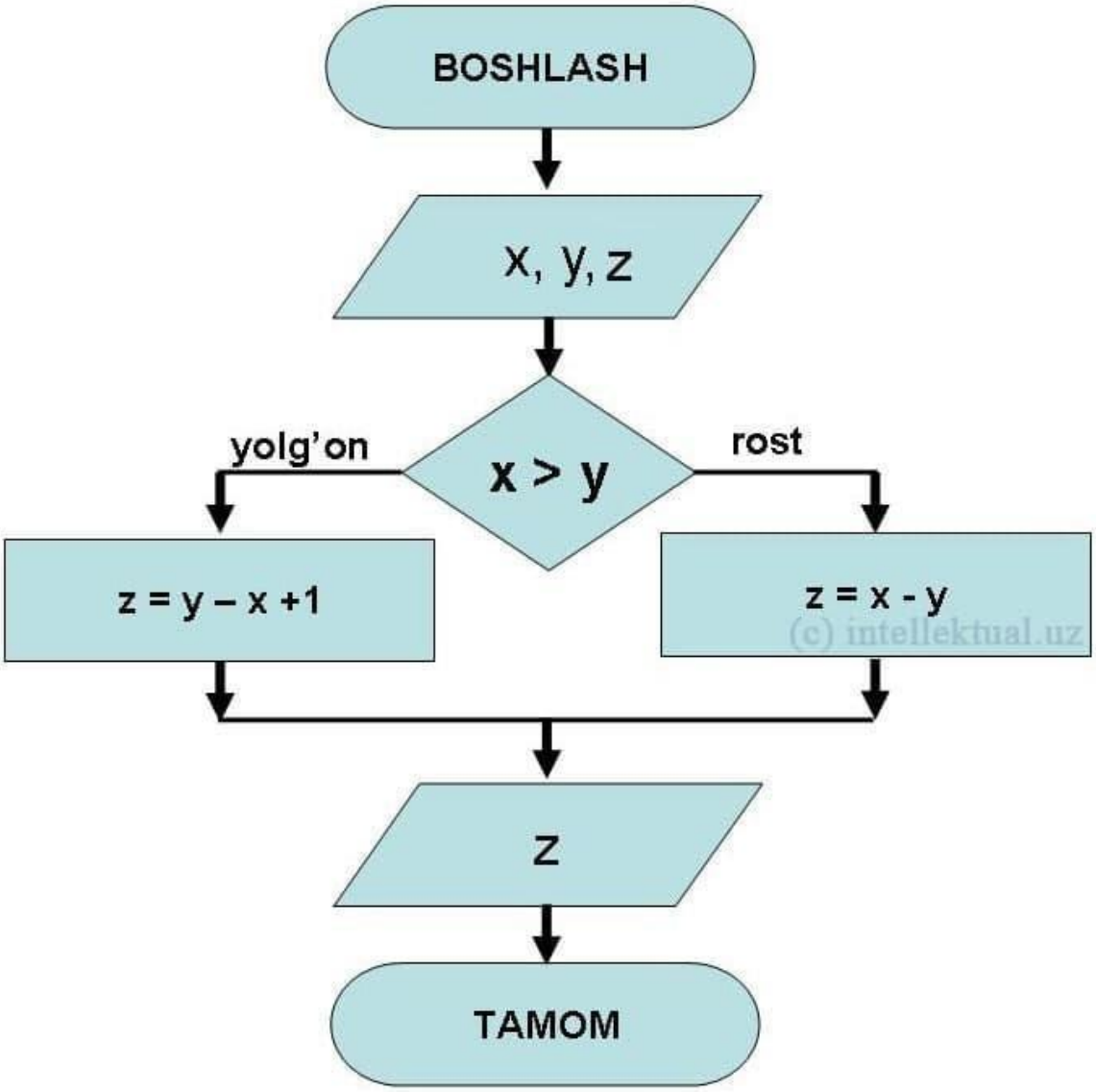


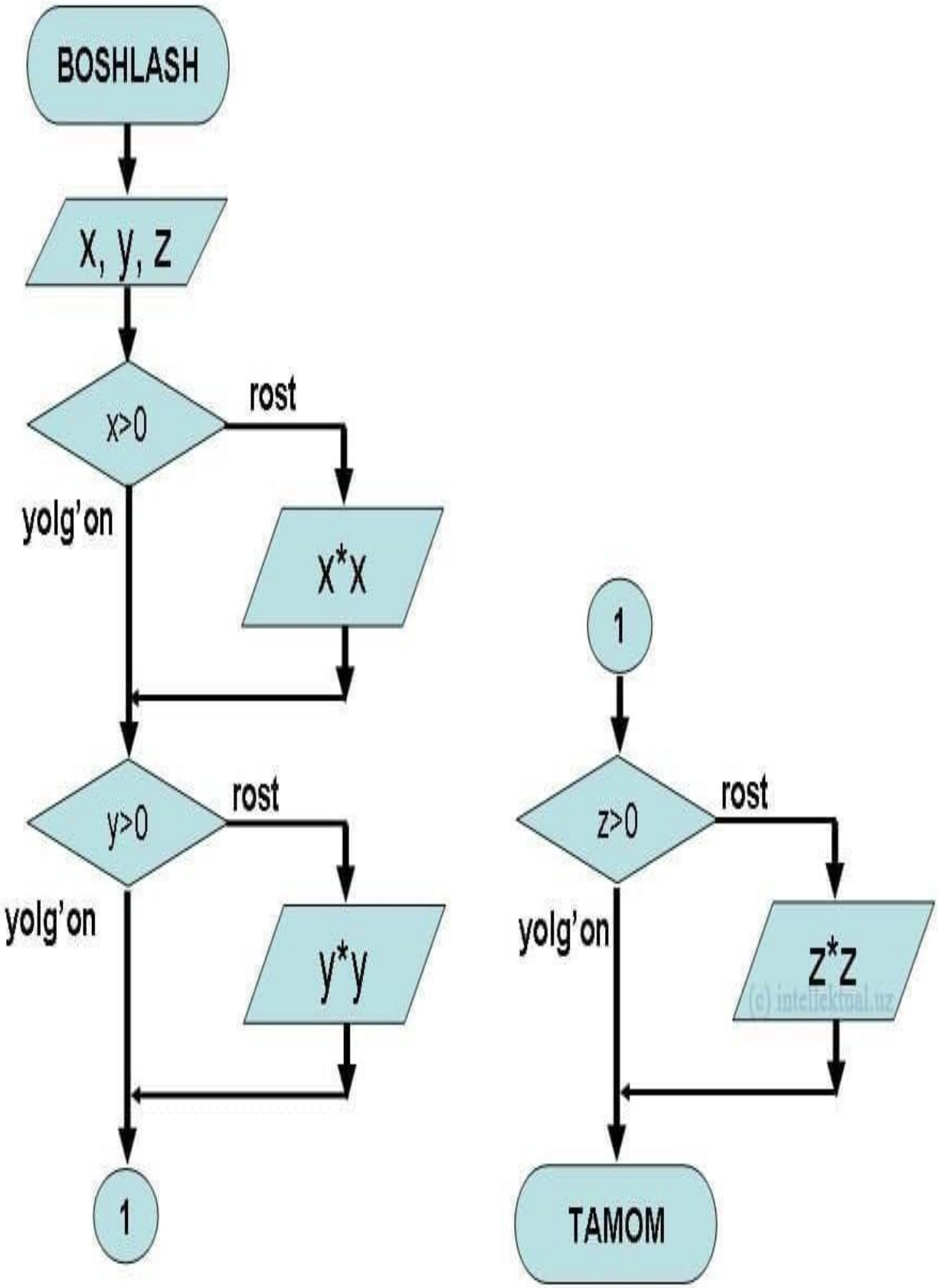
(c) intellektual.uz

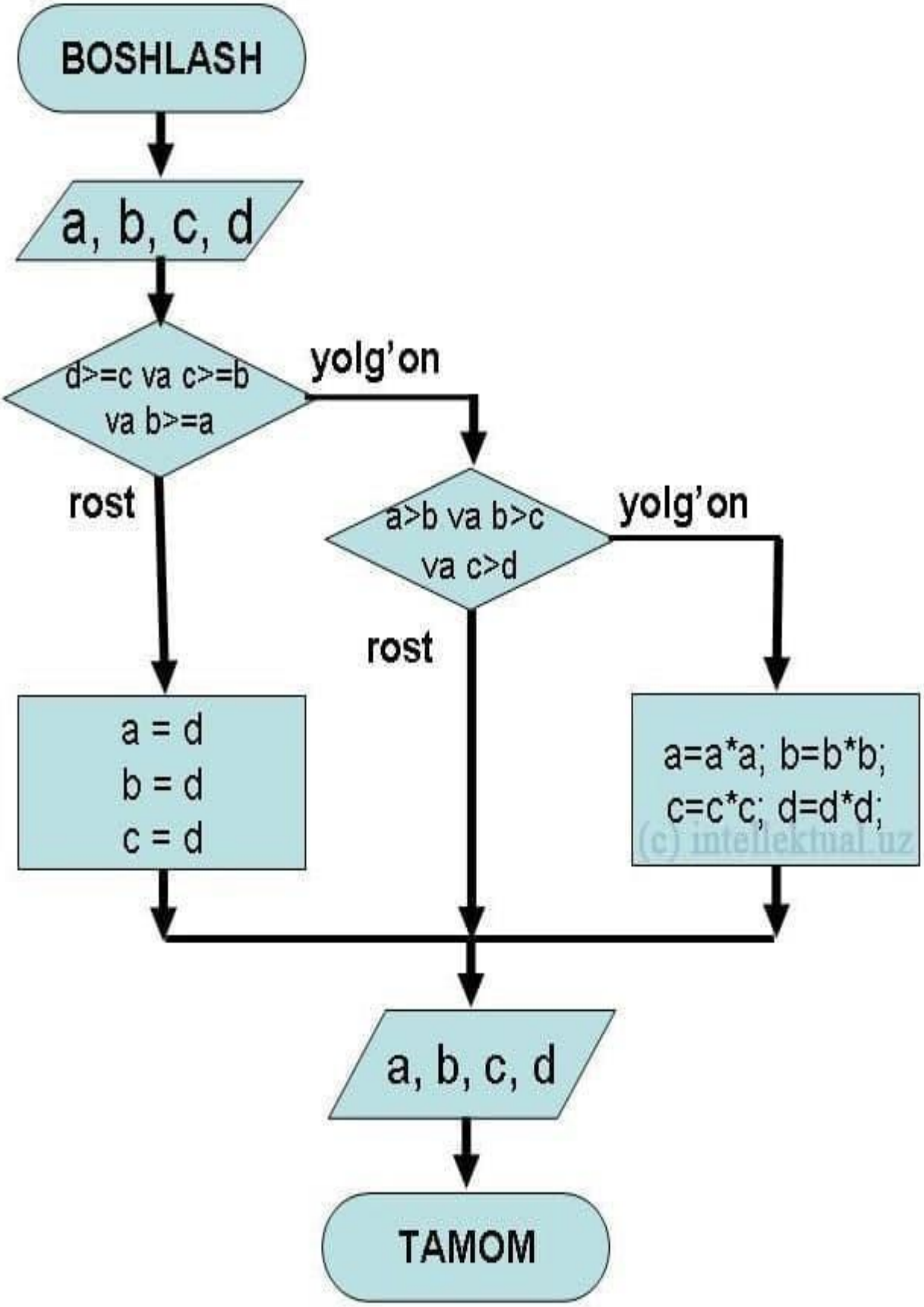

BOSHLASH
t1 , v1 , t2, v2
$t = \frac{(V_1 + V_2)t_2 + V_1 * t_1}{2V_1 + V_2}$; v=v1+v2
t, v
TAMOM

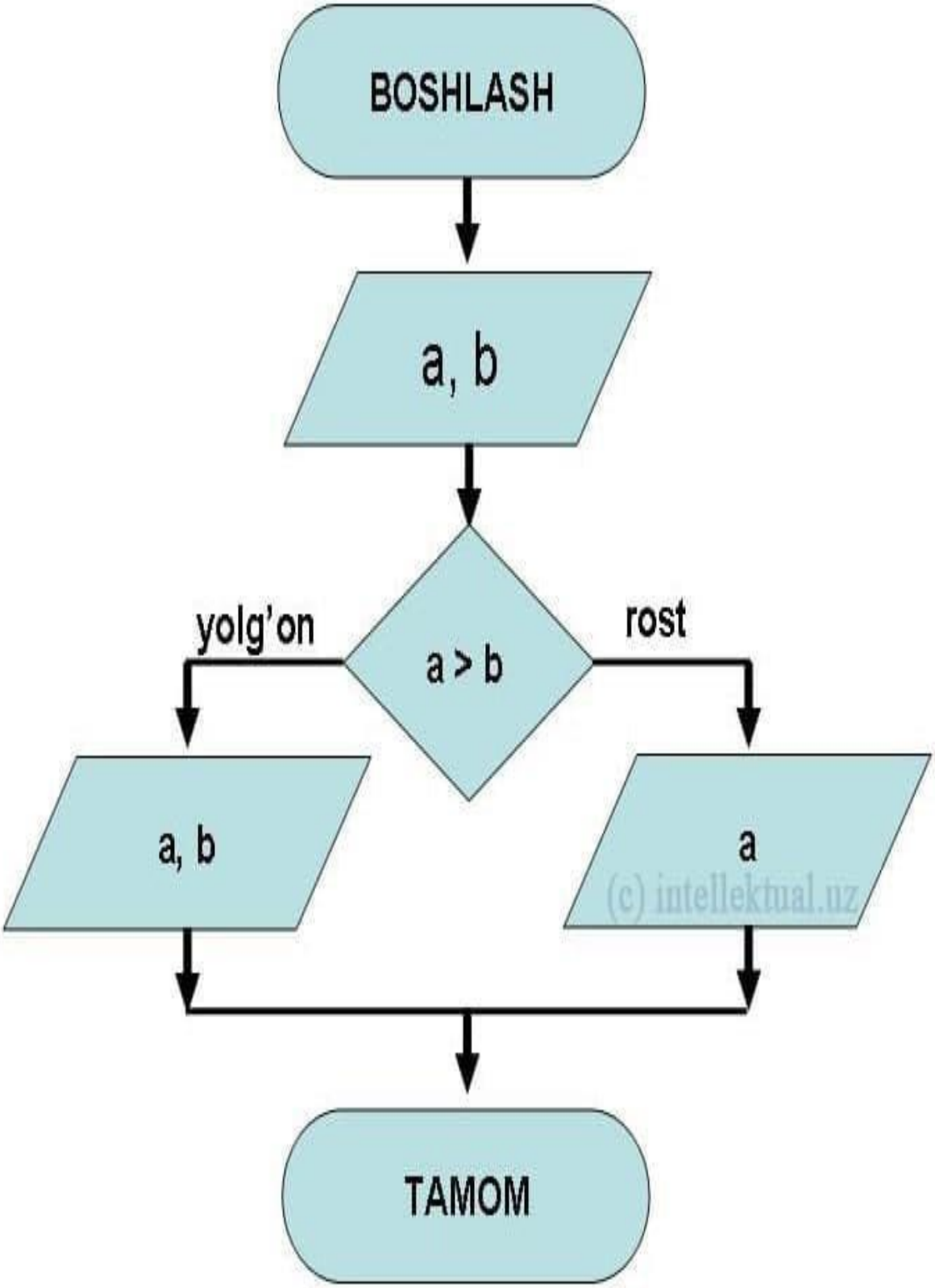

BOSHLASH
x, y, z
yolg'on
x > y
rost
max=y
max=x
yolg'on
max > z
rost
z
max
TAMOM
(c) intellektual.uz

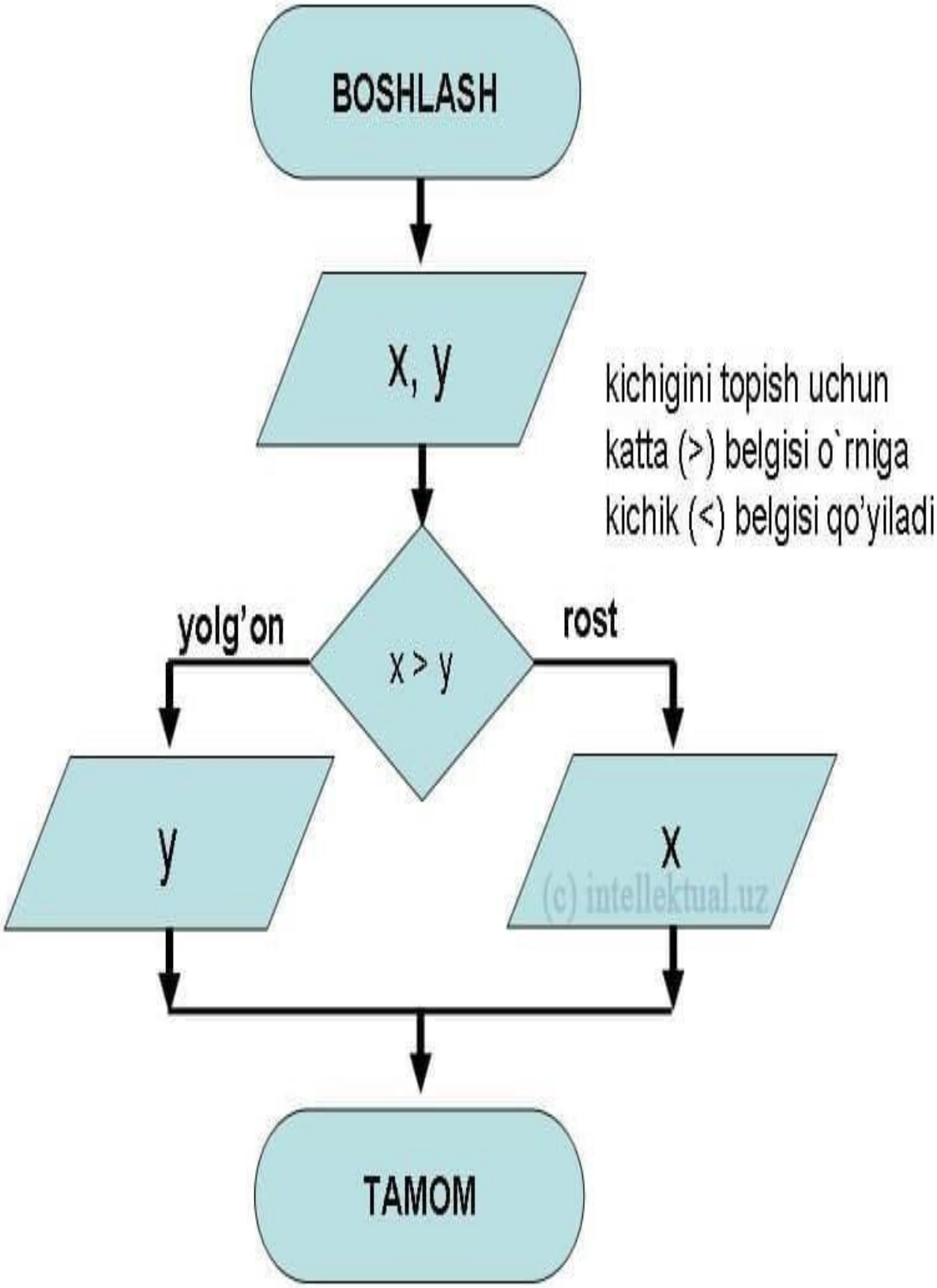

BOSHLASH
x, y, z
yolg'on
x+y+z > x*y*z
rost
max=x*y*z
max=x+y+z
(c) intellektual.uz
max
TAMOM


BOSHLASH
x, y, z
x > y
yolg'on
rost
z = y – x +1
z = x - y
(c) intellektual.uz
z
TAMOM


BOSHLASH
x, y, z
x>0
rost
yolg'on
x*x
y>0
rost
yolg'on
y*y
1
1
z>0
rost
yolg'on
z*z
TAMOM


BOSHLASH
a, b, c, d
d>=c va c>=b va b>=a
yolg'on
rost
a>b va b>c va c>d
yolg'on
rost
a = d
b = d
c = d
a=a*a; b=b*b;
c=c*c; d=d*d;
(c) intellektual.uz
a, b, c, d
TAMOM


BOSHLASH
a, b
a > b
yolg'on
rost
a, b
a
(c) intellektual.uz
TAMOM


BOSHLASH
x, y
kichigini topish uchun
katta (>) belgisi o`rniga
kichik (<) belgisi qo'yiladi
yolg'on
x > y
rost
y
x
(c) intellektual.uz
TAMOM

GLOSSARIY

Abstrakt toifalar – bu ma’lumotlar toifalarining mantiqiy xususiyatlarini aniqlashda eng foydali qurol sifatida hisobga olinadi.

Adreslar jadvalini saralash usuli – agar saralanayotgan yozuvlar xotirada

katta hajmni egallasa, u holda ularni almashtirishlar katta sarf (vaqt va xotira ma’nosida) talab qiladi. Ushbu sarfni kamaytirish maqsadida, saralash kalitlar adresi jadvalda amalga oshiriladi. Bunda faqatgina ma’lumot koʻrsatkichlari almashtirilib, massiv oʻz joyida qoladi.

Algoritm qadami – bu ketma-ket joylashtirilgan elementar va amallar toʻplami, uning bajarilish vaqtiga hamda kirish qadamiga bogʻliq emas, ya’ni yuqoridan qandaydir chegaraviy qadamlar bilan cheklangandir.

Algoritmni yozish usullari turli koʻrinishlarda boʻladi: soʻzli, formulali, jadvalli, grafikli, dasturli.

Algoritmning aniqligi–bu har bir qadam bajarilishining bir

qiymatliligidir.

Axborot – bu qayta ishlangan ma’lumotlar hisoblanadi.

Belgili toifa – bu belgilarning chekli toʻplami, ularga lotin alifbosidagi harflar va unda yoʻq kirill harflar, matematik va maxsus belgilar, oʻnlik raqamlar tashkil topgan.

Binar daraxt – bunda tugunlardan maksimal chiqish darajasi ikki boʻladi.

Binar daraxtda elementni oʻchirish – tugunni oʻchirib tashlash natijasida daraxtning tartiblanganligi buzilmasligi lozim. Tugun daraxtda oʻchirilayotganda 3xil variant boʻlishi mumkin: 1) Topilgan tugun terminal (barg). Bu holatda tugun shunchaki oʻchirib tashlanadi; 2) Topilgan tugun faqatgina bitta oʻgʻilga ega. U holda oʻgʻil ota oʻrniga joylashtiriladi; 3) Oʻchirilayotgan tugun ikkita oʻgʻilga ega. Oʻchirilayotgan tugun oʻrniga quyidagilarni biri chiqishi mumkin: yoki chap qism daraxtning eng oʻng tomondagi elementi, yoki oʻng qism daraxtning eng chap elementi.

Binar daraxtni mantiqiy tasvirlash–bunda har bir tugun to‘rtta maydonga ega yozuv ko‘rinishida ifodalandi.

Binar qidiruv – oraliqni teng ikkiga bo‘lish orqali qidiruvni amalga oshirish. Mazkur ko‘rinishdagi qidiruvni tadbiq etish uchun ma’lumotlar jadvalini tashkil etuvchi elementlar o‘sish tartibida joylashgan bo‘lishlari lozim. Faraz qilaylik, ma’lumotlar jadvali massiv ko‘rinishida berilgan bo‘lsin, u holda uning ishlash tamoyili quyidagicha massiv o‘rtasidagi element olinadi va u X qidiruv argumenti bilan taqqoslanadi, agar ular teng bo‘lsa, u holda qidiruv yakunlanadi; agar tanlangan element kaliti qidirilayotgan element kalitidan kichik bo‘lsa, u holda navbatdagi qidiruv jarayonidan tanlangan elementgacha bo‘lgan elementlar chiqarib yuboriladi va aksincha.

Birlashtirish – bu ikkita tuzilmani birlashtirish amali hisoblanadi.

Bog‘langan ro‘yxat–bunda ro‘yxatni tashkil etuvchi elementlar orasidagi bog‘liqlik ko‘rsatkichlar orqali amalga oshiriladi.

Bosh ro‘yxat–bunda ro‘yxatni tashkil etuvchi elementlar mavjud emas.

Boshlang‘ich kalit – ko‘pincha bu yerda asosiy kalit hisoblanadigan bitta maydon ma’lumoti ishlatiladigan kalit.

Butun tur–bu butun sonlar to‘plamini qandaydir qism to‘plamini ifodalab, uning qiymatlar sohasi kompyuter konfigurasiyasiga bog‘liq ravishda o‘zgarib turadi.

Chiquvchi–informatsiyaga ishlov berish natijasida dasturda hosil bo‘lgan jarayonlar. Matn, grafik,videotasvir va boshqalar ko‘rinishda bo‘lishi mumkin.

Chiziqli algoritmga–ketma-ketlikda bajariluvchi amallar va biror-bir shart tekshirilmasdan bajariluvchi algoritmlar kiradi**.**

Chiziqli ikki bog‘lamli ro‘yxat–bu elementlari soni bir xil faqatgina teskari ketma-ketlikda yozilgan ikkita bir bog‘lamli ro‘yxatdir.

Chiziqli ro‘yxat – bunda ro‘yxatni tashkil etuvchi elementlar orasidagi bog‘liqlik qat’iy tartiblangan bo‘lib, element ko‘rsatkichi o‘zidan bitta navbatdagi yoki bitta oldingi element adresini o‘z ichiga oladi.

Chiziqsiz ma’lumotlar tuzilmasi – tuzilmani tashkil etuvchi elementlar orasidagi munosabatlar ixtiyoriy bo‘lib, tuzilmani har bir elementi boshqa ixtiyoriy elementga va aksincha, har bir elementga tuzilmaning ixtiyoriy sondagi elementi murojaat qilishi mumkin.Bundan tashqari, murojaatlar og‘irlikga, ya’ni murojaatlar iyerarxik ko‘rinishga ega bo‘lishi mumkin.

Chiziqsiz ma’lumotlar tuzilmasi klassifikatsiyasi – 1) chiziqsiz bog‘langan ro‘yxatlar; 2) daraxtlar; 3) graflar.

Chiziqsiz ma’lumotlar tuzilmasini mantiqiy tasvirlash–qo‘shma matrisa yoki ko‘rsatkichli bog‘langan ro‘yxat.

Daraxt–bu chiziqsiz, iyerarxik bog‘langan ma’lumotlar tuzilmasi bo‘lib, unda shunday bitta element borki, bunga tuzilmaning boshqa elementlaridan murojaat yo‘q, mazkur element daraxt ildizi deyiladi.

Daraxt balandligi–bu daraxt bosqichlari sonidir.

Daraxt ko‘ruvi prosedurasi 1) Ildizni qayta ishlash;2)Chap tarmoq (shox)ni qayta ishlash; 3) O‘ng tarmoq (shox)ni qayta ishlash.

Daraxtdagi amallar–1)daraxt ko‘ruvi (elementlarini ma’lum bir ko‘rinishda tartiblash yoki chop etish); 2) daraxtga yangi tugun qo‘yish; 3) daraxt tugunini o‘chirish; 4) daraxt tugunini qidirish.

Daraxtga yangi element qo‘shish – bunda birinchi navbatda qo‘shmoqchi bo‘lgan yangi tugun kalit bo‘yicha daraxtda qidiruv amalga oshiriladi. Agar mazkur kalitga teng kalitli tugun mavjud bo‘lsa, u holda daraxtga tugun qo‘shilmaydi, aks holda tartiblangan binar daraxtni qurish qoidasi bo‘yicha yangi tugun qo‘shiladi.

Dasturlash – bu dastur tuzish bilan bog‘liq bo‘lgan nazariy va amaliy faoliyatdir.

Dek (DEQ - Double Ended Queue) – bu ikki chetga ega bo‘lgan navbatdir. Talabga xizmat ko‘rsatish tizimning har ikkala tomonidan amalga oshirilishi mumkin.

Dinamik ma’lumotlar tuzilmasi – dastur bajarilishi mobaynida tuzilma

elementlari orasidagi munosabatlarning o‘zgarishidir.

Diskretlilik – masalani yechish jarayonini bajarilish vaqtida kompyuter yoki insonga qiyinchilik tug‘dirmasligi uchun bir necha sodda ko‘rinishlarga bo‘lishidir.

Fayl – bu bir-biriga bog‘liq bo‘lgan yozuvlar jamlanmasidir. Masalan, bunda barcha talabalar haqidagi yozuvlarni o‘z ichiga olishi tushuniladi.

Gipergraf – bu shunday ma’lumotlar tuzilmasi bo‘lib, bunda R to‘plam ikki yoki undan ko‘proq turli tartibdagi munosabatlardan tashkil topgan bo‘ladi.

Graf–bunda R munosabatlar to‘plami faqatgina bitta binar tartibli munosabatdan tashkil topgan bo‘ladi.

Halqasimon ro‘yxat–chiziqli ro‘yxatda eng so‘nggi elementning ko‘rsatkichlari maydoniga ro‘yxatning birinchi elementi manzili o‘zlashtiriladi.

Haqiqiy tur - mazkur turga kasr qismlari bor chekli sonlar to‘plami kiradi. To‘plamni chekli bo‘lish sharti kompyuterda sonlarni ifodalash chegaralanganligi bilan bog‘liq.

Ichki kalit–bunda kalit yozuvning bir maydoni sifatida jadvalda saqlanadi.

Ideal muvozanatlangan daraxt - bunda daraxtning o‘ng va chap qism daraxtlari bosqichlari va vazni teng bo‘ladi.

Ildiz–bunday tugunga tuzilmaning hech qaysi elementidan murojaat yo‘q.

Imitatsion modellashtirish – bu asosan, sanoatda qo‘llaniladi, hisoblash texnikasi va maxsus dasturiy ta’minot yordamida real mavjud bo‘lmagan qurilmada bir qator tekshirishlar o‘tkazish imkoniyatini beradi.

Jadval yoki fayl–ixtiyoriy ma’lumotlar majmuasi.

Joriy yoki ichki–dasturning ichida informatsiyani saqlash va ishlov berish uchun ishlatiladi.

Kalit–bu elementni tasniflab bemvchi biror-bir ma’lumot bo‘lib, yozuvdagi maydon bo‘lib, aynan shu yozuvni boshqa yozuvlardan ajratib turishga xizmat qiladi, uning qiymati boshqa yozuvlarda takrorlanmas bo‘ladi.

Kalitlarni akslantirish – xeshlashtirishda ma’lumotlar jadvali oddiy massiv sifatida ifodalangan bo‘lib, bunda ma’lumotlar kalitlari massiv indekslariga akslantiriladi.

Karrali kalit – ba’zida bittadan ko‘p maydonlar qiymatlari elementlararo betakror bo‘lishidir.

Ketma-ket qidiruv – bunda ma’lumotlar butun jadval bo‘yicha operativ xotirada kichik adresdan boshlab, to katta adresgacha ketma-ket qarab chiqiladi. Mazkur ko’rinishdagi qidiruv agar ma’lumotlar tartibsiz yoki ular tuzilishi noaniq bo‘lganda qo‘llaniladi.

Ketma-ket qidiruv algoritmi samaradorligi – $M_{min} = 1$. $M_{max} = n$. Agar ma’lumotlar massiv yacheykasida bir xil ehtimollik bilan taqsimlangan bo‘lsa, u holda $M_{o'r} \sim (n + 1)/2$ bo‘ladi.

Ketma-ketlik – shunday abstrakt tuzilmaki, bunda R to‘plam faqatgina bitta chiziqli munosabatdan iborat (ya’ni, birinchi va ohirgi elementdan tashqari har bir element uchun o‘zidan oldin va keyin keladigan element mavjud.

Kiritish – tuzilmaga yangi element kiritish va chiqarish amali.

Ko‘rikdan o‘tkazish – tuzilma elementlariga bir martadan murojaat qilish amali.

Ko‘rsatkichli toifalar – ma’lumotlarning ko‘rsatkichlari yoki manzillari to‘plamini namoyon qiladi, ya’ni ko‘rsatkichlar ma’lumotlarini emas, balki bu ma’lumotlar joylashgan xotiradagi manzilni o‘z ichiga qamrab oladi.

Ko‘rsatkichli tur – bu tur o‘zgaruvchilari ma’lumotlarni ko‘rsatkichlari yoki manzillari (adres) to‘plamini namoyon qiladi, ya‘ni ko‘rsatkichlar ma’lumotlarni emas, balki bu ma’lumotlar joylashgan xotiradagi manzilni o‘z ichiga oladi.**m bog‘lamli ro‘yxat** – bunda tuzilma elementlari ko‘pi bilan tuzilmaning m ta elementi bilan bog‘langan bo‘ladi.

Ma’lumot – bu biror-bir ob’ekt, jarayon, hodisa yoki voqelikni ifodalab (tasniflab) beruvchi belgi yoki belgilar majmuasidir.

Ma’lumotlar toifasi – bu qandaydir qiymatlar yig‘indisi hisoblanib, ular

ustida ma’lum amallarni bajarish sanaladi.

Ma’lumotlar tuzilmasi – bu xotirada tashkil etiladigan elementlar yig‘indisi hisoblanib, ular ustida dastur yordamida amallar bajariladi.

Ma’lumotlar tuzilmasi elementi – bu qiymatlar to‘plamining bir bo‘lagi bo‘lib, u tuzilma elementini qiymatlar jamlanmasi hisoblanadi.

Ma’lumotlar tuzilmasini fizik tasvirlash – bunda qaralayotgan ma’lumotlar tuzilmasi kompyuter xotirasida, aniqrog‘i, operativ xotirada qanday joylashishi tushuniladi.

Ma’lumotlar tuzilmasini klassifikasiya qilish – ma’lumotlar tuzilmasini bir jinsli guruhlarga ajratish jarayoni.

Ma’lumotlar tuzilmasini mantiqiy tasvirlash – bu tuzilmani biror bir dasturlash tilida ifodalashdir.

Ma’lumotlarni standart turlari – barcha dasturlash tillarida aniqlangan, boshqa ixtiyoriy turlarning ma’lum kombinatsiyalari orqali aniqlanadi. Bular butun, haqiqiy, mantiqiy, belgili (simvol), ko‘rsatkichli turlardir.

Mantiqiy toifa – bu mazkur toifa mantiqiy mulohazalarni to‘g‘riligini aniqlash uchun turli xil dasturlash tillarida turlicha ifodalanadigan ifodalarni ikkita true (1), false (0) ko‘rinishda anglatadi.

Mashinaviy dasturlar – hisoblash mashinasi tilida (dasturlashda) ketma-ket buyruqlar ko‘rinishida berilgan masalalar yechimlarining algoritmlaridir.

Massiv – elementlari bir turga tegishli, ketma-ket joylashgan va umumiy nomga ega bo‘lgan tuzilmadir.

Matematik modellashtirish – hodisa tadqiqoti va ifodasi uchun matematik apparatni qo‘llashdan iborat. Aniq matematik model ob’ektning holatini kuzatish va uni tahlil qilish imkoniyatini beradi.

Matrisa – shunday tuzilmaki, bunda R munosabatlar to‘plami ikkita chiziqli munosabatdan tashkil topgan bo‘ladi.

Maydon – bu ob’ektlarning attributlari yoki xususiyatlarini ifodalovchi tushuncha bo‘lib, sonli yoki son bo‘lmagan qiymatlarni o‘zlashtirishi

imkoniyatini beradi.

m-chi tartibli daraxt – bunda tugunlardan maksimal chiqish darajasi m.

Model – fransuzcha so‘zdan kelib chiqqan bo‘lib, dastlabki ma’nosi bu – biror fizik ob’ekt yoki hodisaning aniq ko‘rinishini belgilab beruvchi namunadir.

Murakkablikni baholash – bu algoritmlarning murakkabligi odatda bajarilish vaqti yoki ishlatilgan xotira bo‘yicha hisoblanadi.

Muvozanatlangan daraxt – bunda daraxtning o‘ng va chap qism daraxtlari bosqichlari orasidagi farq birdan katta bo‘lmaydi.

Natijaviylik – oxirgi qadamlarda dastlabki ma’lumotlarga ega bo‘lgan kerakli natijani olishga imkon beruvchi algoritmning yakuni.

Navbat (FIFO – First input-First outpu) – bunda tizimga kelib tushgan birinchi talabga birinchi bo‘lib xizmat ko‘rsatiladi va tizimdan chiqariladi.

Noyob kalit – mazkur kalitga ega element jadvalda yagona.

Ob’ekt – bu xususiyatlar va attributlariga ega bo‘lgan va bu xususiyatlarga qiymat qabul qilishi mumkin bo‘lgan tuzilmadir. Masalan, talaba bu ob’ekt deb qaralishi mumkin.

O‘chirish – tuzilmadan bironta elementni o‘chirish yoki tozalash amali. Bunda element shunday o‘chirilishi kerakki, qolgan elementlar o‘z holatida saqlangan holatda bo‘lishi kerak, ya’ni ayrim tuzilmalarda nosozlik holatlari yuzaga kelmasligi kerak.

Ommaviy xizmat ko‘rsatish turlariga – bunga stek, navbat va deklar kiradi. Umuman olganda, ular tizimga kelib tushayotgan talablarga xizmat ko‘rsatish tartibini aniqlab beradi.

Ommaviylik – belgilangan masalalar sinfini yechish uchun algoritmni foydaliligi jarayoni.

Oraliq tugun – bunday tugun tuzilmaning kamida boshqa bitta tuguni bilan bog‘langan bo‘ladi, ya’ni mazkur tugunning ko‘rsatkichlari maydonining kamida bittasi bo‘sh emas.

Primitiv toifalar – bu ma’lumotlarning eng sodda toifalaridir. Bular

oldindan ma’lum bo‘ladigan sozlangan, toifalar bo‘lib, turli dasturlash tillarida turlicha bo‘lishi hisobga olinadi.

Qidiruv algoritmi – biror-bir ma’lumotni topish uchun ma’lumotlar bazasida elementlarni ko‘rib chiqishning qat‘iy ketma-ketligi, qidiruv vazifasi – berilgan argument bo‘yicha ma’lumotlar ichidan mazkur argumentga mos ma’lumotlarni topish yoki yo‘qligini aniqlashdan iborat.

Qidiruv tushunchasi – bu foydalanuvchi tomonidan ma’lumotlar tuzilmasi elementlarini biror-bir maqsadda ko‘rib chiqish jarayonidir.

Quiksort – mazkur usul almashtirish usulidagi saralashning modifikasiyasi bo‘lib, bunda uning asosini kalitlarni tanlangan kalitga nisbatan ajratish tashkil qiladi. Algoritm samaradorlig: O (n log n).

Rekursiv algoritm – bu algoritmni aniqlashda o‘ziga bevosita yoki bilvosita murojaat qilishdir.

Rekursiv ma’lumotlar tuzilmasi – bunda tuzilmani tashkil etuvchi elementlar ham mazkur tuzilmaga o‘xshash tuzilma bo‘ladi.

Rekursiya – bu shunday jarayonki, unda tadqiq qilinayotgan jarayonni aniqlash mazkur jarayonga murojaat qilish orqali amalga oshiriladi.

Ro‘yxat uzunligi – bu ro‘yxatni tashkil etuvchi elementlari soni. Umumiy holda ro‘yxat elementlari soni chegaralanmagan va dastur bajarilishi mobaynida o‘zgarib turishi mumkin.

Ro‘yxatga murojaat – ro‘yxatga murojaat faqat ro‘yxat boshidan amalga oshiriladi. Teskari aloqa yo‘q.

Saralash – bu berilgan to‘plam elementlarini biror-bir tartibda joylashtirish jarayonidir.

Saralash maqsadi – tartiblangan to‘plamda kerakli elementni topishni osonlashtirishdan iborat.

Saralash samaradorligi (taqqoslashlarga nisbatan) – O(n log n) dan O(n2) gacha; O(n) - ideal holatda.

Saralash samaradorligi mezonlari – saralashga ketgan vaqt: saralash

uchun talab qilingan operativ xotira dasturi ishlab chiqishga ketgan vaqt.

Saralash turlari – 1). ichki saralash bu operativ xotiradagi saralash va 2). tashqi saralash - tashqi xotirada saralash.

Shell saralashi – bu qisqarib boruvchi qadamlar orqali saralashga kiradi.

Sikllik algoritmga – alohida jarayonlar yoki jarayonlar guruhi bir necha marta bajariladigan algoritm kiradi.

So‘z – bir vaqtning o‘zida qayta ishlanishi mumkin bo‘lgan minimal sondagi bit.

Statik ma’lumotlar tuzilmasi – bu dastur bajarilishi mobaynida tuzilmani tashkil etuvchi elementlar orasidagi munosabatlar o‘zgarmsdan qoladi.

Stek (LIFO – Last input – First output) – bunda tizimga kelib tushgan oxirgi talabga birinchi bo‘lib xizmat ko‘rsatiladi va tizimdan chiqariladi.

Tarkibli kalit – ba’zida esa yozuvlarning yagona qiymatli kalit maydonni yo‘qligi sababli kalit sifatida bir nechta maydonlar hisobga olinadigan kalit.

Tarmoqlanuvchi algoritmga – belgilangan shartlarning o‘zgarishiga bog‘liq holda ko‘rsatmalarning variantlari oldindan mo‘ljallanadigan algoritmlar kiradi.

Tartiblangan binar daraxtni qurish – bunda otaga tugunga nisbatan chap tomondagi o‘g‘il qiymati kichik kalitga, o‘ng tomondagi o‘g‘il esa katta qiymatli kalitga ega bo‘ladi, ya‘ni **key(left_son)< key(father)<key(right_son).**

Tashqi kalit – bunda kalitlar ma’lumotlar jadvalidan ajratib olinib alohida fayl sifatida saqlanadi.

Terminal (barg) – mazkur tugun tuzilmaning boshqa hech qanday tuguni bilan bog‘langan emas, ya’ni bunday tugunning barcha ko‘rsatkichlar maydoni bo‘sh bo‘ladi.

To‘g‘ridan-to‘g‘ri tanlash usuli bilan saralash – bunda berilgan elementlar

ichidan eng kichik kalitga ega element tanlanadi va ushbu element boshlang‘ich ketma-ketlikdagi birinchi element bilan o‘rin almashadi. Undan keyin ushbu

jarayon qolgan n-1 ta element, n-2 ta element va xokazo, toki bitta eng “katta” element qolguncha davom ettiriladi.

To‘liq binar daraxt – bunda har bir tugundan chiqish darajasi 0 yoki 2 bo‘ladi.

To‘liq m-chi tartibli daraxt – bunda har bir tugundan chiqish darajasi 0 yoki m bo‘ladi.

To‘plam – bu bir xil toifadagi elementlarning tartibsiz joylashuvidir va unda elementlar takrorlanmas ko‘rinishda bo‘ladi. To‘plam ustida bajariladigan amallar quyidagi ko‘rinishlarni oladi: element o‘chirish, kiritish, elementlar sonini aniqlash, bo‘shliqqa tekshirishdan iborat.

To‘plam ko‘rinishidagi ma’lumotlar tuzilmasi – bu shunday tuzilmaki, uning elementlari orasida hech qanday munosabat o‘rnatilmagan. Bundan tashqari to‘plam elementlari bir turga tegishli va ular takrorlanmaydi.

To‘r (set) – bunda tuzilmaning barcha elementlari bir-biri bilan bog‘langan bo‘ladi.

Translyatsiya – algoritmni mashina tiliga o‘girish jarayoni.

Tugundan chiqish darajasi – bu daraxt tugunlaridan chiqayotgan shoxlar soni.

Tushunarlilik – ijrochiga tavsiya etilayotgan ko‘rsatmalar uning uchun tushunarli bo‘lishi kerak, aks holda ijrochi oddiy amalni ham bajara olmay qoladi.

Tuzilma elementlarini utilizatsiya qilish – tuzilmani keraksiz elementlardan
tozalash.Mazkur ishni ikki xil yo‘l bilan amalga oshirish mumkin: hisoblagichlar (o‘chetchiklar) va markerlar usuh.

Utilizasiya – keraksiz, ortiqcha elementlami tozalash, yo‘qotish va hokazolar.

Vaqt bo‘yicha qiyinligi – algoritmda sarflanayotgan uzoq vaqt, masalaning o‘lchami. Bunday funksiyada masalaning hajmi oshganda limit

ostidagi o‘zgarish yuqori bo‘ladi, bu esa asimptotik qiyinlik deb aytiladi.

Vektor – bu eng sodda statik va chiziqli tartiblangan tuzilma bo‘lib, uning elementlari bir turga tegishli hamda operativ xotirada ketma-ket kelgan yacheykalarda joylashadi.

Xesh jadvali – bu assotsiativ massiv interfeysini amalga oshiruvchi ma’lumotlar tuzilmasi bo‘lib, ya’ni juftlarni saqlashga va uchta amalni bajarishga imkon beradi: yangi juftlikni qo‘shish, qidirish amali hamda juftlikni kalit bilan o‘chirishda foydalaniladi.

Xeshlash – bu ixtiyoriy uzunlikdagi kirish ma’lumotlari majmuasini ma’lum bir algoritm tomonidan bajarilgan, belgilangan o‘lchamdagi chiqish massiviga aylantirish jarayoni bo‘lib xizmat qiladi. Bunday algoritmni amalga oshiruvchi funksiya xesh funksiya yoki transformatsiya natijasi xesh qolaversa xesh yig‘indisi deb yuritiladi.

Yo‘naltirilgan graf (orgraf) – G=(V,E) juftlikka yo‘naltirilgan graf (orgraf) deyiladi, bunda V – tugunlar (elementlar) to‘plami, E - esa yoylar (yo’naltirilgan yoqlar), aniqrog‘i tartiblangan munosabatlar to‘plami, ya’ni (v,ω#(ω,v).

Yozuv – maydon deb ataluvchi chekli sondagi ma’lumotlar tuzilmasidir. Turli maydon o‘zgaruvchilari turli turga tegishli bo‘ladi.

FOYDALANILGAN ADABIYOTLAR RO‘YXATI

I. O‘zbekiston Respublikasi qonunlari

1. Ўзбекистон Республикасининг «Таълим тўғрисида»ги ЎРҚ-637-сон Қонуни. 23.09.2020. https://lex.uz/docs/5013007

2. Ўзбекистон Республикасининг «Илм-фан ва илмий фаолият тўғрисида»ги ЎРҚ-576-сон Қонуни 29.10.2019. https://lex.uz/docs/4571490

II. O‘zbekiston Respublikasi Prezidenti farmonlari va qarorlari

1. Мирзиёев Ш.М. Буюк келажагимизни мард ва олижаноб халқимиз билан бирга қурамиз. Тошкент: «Ўзбекистон» НМИУ, 2017. -488 б.

2. Мирзиёев Ш.М. Қонун устуворлиги ва инсон манфаатларини таъминлаш - юрт тараққиёти ва халқ фаровонлигининг гарови. -Тошкент: «Ўзбекистон» 2016. -50 б.

3. Мирзиёев Ш.М. Эркин ва фаровон, демократик Ўзбекистон давлатини биргаликда барпо этамиз. -Тошкент: «Ўзбекистон» 2016. -59 б.

III. Darslik, o‘quv qo‘llanma va dissertatsiyalar

1. Eshtemirov S., Nazarov F.M. Algorimlash va dasturlash asoslari. O‘quv qo‘llanma. SamDU. 2019 y. –208 b.

2. Baratov R.J. Algoritmlar va programmalash. O‘quv qo‘llanma Toshkent. TIQXMMI. 2018 –302 b.

3. Sharipov B.A., Tashpulatova N.B., Ganixodjayeva D.Z. Ma`lumotlar tuzulmasi va algoritmlar fanidan laboratoriya ishlarini bajarish bo‘yicha uslubiy qo‘llanma. Toshkent. 2022. –100 b.

4. A.M.Polatov Algoritmlar va C++ tilida dasturlash asoslari. O‘quv qo‘llanma / Toshkent. Toshkent universiteti. 2017. – 123 b.

5. A. Sattorov Ma’lumotlar bazasini boshqarish. Darslik. T., Fan va texnologiya. 2006. –304 bet.

6. Хаггарти Р. Дискретная математика для программистов Издание 2-е, исправленное. Москва: Техносфера, 2012. – 400 с.

7. Shukla, Rajesh K. Data Structures Using C and C++ : monograph - New Delhi : Wiley India, 2012. - 502 p.

8. Kruse, Robert L. Data Structures and Program Design in C : monograph. - New Delhi: Dorling Kindersley (India) Pvt. Ltd, 2012. - 607 p.

9. Роберт Седжвик Фундаментальные алгоритмы на C++. Анализ. Структуры данных. Сортировка. Поиск // Перю с англ. Робеот Седжвик. – К.: Изд. «ДиаСофт», 2001. – 688 с.

10. Т.Кормен., Ч.Лейзерсон, Р.Ривест, К.Штайн Алгоритмы: построение и анализ. 3-е изд. / – М.: «И.Д. Вильямс», 2013. – 1328 с.

11. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона +СД / пер. с англ. Ткачев Ф.Б. – М.: ДМК Пресс. 2010. – 272 с.

12. Даступа С., Пападимтриу Х., Вазирани У. Алгоритмы. – М.: МЦ-НМО, 2014 -320 с.

13. Кнут Д.Э. Искусство программирования. Том 1. Основные алгоритмы. М.: Вильямс, 2010. – 720 с

14. Кнут Д.Э. Искусство программирования. Том 2. Получисленные алгоритмы. М.: Вильямс, 2011. – 832 с

15. Кнут Д.Э. Искусство программирования. Том 3. Сортировка и поиск. - М.: Вильямс, 2012. – 824 с

16. Литвиненко Н. А. Технология программирования на C++ Win32 API-приложения: учеб. пособие для студ. вузов. – СПб.: БХВ – Петербург, 2010. – 280 с.

17. Шилдт Г. Самоучитель C++. – 3-е изд. – СПб.: БХВ – Петербург, 2002. – 151 с.

18. Томас Х.Кормен Алгоритмы. Вводный курс. Диалектика-Вильямс. 2020 г. – 208 с.

19. Уоррен Генри С. Алгоритмические трюки для программистов.: Пер. с англ. – М.: Издательский дом. “Вильямс”, 2007. – 288с.

20. Adam Drozdek. Data structures and algorithms in C++. Fourth edition. Cengage Learning. 2013. – 818 p.

21. Alishev Sh.A. Ko’p bosqichli o’zgaruvchan parametrli ob’ektlarni adaptiv boshqarish algoritmlari. Texnika fanlari bo’yicha falsafa doktori (PhD) dissertatsiyasi. Toshkent. 2023 y. 120 b.

22. Eied Mahmoud Mohammed Xalil Chekli va cheksiz o’lchamli chigal holatlarni matematik modellashtirish va ularni kvant ma’lumotlarida qo’llash. ilmiy ma’ruza shaklidagi fizika-matematika fanlari doktori (DSc) dissertatsiyasi. Toshkent. 2021 y. 97 b.

23. Djafarova N.A. Turizmda statistik ma’lumotlarni yig’ish metodologiyasini xalqaro standartlarga moslashtirish yo’llari. Texnika fanlari bo’yicha falsafa doktori (PhD) dissertatsiyasi. Buxoro. 2017 y. 106 b.

24. Djamilov S.S. Texnik fanlar bo’yicha bilimlarni nazorat qilishning kompyuter testlarini avtomatlashtirilgan qurish usuli va algoritmlari. MSc darajasini olish uchun dissertatsiya. Buxoro. 2023 y. 98 b.

25. Пасечников К.А. Синтез оптимальных структур данных для алгоритмов решения комбинаторных задач на графах. Кандидат технических наук. Москва. 2009 г. 162 с.

IV. Internet saytlari

1. www.edu.uz – Axborot ta’lim portali;

2. www.pedagog.uz – Malaka oshirish muassasalari sayti;

3. www.school.edu.ru – Umum ta’lim portali;

4. www.ziyonet.uz – Ta’lim portali;

5. www.allbest.ru - Internet resurslari elektron kutubxonasi;

6. www.schulen-ans-netz.de – Germaniya “Internet-Maktab” sayti;

7. www.educasource.education.fr – Fransiya ta’lim sayti;

8. https://www.intuit.ru/studies/courses/40/40/info – ИНТУИТ milliy ochiq universitet

MUNDARIJA

O‘QUV USLUBIY NASHR

ABLAQULOV KAMOLIDDIN BAHRIDDINOVICH

MA’LUMOTLAR TUZILMASI VA ALGORITMLARI

60610500 - Kompyuter injiniringi bakalavriat yo‘nalishi uchun

O‘QUV QO‘LLANMA

Nashriyot litsenziyasi 062153, 07.02.2023 yil.
Terishga 20.05.2025 yilda berildi.
Bosishga 16.06.2025 yilda ruxsat etildi.
Bichimi 64x84 1/16. Shartli bosma tabog‘i 16.25.
Ofset usulida chop etildi. Times New Roman garniturasi.
50 nusxada. Buyurtma 090-BMW/25. Hajmi 260 bet. Erkin narxda.

“BIG MAKRO WORLD” MCHJ nashriyoti, 180100.
Qarshi shahri, Ko‘chabog‘ ko‘chasi, 17-uy

“BIG MAKRO WORLD” MCHJ bosmaxonasida bosildi.
Qarshi shahri, Beglar MFY, Ko‘chabog‘ ko‘chasi, 2/49-uy
Tel.: +998908680001