Otaxanov Nurillo Abdumalikovich

ALGORITM QURISH METODLARI

Otaxanov Nurillo Abdumalikovich

ALGORITM QURISH METODLARI

Namangan-2019

***UO'K:** 821.512.133.9*
***KBK:** 84 (5 Ўзб) 7*
O-18

Ushbu monografiya dastur ishlab chiqish uchun asos bo'lib xizmat qiladgan algortimlar va ularning qurush usullariga bag'ishlangan. Unda yangi dasturiy ta'minot ishlab chiqish uchun mo'ljallangan algoritm qurishning bir qator usullari va bu usullarning samaradorlik darajalari bayon etilgan. Keltirilgan har bir algoritm qurish usulini konkret misollar yordamida amaliyotga tatbiq etish yuzasidan tavsiyalar ishlab chiqilgan.

Monografiya informatika va axborot texnologiyalari bo'yicha barcha yo'nalish bakalavr va magistrlari xamda dasturchilik bilan qiziqqan kitobxonlar uchun mo'ljallangan.

Taqrzchilar:

1. Jakbarov O. O. – Namangan Qurilish instituti informatika kafedrasi dotsenti, t.f.n.
2. Boltaboyev Sh. – Namangan davlat universiteti Amaliy matematika kafedrasi katta o'qituvchisi, t.f.n.

Ilmiy maslahatchi:

Imomov A.– Namangan davlat universiteti Amaliy matematika kafedrasi dotsenti, f.-m.f.n.

NamDU Ilmiy texnikaviy kengashining 2019 yil 10 iyundagi yig'ilishining 6- sonli qarori bilan chop qilishga ruxsat etildi.

ISBN 978-9943-5644-5-9

SO'ZBOSHI

Insoniyat mavjud ekan, doimo o'z oldiga Nima? Qachon? Qaerda? Qanday? degan savollar qo'yadi va javob izlaydi. Aynan shu savollar dunyoni bilishga, uning sabab va mohiyatini o'rganishga majbur qiladi. Natijada fan va hayotga, kishilik jamiyatiga taalluqli bo'lgan masalalar uchun turli hil algoritmlar ishlab chiqiladi.

Eng sodda jonivorlardan boshlab, to eng mukammal hisoblangan insongacha bo'lgan hayot algoritmlar asosida ko'payadi, yashaydi va nobud bo'ladi. Algoritmlar fan va jamiyatning rivojlanishiga sabab bo'ladi. Ayniqsa, bugungi kungi hayotni telefon, kompyuter, datchik, stanok, avtomobil kabi ko'plab moslama va qurilmalarsiz tasavvur qilishning umuman iloji yo'q. Fan va texnikaning bunday yutuqlaridan foydalanish uchun ma`lum bir algoritmlarni bilish lozim bo'lib qoldi. Masalan, uyali telefon apparatidan foydalanish uchun uning sensorli ekranida abonent barmoqlari bilan ma`lum bar xarakatlarni belgilangan tartibda bajarishi kerak.

Algoritmlarning ahamiyati nimada? Nima uchun ularni o'rganish lozim? Gap shundaki, algoritmlar ikki vazifani bajaradi: amaliy va nazariy. Amaliy ahamiyati shundaki, turloi sohalarga oid masalalarni hal qilish uchun kishilar maxsus ishlab chiqilgan algoritmlarni bilishi kerak. Bundan tashqari, zarur xollarda yangilarini ishlab chiqishga yoki mavjudlarini tahlil qilib, ulardan eng maqbulini tanlashga tayyor bo'lishi lozim. Nazariy tomondan, algoritmlar hech bir istisnosiz, barcha fanlarning rivojlanishi uchun mustahkam poydevor bo'la oladi. Shuningdek, ular informatika fanining mustaqil bir bo'lagi bo'lgan algoritmikaning shakllanish va rivojlanishiga asos bo'ldi.

Algoritmika – informatika fanining asosi hisoblanadi, va aytish mumkinki, u zamonaviy fanlar, texnika va biznesning rivojlanishiga ulkan hissa qo'shadi.

Ko'pchilik, algoritmlar faqat dasturchilarga hos deb hisobalaydi.

Aslida, inson kim va qaysi soha mutaxassisi bo'lishidan qat`iy nazar algoritmlarni o'rganishi uchun yetarli sabablar mavjud. Bugungi kunda insoniyatni ikki qismga ajratish mumkin: ishlab chiquvchilar va foydalanuvchilar. Ishlab chiquvchilar yangi-yangi g'oyalarni ilgari suradilar, ularni hal qilish uchun turli algoritmlar o'ylab topadi va ular asosida yangi dasturlar, texnik qurilma va moslamalar ishlab chiqadilar. Foydala- nuvchilar esa fan va texnikaning bu yutuqlaridan foydalanish imkoniyatiga ega bo'lish uchun ma`lum bir algoritmlarni bilishlari zarur.

Algoritmlarni o'rganishning yana bir sababi talabalarning mantiqiy fikrlash qobiliyatlarini rivojlantiradi. Ma`lumki, kishilik hayotida mantiqiy fikrlash katta ahamiyat kasb etadi va insonning garmonik rivojlanishida uchun muhim poydevor bo'lib xizmat qiladi. Internet tarmoqlari orqali olimlarning bong urishicha, zamonaviy inson hayotini qulaylashtirish uchun shunchalik ko'p qurilmalar ishlab chiqildiki, kishilarda fikrlashga ehtiyoj borgan sari kamayib, deyarli hamma ishni miyada fikrlab emas, balki to'g'ridan – to'g'ri barmoqlarni xarakatlanish yoki ma`lum bir tugmalarni bosish bilan cheklanib qolmoqda. Oqibatda insonlarning aqliy qobiliyati pasayib bormoqda. Mantiqiy fikrlash esa ana shu miyani "yana ishlashga majbur qilishi" bilan ahamiyatli hisoblanadi. Bu borada atoqli olim D. Knut shunday yozadi: "Yaxshi mutaxassislar yangi algoritmlarni qanday ishlab chiqish, mavjudlari o'zgartirish, tushunish va tahlil qilishni bilishlari shart. Bu bilimlar ... universal fikrlash apparati uchun muhim poydevor bo'ladi va boshqa fanlar asoslarini o'rganishda bebaho vosita bo'lib xizmat qiladi[1]". Ko'rinib turibdiki, inson kim bo'lishidan qat'iy nazar, algoritmlarni o'rganishi han foydali, ham muhim shart hisoblanadi.

---

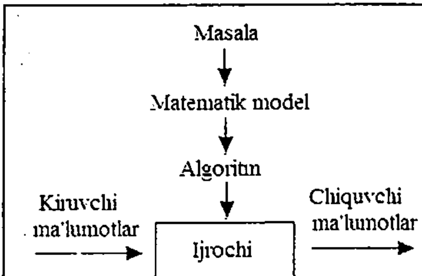
[1] Д. Кнут. Искуссво программирования. 1-том. Основные алгоритмы . 3-изд. М.: Вильямс. 2000. –стр. 9.

§-1. ALGORITM TUSHUNCHASI HAQIDA

Algoritm tushunchasi uchun bir qator ta'riflar mavjud bo'lib, bu borada olimlar hozircha yagona fikrda to'xtalganlaricha yo'q. Shunday bo'lsada, bizning fikqimizga ko'ra, quyidagi ta'rif haqiqatga yaqinroq ko'rinadi.

Ta'rif: Algoritm deb ijrochiga tushunarli bo'lgan va qo'yilgan masalani yechish uchun bajarishi lozim bo'lgan qat`iy va aniq ko'rsatilgan buyruqlar ketma-ketligiga aytiladi.

Boshqacha aytganda, algoritm bu to'g'ri kiritilgan boshlang'ich ma`lumotlar uchun ma`lum bir vaqt mobaynida qo'yilgan masalaning kutilgan yechimlarini (chiquvchi ma`lumotlarni) ta`minlab bera oladigan buyruqlar ketma-ketligidan iborat. Ularni amaliyotda qo'llasg sxemasi 1.1-sxemada keltirilgan.



1.1-rasm. Algoritmdan foydalanish sxemasi

Qandaydir masalaning yechimini topish uchun unga mos qurilgan algoritm ijrochi tomonidan bajarilishi lozim. Umuman olganda, inson, jonivor yoki texnik qurilma algoritmning ijrochisi bo'lishi mumkin. Ammo, bugungi kunda ijrochi sifatida kompyuterlar tan olinadi.

Har qanday algoritm quyidagi hususiyatlarga ega bo'lishi lozim:

1. har bir buyrug'i bir qiymatli va aniq ifodalangan bo'lishi lozim;

2. algoritm tomonidan qayta ishlanishi talab qilingan kiruvchi va chiquvchi ma`lumotlar diapazoni aniq ko'rsatiladi.

3. algoritm ko'rsatmalari ko'rsatilgan tartibda qat'iy bajariadi;

4. bitta algoritm yordamida shu masalaga o'xshash ko'plab

masalalarni yechish mumkin bo'ladi.

Yangi dasturchilarni tayyorlashda dasturlash tillari bilan bir qatorda masalalar uchun yangi algoritmlarni qurish yoki mavjudlarini tahlil qilishga o'rgatish eng muhim bosqichlardan biri hisoblanadi.

Masala-1. Berilgan ***a*** va ***b*** natural sonlari uchun eng katta umumiy bo'luvchini toping.

Yechish. Izlanayotgan sonni *ekub(n, m)* orqali belgilaylik. Qo'yilgan masalani yechish algoritmini eramizdan avvalgi III asrda qadimgi grek olimi Yevklid taklif etgan usuldan foydalanamiz:

$$ekub(n, m) = ekub(n, n \bmod m).$$

Masalan, (46. 69) sonlar juftligi uchun *ekub* quyidagicha hisoblanadi:

$$ekub(46,\ 69) = ekub(23,\ 46) = ekub(23,\ 0) = 23.$$

Qaralayotgan masala algoritmini quyidagicha yozish mumkin.

1. Agar ***n*** = 0 bo'lsa, u xolda yechim sifatida *m* ni oling va ishni tugating; aks xolda 2 qadamga o'ting;
2. ***m*** ni ***n*** ga bo'ling va qoldig'ini ***r*** ga yozing;
3. ***n*** o'zgaruvchiga ***m*** ning, ***m*** ga esa ***r*** qiymatini bering; 1 qadamga o'ting.

Mazkur algoritmga mos buyruqlarni C++ dasturlash tilida quyidagicha yozish mumkin:

```
while (n!=0)
{r=n % m; n=m; m=r;}
```

Qachondir Yevklid algoritmi tugaydimi? Ha, albatta. Chunki, algoritm tarkibidagi tsikl xar gal takrorlanganda ***n*** ning qiymati kamayadi va uning avvalgi qiymatidan kichik bo'ladi va qachondir u 0 ga teng bo'lib qoladi.

Shu masalaning yana bir algoritmini ko'raylik. Unda *ekub* ni tanlash yordamida aniqlanadi. Ma`lumki, *ekub* berilgan sonlarning kichigidan katta bo'la olmaydi. Shuning uchun ularning kattasini kichigiga bo'linadi. Agar bo'linmasa, u xolda kichik son 1 ga kamaytiriladi va jarayon yana takrorlanadi. Masalan, yuqoridagi (46, 69) sonlari uchun 69 dastlab 46 ga, so'ngra 45 va hokazo sonlarga bo'linadi. Jarayon 23 ga kelganda to'xtaydi. Ushbu g'oyaga mos keluvchi algooitm quyidagicha yoziladi:

1. *min*(*n*, *m*) ni aniqlab, *t* ga yozing;

2. ***m*** ni *t* ga bo'ling. Agar qoldiq 0 ga teng bo'lsa 3 ga, aks xolda 4 qadamga o'ting;

3. ***n*** ni *t* ga bo'ling. Agar qoldiq 0 ga teng bo'lsa *t* javob sifatida qabul qiling va ishni tugating; aks xolda 4-qadamga o'ting;

4. *t* dan 1 ni ayiring va 2 –qadamga o'ting.

Ko'rinib turibdiki, bu algoritm berilgan sonlardan biri 0 ga teng bo'lganda natija bermaydi.

Ekub topishning yana bir usuli maktab algebra kursidan ma`lum:

1. ***m*** sonini tub ko'paytuvchilarga ajrating;

2. ***n*** sonini tub ko'paytuvchilarga ajrating;

3. 1 va 2-qadamda topilgan umumiy bo'luvchilarni ajrating;

4. Barcha ajratilgan umumiy bo'luvchilarni ko'paytiring va ko'paytmani *ekub* sifatida qabul qiling.

Masalan, (48, 72) sonlari uchun ye*kub* quyidagicha hisoblanadi:

$48=2\cdot2\cdot2\cdot2\cdot3$; $72=2\cdot2\cdot2\cdot3\cdot3$; *ekub*$(48, 72)=2\cdot2\cdot2\cdot3$.

Agar bitta masala uchun bir nechta algortim mavjud bo'lsa, u xolda ularning qaysi biridan foydalanish lozim? degan savol paydo bo'ladi. Biz bu savolga keyinroq javob beramiz.

Masalalarni oxirgi usul bilan hal qilishda yangi bir muammo, ya`ni tub sonlarni aniqlash masalasi uchrab qoldi.

Masala-2. Berilgan ***N*** gacha bo'lgan tub sonlarni aniqlang.

Yehish. Masalani Eratosfen g'alviri yordamida hal qilishga urinib ko'ramiz. Bu usulda 2 dan boshlab berilgan *N* sonigacha bo'lgan barcha natural sonlar ro'yxati yozib chiqiladi. Ro'yxatni bir necha marta qarab chiqiladi. Ro'yxatda birinchi o'chmagan son 2 ni qoldirib, unga karrala bo'lgan barcha sonlar o'chiriladi. So'ngra, navbatdagi o'chmagan son 3 ni qoldirib, unga karrali bo'lgan sonlar ro'yxatdan o'chiriladi. Jarayon ro'yxatda o'chirilishi mumkin sonlar qolmaguncha davom ettiriladi va o'chmay qolgan sonlarni tub sonlar sifatida qabul qilinadi. Masalan, 16 gacha bo'lgan tub sonlarni aniqlash talab qilingan bo'lin. U xolda 1 va 2 – marta o'tishda mos ravishda 2 va 3 ga karrali sonlar o'chiriladi:

1-chi o'tish: 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~
2-chi o'tish: 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~

Navbatdagi 3-o'tishda 5 ga karrali 10 va 15 sonlari, 4-marta o'tishda esa 7 ga karrali 14 soni takroran o'chiriladi. Ro'yxatda o'chmay qolgan 2, 3, 5, 7, 11 va 13 sonlari izlangan tub sonlarni beradi.

Mazkur jarayonni quyidagi algoritm orqali ifodalash mumkin:

1. 2 dan *N* gacha sonlar ro'yxatini yozing;
2. ***k*** ga 1 qiymat bering;
3. Agar ***k*** - chi son o'chirilgan bo'lsa 5-qadamga o'ting;
4. ***k*** - sonni qoldiring va unga karrali bo'lgan boshqa sonlarni ro'yxatdan o'chiring;
5. ***k*** ni qiymatini 1 ga orttiring;
6. agar ***k***<*N* bo'lsa 3 – qadamga, aks xolda 7 - qadamga o'ting;
7. o'chmay qolgan sonlarni aniqlang va ularni yechim sifatida qabul qiling.

Dastur ishlab chiqish amaliyotida ko'pincha u yoki bu natural son bo'luvchilarining mavjud yoki mavjud emasligini aniqlashga to'g'ri keladi.

Masala-3. Berilgan *N* sonining (*N*>2) tubligini aniqlang.

Yechish. Berilgan son tub bo'lsin deb faraz qilaylik. Ma`lumki, ihtiyoriy N natural sonining eng katta bo'luvchisi $\sqrt{N}$ dan katta bo'la olmaydi. Shuning uchun bo'luvchilarni [2, $\sqrt{N}$] oraliqdan izlaymiz. Agar N soni bu oraliqdagi biror songa bo'linsa, u xolda berilgan son murakkab bo'ladi. Aks xolda faraz o'z kuchini saqlaydi.

Berilgan masala uchun algoritmni quyidagicha qurish mumkin:

1. y o'zgaruvchiga "tub" qiymatini bering;
2. k o'zgaruvchiga $\sqrt{N}$ qiymatini bering;
3. t o'zgaruvchiga 2 qiymatini bering;
4. agar $t>k$ bo'lsa, 8 - qadamga o'ting;
5. Agar N soni t ga qoldiqsiz bo'linsa, y o'zgaruvchiga "murakkab" qiymatini bering va 8 - qadamga o'ting;
6. t ning qiymatini birga orttiring;
7. 3-chi qadamga o`ting;
8. y ni yechim deb qabul qiling.

Yuqorida qaralgan masalalar sof matematik bo'masada, dasturlash amaliyotida tez-tez uchrab turadi.

Yuqori malakali dasturchi bo'lish uchun o'z ustida ko'p ishlash, yangi masalalar uchun algoritmlar qurishni xamda ularni tahlil qilishni o'rganish talab qilinadi.

§-2. ALGORITM QURISH ASOSLARI

Algoritm qurishda asos bo'lib masalaning yechimlari emas, balki bu yechimlarni ta`minlay oladigan va aniq ifodalangan ko'rsatma-buyruqlar ketma-ketligi xizmat qiladi.

Algoritmlarni loyihalash va tahlil qilish quyidagi ketma-ketlikda amalga oshiriladi:

1. Masalani tushunish;

2. Kompyuter imkoniyatlarini aniqlash;
3. Aniq yoki taqribiy yechish usulini tanlash;
4. Ma`lumotlar uchun mos tuzilmalarni tanlash;
5. Loyihalash metodlarini tanlash;
6. Ifodalash usullarini tanlash;
7. Algoritm korrektligini (to'g'ri ishlashini) baholash;

Algoritmlarni tahlil qilish.

1. Masalani tushunish. Matematiklarda "masala shartini to'g'ri tushunish" yechimni 50% ga topish" degan gap bor. Shuning uchun biror masalaga algoritm qurishdan avval uning shartini diqqat bilan o'qib chiqish, ochiq qolgan savollarning bor-yo'qligini aniqlash, zarur bo'lsa, bir necha oddiy namunalar yordamida tahlil qilish lozim. Masalaning hususiy xollarini o'rganib chiqish ham algoritm qurishda katta yordam berishii mumkin.

Bugungi kunda katta sondagi tipik masalalar uchun algoritmlar ishlab chiqilgan. Agar masala shulardan biriga o'xshasa, u xolda tayyor algoritmdan foydalanish mumkin.

Algoritm uchun boshlang'ich ma`lumotlar masalaning alohida bir nusxasini hosil qiladi. Bunda algoritm uchun mumkin bo'lgan ma`lumot- lar diapazonini aniq ko`rsatish muhim sanaladi. Chunki, algoritm ko'plab boshlang'ich ma`lumotlar uchun to'g'ri natija berishi mumkin, ammo, "kritik" deb ataluvchi boshqa ma`lumotlar uchun to'g'ri ishlamasligi mumkin. Bu o'rinda, faqat ko'plab boshlang'ich ma`lumotlar uchungina emas, balki har qanday boshlang'ich ma`lumotlar uchun to'g'ri natijani kafolatlaydigan algoritmlarni to'g'ri (korrekt) algoritm deb qabul qilinishini nazarda tutish lozim.

2. **Kompyuter imkoniyatlarini aniqlash.** Algoritmni qurish jarayonida uning ijrochisi bo'lgan kompyuter imkoniyatlarini ham baholashga to'g'ri keladi. Kompyuter imkoniyatlarini aniqlashda quyidagi holatlarga e`tibor berish zarur:

a) buyruqlarni bajarish rejimiga. Bunda bir xil toifadagi kompyu- terlar muayyan bir vaqtda faqat bitta buyruqni bajarsa, boshqalari shu vaqt mobaynida parallel ravishda bir necha buyruqlarni bajarishi mumkinligiga qaraladi;

b) kompyuterlarda to'g'ridan – to'g'ri qayta ishlash mumkin bo'lgan ma`lumotlar diapazoniga. Bugungi kunda kompyuterlar 10-20 xonali sonlar ustida to'g'ridan – to'g'ri amallarni bajarishi mumkin. Agar masala shartida berilgan sonlar bundan ham ko'p xonali bo'lsa-chi?

c) amallarni bajarish tezligiga. Ma`lumki, kompyuterlar ichki qurilmalariga bog'liq ravishda amallarni turli tezliklarda bajaradi. Xo'sh, algoritmni bajarish uchun zarur bo'lgan vaqt masala shartida ko'rsatilgan vaqtdan katta bo'lsa, nima qilish kerak?

d) masalani hal qilish uchun jalb qilinadigan kompyuterlar soniga. Albatta, katta sondagi masalalarni bitta kompyuter yordamida hal qilinadi. Ammo, shunday masalalar mavjudki, ularni hal qilish uchun bitta kompyuter kamlik qiladi. Bunda algoritmning buyruqlari kompyuterlar o'rtasida taqsimlanadi. Shu o'rinda 17 million xonali sonni topish uchun 1000 ta kompyuter 1 hafta ishlaganini yodga olish mumkin.

3. Aniq yoki taqribiy yechish usulini tanlash. Aniq yechishning iloji bo'lmagan masalalarni taqribiy yechishga to'g'ri keladi. Ayrim hollarda masalaning aniq yechimiga olib boruvchi algoritmlar o'ta murakkab va ko'p vaqt talab qilgani sababli ham, bunday masalalar taqribiy yechiladi.

**4. Ma`lumotlar uchun mos tuzilmalarni tanlash.** Ko'pincha, algoritmlar uchun boshlang'ich ma`lumotlar maxsus tuzilmaga ega bo'lishi shart bo'lmaydi. Ammo, shunday masalalar borki, ular uchun boshlang'ich ma`lumotlar maxsus ko'rinishda ifodalangan bo'lishi shart. Maxsus tuzlimaga ega bo'lgan ma`lumotlar algooritm yordamida qayta ishlanadigan ma`lumotlarni kiritish, chiqarish va nazorat qilishda dasturchilar ishini osonlashtiradi. Bu holat ayniqsa ma`lumotlar bazasi bilan bog'liq masalalarda yoki zamonaviy ob`ektga asoslangan dasturlash tillarida alohida ahamiyat kasb etadi.

5. Loyihalash metodlarini tanlash. Qo'yilgan masalani hal qilish uchun algoritmlarni qay tarzda qurish lozim?

Algoritmlarni loyihalash metodlari – bu turli sohalarga oid bo'lgan masalalarni algoritmik yechishga qaratilgan bo'lib, alohida sinflarga oid masalalar uchun individual yondoshuvni talab qiladi.

Mazkur metodlarni o'rganish quyidagi sabablarga ko'ra muhim sanaladi. Birinchidan, ular yangi yoki yaxshi algoritmlarni ishlab chiqish uchun foydalanish mumkin bo'lgan universal printsiplar jamg'armasini taqdim etadi. Ikkinchidan, algoritmik metodlar informatika fanining asosi va mazmunini tashkil qiladi. Loyihalash mexanizmlariga ko'ra algoritmlarni sinflarga ham ajratish mumkin.

6. Ifodalash usullarini tanlash. Algoritm loyihasi qabul qilinganidan so'ng, endi uni qandaydir ko'rinishda ifodalash lozim. Bugungi kunda algoritmlarni so'zlar, blok-sxema, matematik formulalar orqali ifodalash usullari keng tarqalgan[2]. Algoritmlarga bag'ishlangan ilmiy adabiyotlarda asosan psevdokod usulidan, programmalashga bag'ishlangan adabiyotlarda esa dasturlardan foydalaniladi.

Psevdokod – bu tabiiy va dasturlash tillariga oid ayrim ko'rsatmalar majmuasidan iborat. Odatda, algoritmlarni psevdokodlar yordamida tabiiy tillarga qaraganda oson, qisqa va tushunarliroq ko'rinishda ifodalash mumkin. Shuni alohida ta`kidlash joizki, mutaxassislar tomonidan psevdokodlar uchun standart variant qabul qilinmagan va shu sababli mualliflar adabiyotlarda o'zlari uchun qulay

---

[2] Aripov M., Otaxanov N. Dasturlash usullari. –T.:Tafakkur gulshani, 2015. – betlar.

bo'lgan "sheva" laridan foydalanadilar. Bu o'rinda asosiy e`tibor psevdokodlar orali algoritmlarning avvalo ijrochilarga, qolaversa o'quvchilarga tushunarli bo'lishiga qaratiladi.

7. Algoritm korrektligini (to'g'ri ishlashini) baholash. Dasturchi o'zi qurgan yoki oldindan mavjud algoritmlarni chekli vaqtdan so'ng kutilgan natijani berishga qodirligini oldindan baholashi lozim. Masalan, avvalgi bobda keltirilgan ***ekub*** ni topish haqidagi masalani yechishning birinchi usuli har qanday natural sonlar juftligi uchun natija bersa, ikkinchi usul sonlardan biri 0 ga teng bo'lganda natija bermaydi. Ayrim algoritmlarning korrektligini ko'rsatish juda ham oson, bir qator algoritmlar uchun bu masala o'ta murakkab hisoblanadi.

Agar algoritmlar ma`lum bir boshlang'ich ma`lumotlar uchun to'g'ri natija berib, boshqalari uchun kutilgan natidjani bermasa, bunday algoritmlarga tegishli o'zgartirishlarni kiritish lozim bo'ladi. Taqribiy algoritmlarda aniq yechimdan ruxsat etilgan chetlanish masala shartida ko'rsatilganidan chegaralardan chiqmasligini isbotlash kerak bo'ladi.

8. Algoritmlarni tahlil qilish. Bu masala asosan algoritm berishi mumkin bo`lgan samara bilan bog'liq. Tahlil jarayonida algoritmlar samaradorligini ikki jihatdan baxolash mumkin: vaqqtbay va fazoviy samaradorlik. Vaqtbay samaradorlik algoritm ishlash tezligining ko'rsatkichi, fazoviy samaradorlikda esa algoritmning ishlashi uchun zarur bo'lgan tezkor xotira xajmi bilan baholanadi.

Algoritmlarning yana bir muhim xarkteristikasi soddalik bilan bog'liq. Bu hususiyat sub`ektiv hisoblanadi va turli odamlarda turlicha namoyon bo'lishi mumkin. Sodda algoritmlarni o'qish va tushunish oson bo`lishi bo`lishi bilan birga oson dasturlanadi. Demak, dastur matnida xatoliklar kam bo'ladi.

Algoritmlarning yana bir muxim tomoni umumiylik (universallik) bilan bog'liq. Ushbu jihat ikki holat bilan tavsiflanadi:

algoritm qurilgan masalaning umumiyligi xamda mumkin bo'lgan boshlang'ich ma`lumotlar diapazoni. Masalaning umumiyligi deganda shuni e`tiborga olish kerakki, umumiy masalalar uchun algoritm qurishning iloji bo'lmaganda, hususiy masala uchun algoritm ishlab chiqish tavsiya etiladi. Masalan, ikki natuarl sonnipng o'zaro tubligini tekshirish algoritmiga qaraganda, ularning ***ekub*** ni topish osonroq bo'ladi.

Mumkin bo'lgan boshlang'ich ma`lumotlar diapazogiga kelsak, shuni yodda tutish kerakki, odatda boshlang'ich ma`lumotlar katta diapazonda o'zgarishi mumkin bo'lib, yechilayotgan masala shartiga mos bo'lishi lozim. Masalan, kvadrat tenglama uchun kompleks sonlar ham boshlang'ich ma`lumotlar sifatida ishtirok etishi mumkin bo'lsada, odatda ular e`tiborga olinmaydi.

Algoritmlarni kodlash. Ko'pchilik algoritmlar qachondir kompyuter dasturlariga aylantiriladi. Dastur - bu algoritm buyruqlarni maxsus dasturlash tillaridan birida kompyuterlarga tushunarli ko'rinishda yozish usuli bo'lib, uning ijrochisi sifatida kompyuter tan olinadi. Demak, ta`rifga ko'ra dasturlarni ham algoritm deb qarash mumkin. Dasturlarning to'g'riligi (korrektligi) test sinovlar orqali tekshiriladi. Bu jarayonda boshlang'ich ma`lumotlar mumkin bo`lgan dipazaondan chetga chiqishi mumkinligiga alohida e`tibor qaratish zarur. Dasturning ishchi versiyasi ishlab chiqilganidan so'ng uning asosida yotgan algoritmni empirik tahlil qilishga to'g'ri keladi. Bunday boshlang'ich ma`lumotlarning turli qiymatlari uchun dasturning bajarilishi vaqti tahlil qilinadi.

Algoritmlarni yaxshilash. Algoritmlarni loyihalash ziddiyatli vaziyatlarda qaror qabul qilishni talab qiladigan o'ta murakkab masala hisoblanadi. Qo'yilgan masala uchun birinchi algoritm qurilganidan keyin, uni yaxshilash masalasini o'ylab ko'rish mumkin. Buning uchun yuqoridagi bosqichlarning ayrimlarini qayta va qayta bosib o'tishga to'g'ri keladi.

Umuman olganda, birinchi urinishdayoq yaxshi algoritm qurishga umid qilmasa ham bo'ladi. Ammo, qandaydir buyruqlarni qo'shish yoki olib tashlash evaziga uni yaxshilashga urinib ko'rish mumkin. Bu o'rinda Sent Ekzyuperining quyidagi so'zlar esga keladi: "Konstruktor o'zi yaratgan mahsulotga yangi narsalarni qo'sha olmagan xolda emas, balki olib tashlaydigan ortiqcha narsalar qolmagandagina mukammallikka erishdi deb hisoblash mumkin". Algoritmlarni yaxshilash jarayonini vaqt, resurs, mablag', mehnat kabi bir qator mavjud cheklovlarni e'tiborga olgan xolda to'xtatiladi.

3-§. ALGORITMLASHNING TIPIK MASALALARI

Bugungi kunda informatika fani deyarli hayotda uchraydigan barcha masalalarni qamrab olgan. Ular orasidan bir-biriga yaqinlarini shartli ravishda sinflarga birlashtirilgan. Ana shu sinflar orasida algoritmlar nazariyasi, dasturlash asoslari kabi fanlarni o'zlashtirishda ham nazariy, ham amaliy yordam beradigan quyidagi tipik masalalar ajratib olingan.

Natural sonli masalalar;

Tartiblash va izlash masalalari;

Kombinatorika masalalari;

Satrlarni qayta ishlash masalalari;

Optimallashtirish masalalari;

NP-to'liqligidagi masalalar.

Natural sonli masalalar. Hayotda natural sonlar bilan bog'liq masalalar juda ham ko'p uchraydi. Namuna tariqasida tub sonlar, fibonachchi sonlari, sanoq sistemalari bilan bog'liq masalalarni tilga olish mumkin. Bunday masalalar uchun qo'yiladigan asosiy talab boshlang'ich (kiruvchi) hamda natijaviy (chiquvchi) ma'lumotlarning hammmasi har qanday xolda ham faqat natural sonlardan iborat bo'ladi.

Tartiblash va izlash masalalari. Bunday masalalarda qandaydir ma`lumotlar ro'yhati taqdim etiladi va ularni ma`lum bir shart asosida tartiblash talab qilinadi. Berilgan familiyalar, abonentlar ro'yxati, ballar jamg'armasi, rejalashtirilgan tadbirlarni tartiblash kabi masalalar ana shular jumlasidan hisoblanadi. Bu sinf masalalari o'sish yoki kamayish munosabatini nazarda tutishi bilan boshqalaridan farq qiladi. Tartiblash bir yoki bir necha alomatlar yuzasidan amalga oshirilishi mumkin.

Izlash masalalari odatda tartiblash masalalari bilan chambarchas bog'liq bo'lib, ma`lum bir tartib bilan berilgan katta xajmdagi ma`lumotlar ro'yxatidan qandaydir savollarga javob topish muammolarini o'z ichiga oladi. Namuna tariqasida telefon abonentlari ma`lumotnomasidan ko'rsatilgan nomerga aloqador ma`lumotlarni topish masalasini tilga olish mumkin.

Tartiblash va izlash masalalarining amaliy ahamiyati katta hajmdagi ma`lumotlar orasidan qandaydir ma`lumotlarni izlash jarayonini osonlashtirishi bilan belgilansa, nazariy ahamiyati algoritmik usullar optimalligini baholashda xamda ilmiy tadqiqot natijalarini asoslashda ko'rinadi.

Shuni ta`kidlash joizki, tartiblash masalalar uchun o'nlab usullar ishlab chiqilgan va bu usallarning birortasini boshqasidan ustun qo'yib bo'lmaydi. Gap shundaki, bu usullarning bittasi bir hil boshlang'ich ma`lumotlar uchun yaxshi natija (vaqt ma`nosida) bersa, boshqa hil boshlang'ich ma`lumotlar uchun yomon natija berishi mumkin. Xuddi shuningdek, tartiblanmagan ro'yxatdan izlash masalalari uchun ham eng yaxshi algoritm ishlab chiqilmagan.

Satrlarni qayta ishlash masalalari. Bugungi kun amaliyotida raqamli bo'lmagan masalalarni qayta ishlash bilan bog'liq masalalar tez-tez uchramoqda. Satrlar tarkibiga muayyan bir alfavitdan olingan xarflar, raqamlar va boshqa maxsus belgilar, genlar, biror sanoq sistemasidan olingan (masalan, ikkilik yoki o'n oltilik) raqamlardan iborat bo'lishi mumkin. Bir satrning tarkibida ikkinchi satr (ostsatr) ni izlash masalalari ham keng tarqalgan.

Yana shunday masalalar borki, ularda berilgan sonli ma`lumotlar kompyuterning operativ xotirasiga sig'maydi. Bunday xollarda bunday ma`lumotlarni satrli shaklda qayta ishlash tavsiya etiladi. Bu o'rinda uzun sonlar arifmetikasiga doir masalalarni misol qilib keltirish mumkin.

Optimallashtirish masalalari. Ma`lumki, bozor iqtisodi davrida ko'plab masalalarni chegaralangan resurslar (vaqt, mablag`, mehnat, ishchilar soni, elektr quvvati, texnika va h.k.) sharoitida hal qilishga to'g'ri keladi. Bu holda berilgan boshlang'ich ma`lumotlar uchun shunday yechimlarni topish kerak bo'ladiki, bu yechimlar masala shartida ko'rsatilgan alomatlar bo'yicha boshqalaridan yaxshi sanaladi. Eng ko'p foyda olish uchun mahsulotlar ishlab chiqarishni rejalashtirish, resurslarni taqsimlash masalalari, eng qisqa yo'lni topish, avtobuslar xarakatini belgilash, servis punktlarini joylashtirish, dars jadvalini tashkil qilish kabi masalalar ana shular jumlasidan sanaladi.

Bu tipdagi masalalarni hal qilishda graflar nazariyasi elementlaridan keng foydalaniladi. Graflar real hayotda mavjud bo`lgan ko`plab murakkab jarayonlarni ko'rgazmali va tushunarli ko`rinishda ifodalashga yordam beradi. Bunday jarayonlarga turli kommunikatsiya tarmoqlarini, loyihalarni taqvimiy rejalarini, transport xarakati grafiklarini olish mumkin. Eng zamonaviy masalalardan biri WEB-texnologiyalarda uchrab turadigan bir sahifadan ikkinchisiga o'tishni optimallashtirish bilan bog`liq.

NP-to'liqligidagi masalalar. Dastur ishlab chiqishda berilgan buyumlar (xodisalar) uchun mavjud hamma imkoniyatlarni ko'rib chiqish bilan bog`liq masalalar alohida ahamiyat kasb etadi. Bu sinfga ryukzak masalasi, berilgan *N* sonini yig'indi shaklida ifodalashning barcha usullari, shaxmat bilan bog'liq mummmolar kabi masalalar kiradi. Masalaning kutilgan yechimini topish uchun mumkin bo'lgan barcha variantlarni ko'rib chiqishga to'g'ri keladi. Bunday masalalarning yomon tomoni shundaki, buyumlar yoki qarab chiqish kerak bo'lgan xolatlarning bittaga ko'payishi ko'rib chiqish lozim bo'lgan variantlar sonining keskin (eksponentsial) ortishiga sabab bo'ladi. Masalan, 4 ta xarflarni o'zaro o'rin almashtirishlari uchun 24 ta variant, 5 ta xarf uchun 120 ta variant, 6 ta xarf uchun esa 720 ta variant qarab chiqilishi kerak.

Bunday masalalarni NP-to'liqligidagi masalalar deb ataladi. Ular informatika, algoritmlar nazariyasi va dasturlash asoslarini o'rganishda katta nazariy va amaliy ahamiyatga ega. Shuni ta`kidlash joizki, NP-to'liqligidagi masalalar yechimini topish uchun hozirgacha birorta ham optimal algoritm ishlab chiqilmagan va shunday algoritm uchun bir million dollar mukofot e`lon qilingan.

4-§. MA`LUMOTLARNING ASOSIY TUZILMALARI

Ma`lumki, algoritmlar turli shakl va mazmundagi ma`lumotlarni qayta ishlash uchun quriladi. Ammo, ularni tahlil qilish va loyihalash jarayoniga ma`lumot tuzilmalari sezilarli ta`sir ko'rsatishi mumkin.

Ma`lumot tuzilmalari deganda maxsus tashkil qilingan va o'zaro bog'langan elementlar tizimi tushuniladi. Elementlar o'ta sodda (butun, haqiqiy, satrli, mantiqiy va b.) yoki murakkab (massivlar, yozuvlar, ob`ektlar kabi) tuzilmalarga ega bo'lishi mumkin.

Biz murakkab tuzilmali ma`lumotlar ustida to'xtalib o'tamiz.

Chiziqli tuzilmali ma`lumotlar. Bir o'lchovli massivlar xamda bog'langan ro'yxatlarni chiziqli tuzilma sifatida qarash mumkin.

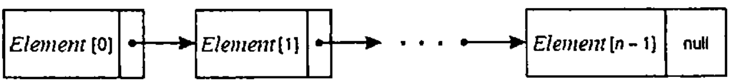
Bir o'lchovli massivlar (2-rasm) bir hil tipdagi elementlardan tashkil topgan bo'lib, uning elementlariga indekslarini ko'rsatish orqali murojaat qilinadi.

[0] –chi element	[1] –chi element	· · ·	[*n*] –chi element

2-rasm. Bir o'lchovli massiv

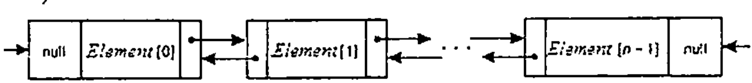
Massiv elementlariga murojjat qilish vaqti hammasi uchun bir hil. Massivlar yordamida belgili massiv yoki satrlarni hosil qilish mumkin. Ular ustida qidirish, uzunligini aniqlash, birlashtirish, bir qismini o'chirish yoki ko'chirish, almashtirish kabi amallarni bajarish mumkin.

Bog'langan ro'yhat – tugun deb ataladigan ma`lumotlar zanjiridan iborat bo'lib, har bir tugun ikki qismdan tashkil topadi. 1-qismi bevosita ma`lumotni, 2-qismi esa navbatdagi tugun manzilini ko'rsatadi (3-rasm).



3-rasm. Bir tomonlama bog'langan ro'yhat.

Tugunlar bog'lanishlar usuliga qarab bir tomonlama yoki ikki tomonlama bo'lishi mumkin. Bir tomonlama ro'yxatlarda birinchi tugundan boshlab barcha elementlar o'zi saqlayotgan ma`lumotdan tashqari, navbatdagi tugun manzilini ko'rsatadi ham ko'rsatadi. Bu xolda ro'yxatning oxirgi elementi hech qanday tugunni ko'rsatmaydi. Ikki tomonlama ro'yxatlarda esa xar bir tugun o'zidan avvalgi va keyingi tugun manzillarini ko'rsatadi. Birinchi tugun o'zidan avvalgi, oxirgi tugun esa o'zidan keyingi tugun manzillarini ko'rsatmaydi (4 – rasm).



4-rasm. Ikki tomonlama bog'langan ro'yhat

Bog'langan ro'yxatlarda berilgan elementga o'tish uchun avval ro'yxatning birinchi elementiga o'tiladi va so'ngra, qadam baqadam o'tib, ko'rsatilgan tugunga o'tiladi. Bog'langan ro'yxatlarning kamchiligi ana shu holat bilan, afzalligi esa kompyuter operativ xotirasidan oldindan joyni band qilish shart emasligi bilan belgilanadi. Bundan tashqari, ro'yxatga yangi tugunlarni qo'shish yoki nokeraklarini olib tashlash massivlarga qaraganda ham oson hal qilinadi.

Stek – bu bir tomonlama bog'langan ro'yxatning hususiy holi bo'lib, ma'lumotlarni qarab chiqish faqat uning uchi deb ataladigan tomonidan amalga oshiriladi. Steklar bilan boshqa amallarni bajarish nazarda tutilmagan. Steklar LIFO (last in – first out – oxirgi kelgan birinchi ketadi) prinsipida ishlaydi.

Navbat – bu bir tomonlama bog'langan ro'yxatning xususiy holi bo'lib, yangi elementlar uning bir uchidan, tanlab olish esa ikkinchi uchidan amalga oshiriladi. Navbat uchun boshqa amallar nazarda tutilmagan. Tanlab olingan elementlar navbatdan chiqariladi. Navbatlar FIFO (first in – first out – birinchi bo'lib kelgan birinchi bo'lib ketadi) prinsipi asosida ishlaydi.

Strukturalar. Dasturlashda eng ko'p uchraydigan ma`lumot tuzilmalaridan biri strukturalar (ayrim dasturlash tillarida yozuvlar yoki aralash tipli ma`lumotlar deb ham ataladi) hisoblanadi. Bu tipdagi bitta ma`lumot bir nechta maydonlardan iborat bo'lishi mumkin. Har bir maydon mazmun va tipi bir xil ma'lumotlarni o'z ichiga oladi. Masalan, bitta talaba xaqidagi strukturani quyidagicha shakllantirish mumkin: birinchi maydonda - familiyasi, ikkinchi maydonda - ismi, uchinchi maydonda - tug'ilgan sanasi, to'rtinchi maydonda - kursi, beshinchi maydon esa informatika fanidan to'plagan ballari saqlanadi. Bu xolda 5 ta maydondan iborat struktura tashkil qilindi deyiladi.

Strukturaning u yoki bu maydoniga uning nomi, so'ngra nuqta belgisi (".") va maydon nomini ko'rsatish orqali murojaat qilinadi.

Masalan, x.Familiya, x.Ismi va h.k. Bu tipdagi ma`lumotlar odatda fayllar yoki ma`lumotlar bazasi bilan ishlaganda qo'l keladi.

5-§. Algoritmlarning samaralilik darajasini aniqlash

5.1. **Algoritmlarning samaradorligi tushunchasi.** Avvalgi bobda ta`kidlanganidek, bitta masalani bir nechta algoritmlar yordamida hal qilish mumkin. Bunday hollarda o'z-o'zidan "bu algoritmlarning qaysi biri yaxshi" degan haqli savol tug'iladi. Unga javob berish uchun albatta har bir algoritmni turli parametrlar (zarur hollarda miqdoriy parametrlar) yordamida o'rganish talab qilinadi.

Aytish joizki, algoritmlarning soddalik va universallik hossalarini intuitiv, samaralilik jihatlarini esa miqdorlar orqali baholash mumkin.

Algoritmlar samaraliligini vaqtbay va fazoviy jihatlariga ko'ra baholash mumkin. ***Vaqtbay samaralik*** algoritm ishlash tezligining indikatori bo'lib hizmat qilsa, ***fazoviy samaralik*** algoritmning ishlashi uchun qancha qo'shimcha xotira talab qilinishini belgilab beradi.

XX asrning 50-yillarida xar ikki resurs ham (protsessorning ishlash tezligi hamda operativ xotira hajmi) hal qiluvchi ahamiyatga ega bo'lgan. Ammo, bugunga keli aytish mumkinki, hisoblash qurilmalarining tezliklari va operativ xotira hajmi beqiyos darajada o'sdi. Shu sababli, qo'shimcha xotiraga ehtiyoj yo'qoldi. Ammo, vaqtbay va fazoviy samaralilik masalasi o'z ahamiyatini yo'qotganicha yo'q.

Dasturchilar uchun algoritmlarning bajarilish vaqti quyidagi parametrlarga bog'liq ekanligi tabiiy:

kiruvchi ma`lumotlar hajmi;

konkret kompyuterning ishlash tezligi;

algoritmlarni dastur shakliga keltirishdagi mohirlik;

kompilyatorning tipi;

dastur real bajarilish vaqtining o'lchashdagi aniqlik.

Biz bu sanab o'tilgan parametrlarga e`tibor bermaymiz. Chunki, bir masala uchun qurilgan algoritmlarni bir hil sharoit va ma`lumotlarga nisbatan baholash mantiqan to'g'ri hisoblanadi.

Bu muammoni hal qilishning mumkin bo'lgan usullaridan biri algoritmning bajariladigan amallari sonini sanashdan iborat bo'lib, uni amaliyotga joriy qilish o'ta murakkab va chigal hisoblanadi. Qo'yilgan muammoni hal qilishning eng yaxshi yo'li – bu algoritmning umumiy bajarilish vaqtiga ta`sir ko'rsatishi mumkin bo'lgan bazaviy amallar va ularning bajarilishlar sonini aniqlashdan iborat.

Odatda algoritmning bazaviy amallarini aniqlash qiyin bo'lmaydi. Bazaviy deganda algoritmning eng ko'p bajarilishi lozim bo'lgan amallar nazarda tutiladi. Algoritmning ichki tsikllarida ko'rsatilgan amallar uning bajarilish vaqtiga ta`sir ko'rsatishi tabiiy. Masalan, saralash algoritmlarida ro'yxatning ikki elementini taqqoslash, matritsani matritsaga ko'paytirishda esa mos elementlarni ko'paytirish va ularning yig'indisini hisoblash ana shunday amallardan sanaladi. Kompyuterlarda ko'paytirish amali yig'indini hisoblashga nisbatan uzoqroq bajarilgani uchun, ularni bemalol bazaviy amallar sarasiga qo'shish mumkin.

Shunday qilib, algoritmlarning samaradorlik darajasini aniqlashda n ta kiruvchi ma`lumotlar uchun bazaviy amallarning bajarilishlar soni e`tiborga olinadi.

Faraz qilaylik, algoritm asosiy amalining konkret kompyuterda bajarilish vaqti - c_{bv}, bu amalning bajarilishlar soni esa - $C(n)$ bo'lsin. U holda bu algoritmning mazkur kompyuterda bajarishning umumiy vaqti – $T(n)$ quyidagi formula bilan tahminiy aniqlanishi mumkin:

$$T(n) \approx c_{bv} C(n).$$

Bu formula asosiy bo'lmagan amallarning bajarilish vaqtini e'tiborga olmaydi, ammo algoritmning tahminiy bajarilish vaqtini aniqlashga yordam bera oladi. Qolaversa, undan bitta algoritmning o'zi tezligi 10 marta katta bo'lgan kompyuterda qancha vaqt mobaynida bajariladi? yoki $C(n) = n(n-1)/2$ bo'lganda kiruvchi ma`lumotlar soni ikki marta orttirilsa algoritmning bajarilish vaqti necha marta o'zgaradi? degan savollarga javob berishda foydalanish mumkin. Ko'rish qiyin emaski, tezlik bu savolarning birinchisida 10 marta, ikkinchisida esa 4 marta sekinlashadi. Haqiqatdan xam, yetarlicha katta n lar uchun quyidagi munosabat o'rinli:

$$C(n) = \frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \approx \frac{1}{2}n^2 = 0{,}5n^2.$$

Shuning uchun

$$\frac{T(2n)}{T(n)} \approx \frac{c_{bv}C(2n)}{c_{bv}C(n)} \approx \frac{0{,}5(2n)^2}{0{,}5n^2} = 4.$$

Ikkinchi savol javobidan ko'rinib turibdiki, kiruvchi ma`lumotlar soni $n$ ning o'zgarishi algoritmning bajarilish vaqtiga jiddiy ta`sir ko'rsatadi. yetarlicha katta n lar uchun funktsining o'sishi tartibini aniqlash talab qilinadi. Quyidagi jadvalda ayrim funksiyalar uchun o'sish tartiblari keltirilgan[3].

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Shuni ta`kidlash joizki, sekundiga trillion (2^{12}) amal

[3] Левитин А. Алгоритмы. Введение в разработку и анализ. –М.:Вильямс, 2006. Стр. 80.

bajaradigan kompyuter uchun 2^{100} ta amalni bajarish uchun $4 \cdot 10^{10}$ yil kerak bo'ladi. Bu omilni e`tiborga olsak, jadvalning oxirgi ustunidan ko'rinib turibdiki, *n* ning kichik o'zgarishiga bajariladigan amallarning soni keskin o'zgaradigan hollarda algoritmlarni *n* ning kichik qiymatlari bajarishga to'g'ri keladi.

Ayrim algoritmlarning bajarilishi tezligiga nafaqat boshlang'ich ma`lumotlarning o'lchami, balki kiruvchi ma`lumotlarga hos hususiyatlar ham ta`sir ko'rsatishi mumkin. Masalan, ketma-ket taqqoslash usuli yordamida boshlang'ich ma`lumotlar orasidan biror kalit so'z izlanayotgan bo'lsa, algoritm o'z ishini yoki izlangan element topilganda, yoki ro'yxatdagi hamma elementlar qarab chiqilganidan keyin to'htatadi. Quyidagi algoritm psevdokodi ana shu holatni namoyish qiladi.

ALGORITM Sequential Search (A [0..n - 1], K)

// **Kiruvchi ma`lumotlar:** sonlar massivi *A[0..p-* 1] va K kalit

// **Chiqivchi ma`lumotlar:** K ga teng bo'lgan birinchi element

// indeksi yoki agar izlangan element topilmasa -1 chiqariladi.

i ←0

while** i < p **and** A[i] ≠ K **do

i←i+1

if i < p

***return** i*

else

***return** -1*

Tabiiyki, bu algoritmning bajarilish vaqti diapazoni *n* ning bir hil qiymati va turli kiruvchi ma`lumotlar uchun juda ham keng bo'lishi mumkin. Eng yomon holat yoki izlanayotgan element massiv oxirida joylashganda yoki umuman mavjud bo'lmaganda yuzaga keladi va bunda algoritm amallari eng ko'p marta bajariladi.

Eng yomon hodisalar oqimi uchun algoritm samaradorligi deganda uning eng yomon hisoblangan kiruvchi ma`lumotlar uchun samaradorligi tushuniladi. Bunday samaradorlikni algoritmni kiruvchi ma`lumotlar uchun tahlil qilish orqali osongina aniqlash mumkin.

Eng yaxshi hodisalar oqimi uchun algoritm samaradorligi deganda uning algoritmning ishlash vaqti eng kichik bo'lishini kafolatlovchi eng yaxshi hisoblangan kiruvchi ma`lumotlar uchun samaradorligi nazarda tutiladi. Bunday samaradorlikni ham algoritmni kiruvchi ma`lumotlar uchun tahlil qilish orqali osongina aniqlasa bo'ladi. Eng yaxshi holat uchun algoritm samaradorligini quyidagicha tahlil qilish mumkin. Dastlab kiruvchi ma`lumotlarning qanday variantlari uchun *C(n)* eng kichik bo'lishi aniqlanadi. Masalan, izlanayotgan ma`lumot ro'yxat boshida joylashganda izlash algoritmi eng qisqa vaqt mobaynida ishlaydi va shu sababli bu holni eng yaxshi holat deb tan olinadi.

Shuni ta`kidlash joizki, eng yaxshi holat uchun algoritm samaradorligi unchalik ham muhim emas, lekin baribir, bunday hollarni albatta hisobga olishga to'g'ri keladi.

Nima bo'lganda ham, algoritm samaradorligini baholashda eng yomon holat uchun qilingan hulosalar muhim sanaladi. Biz ham bunday keyin ana nuqtai-nazardan ish olib boramiz.

5.2. Norekursiv algoritmlarning matematik tahlili. Algoritmlarni tahlil qilishning barcha bosqichlarini o'z ichiga olgan quyidagi sodda misolni ko'raylik.

1-misol. Eng katta elementni topish masalasini *n* – ta elementli ro'yxat (yoki massiv) uchun qarab chiqamiz. Quyida shu masala algoritmning psevdokodi keltirilgan.

ALGORITM MaxElement (A [0..n - 1])

// **Kiruvchi ma`lumotlar:** *A[0..p - 1]* haqiqiy sonlar massivi

// **Chiquvchi ma`lumotlar:** A massivning eng katta elementi

maxval ←A[0]

for i← 1 to n -1 do
if A [i] > maxval
maxval ← A [i]
***return** maxval*

Bu algoritm uchun bazaviy amallar sifatida taqqoslash (*A[i]>maxval)* hamda qiymat berish *(maxval←A[i])* amallari olinadi. Bu holda taqqoslash amali tsiklning xar bir qadamida, qiymat berish amali esa ayrim hollarda bajariladi. Ma`lumki, tsiklning har bir qadamida taqqoslash amali bir marta bajariladi. Qadamlar soni esa *n* ta kiruvchi ma`lumotlar uchun *n-1* ga teng.

Quyida norekursiv algoritmlar samaradorligini tahlil qilishning umumiy rejasi bayon etiladi.

1. Algoritmning kiruvchi ma`lumotlari o'lchamini baholashda e`tiborga olinishi lozim bo'lgan parametr (yoki parametrlar) tanlanadi.

2. Algoritmning asosiy amallarini aniqlang. Odatda bu amallar tsikl ostida joylashadi.

3. Bazaviy amallar sonini kiruvchi ma`lumotlar o'lchamiga bog`liq ekanligini tekshiring. Agar shunday bog'liqlik boshqa omillar uchun ham mavjud bo'lsa, ular uchun ham algoritm samaradorligini tahlil qiling.

4. Bazaviy amallarning bajarilishlar sonlarining umumiy yig'indisini aniqlang.

5. Standart formula va qoidalar asosida bazaviy amallar miqdorini aniqlang. Agar buning imkoni bo'lmasa, u xolda hech bo'lmaganda ularning o'sish tartibini toping.

2-misol. Massivni har xil elementlardan tashkil topganligini aniqlang.

Bu masalani quyidagi keltirilgan sodda algoritm yordamida hal qilinadi.

ALGORITM *UniqueElements (A* [0..n - 1])

// **Kiruvchi ma`lumotlar:** haqiqiy sonlar massivi *A[0..n-1*]

// **Chiquvchi ma`lumotlar:** agar A massiv elementlari har xil

// bo'lsa "**true**", aks holda "**false**"

for $i \leftarrow 0$ **to n-2 do**

for $j \leftarrow i+1$ **to** $n-1$ **do** *if* $A[i] = A[j]$

return false

else return true

Mazkur algoritmda kiruvchi ma`lumotlar o'lchami massiv elementlari soni bilan aniqlanadi. Unda bazaviy amal sifatida tsikl ostidagi taqqoslash amalini olish mumkin. Taqqoslash amallari soni nafaqat massiv elementlar soniga, balki massivda bir hil elementlarning mavjudligi va agar mavjud bo'lsa, ularning qanday o'rinlarda joylashganligiga bog'liq bo'ladi.

Algoritm uchun eng yomon holat (bajariladigan amallar sonining maksimal bo'lishi) ikki holda yuzaga kelishi mumkin: a) massivda bir hil elementlar mavjud bo'lmaganda; b) ikkita bir hil element m avjud va ular massivning oxirida joylashganda. Tashqi tsikl parametri i bir marta o'zgarganda (hammasi bo'lib n marta o'zgaradi) ichki tsiklning j paremetri $i+1$ dan n gacha o'zgaradi va har bir o'zgarishga mos ravishda ichki tsikldagi taqqoslash amali bir marta bajariladi. Shuning uchun bazaviy amallarning umumiy soni $\frac{n(n-1)}{2} \approx \frac{n^2}{2}$ ga teng bo'ladi.

Yana bir misol ko'raylik.

4-misol. 10 lik sanoq sistemasida berilgan musbat sonning 2 lik sanoq sistemasidagi razryadlar sonini aniqlaylik.

ALGORITM Binary (n)

// **Kiruvchi ma`lumotlar** : musbat butun son n

// **Chiquvchi ma`lumotlar:** n sonini 2 lik ko'rinishidagi razryadlar soni *count* $\leftarrow 1$ *while* $n > 1$ *do*

count $\leftarrow$ *count* + *1*

$n \leftarrow \lfloor n/2 \rfloor$

***return** count*

Mazkur algoritmda n>1 taqqoslash amali tsikl ichida joylashmagan bo'lsada, eng ko'p bajariladi va tsiklning takrorlanish yoki takrorlanmasligini hal qiladi. Taqqoslash amali tsiklga qaraganada bir mart ko'p bajarilganligi uchun, bazaviy amalni tanlash unchalik ham muhim bo'lmaydi.

Har gal n ning qiymati ikki marta kamaygani uchun, tsiklning takrorlanishlar soni $log_2 n$ ga, $n>1$ taqqoslash amalining bajarilishlar soni esa $\lfloor \log_2 n \rfloor + 1$ ga teng bo'ladi.

5.3. Rekursiv algoritmlarning matematik tahlili. Quyidagi masalani ko'raylik.

1-misol. Ihtiyoriy musbat n butun soni uchun $F(n) = n!$ faktorialni hisoblang.

Ma`lumki, ta`rif bo'yicha $0! = 1$ hamda barcha $n>1$ lar uchun $n! = (n-1)! \cdot n$. Shu sababli $F(n) = F(n-1) \cdot n$ funksiya qiymatini quyidagi algoritm yoradmida hisoblash mumkin.

Algoritm *F(n)*

// **Kiruvchi ma`lumotlar**: nomanfiy butun son n

// **Chiquvchi ma`lumotlar:** $n!$ ning qiymati

if $n = 0$

  return 1

else

  return $F(n-1) \cdot n$

Bu algoritm uchun bazaviy amal bo`lib ko`paytirish bo`lib, uning bajarilishlar sonini *M(n)* orqali belgilaymiz. Barcha $n>0$ lar uchun *F(n)* funksiyaning qiymati $F(n) = F(n-1) \cdot n$ formula bilan hisoblangani uchun ko'paytirishlar soni quyidagicha bo'ladi:

$$M(n) = \underbrace{M(n-1)}_{\substack{F(n-1)ni \\ hisoblash\,uchun}} + \underbrace{1}_{\substack{F(n-1)ni\; nga \\ ko'paytirish}}$$

Bu formulada *M(n)* ning qiymati oshkormas holda, ya'ni *n* ning funksiyasi orqali berilgan. Boshqacha aytganda, *M(n)* ning qiymati shu funksiyaning avvalgi qadamdagi qiymatiga bog'liq bo'lmoqda. Bunday munosabatlarni rekkurent munosabatlar (yoki tenglamalar) deb ataladi. Bizning maqsad $M(n)=M(n-1)+1$ funksiya qiymatini aniqlashdan iborat. Bu savolga bir qiymatli javob berish uchun boshlang'ich qiymatdan foydalanamiz:

$$\textbf{if } n=0 \textbf{ return 1}$$

Ko'rinib turibdiki, $n=0$ bo'lganda rekursiv murojaat to'xtatiladi va bu hol uchun $M(n)=0$. Shunday qilib, algoritmning bazaviy amallari quyidagicha marta bajariladi:

$$\begin{cases} M(n)=M(n-1)+1, & agar\ n>0 \\ M(0)=0, & agar\ n=0 \end{cases}$$

Bu funksiya qiymatini aniqlash uchun teskari o'rniga qo'yish usulidan foydalanamiz:

$M(n)=M(n-1)+1=$

$M(n-1)=M(n-2)+1$ ni qo'yamiz

$=[M(n-2)+1]+1=$

$=M(n-2)+2=$

$M(n-2)=M(n-3)+1$ ni qo'yamiz

$=[M(n-3)+1]+2=$

$=M(n-3)+3.$

Bu munosabatlarni mavjud qonuniyatni ko'rish mumkin:

$$M(n)=M(n-i)+i.$$

Ushbu formulaning to'g'riligini matematik induktsiya metodi bilan ko'rsatish mumkin.

Oxirgi formulaga nisbatan boshlang'ich shartni qo'llaymiz:

$$M(n) = M(n-1)+1 = ... = M(n-i)+i = ... = M(n-n)+n = n.$$

Shuni ta`kidlash joizki, faktorialni hisoblash uchun soddaroq bo'lgan mexanizmlardan ham foydalanish mumkin va bu holda murakkab rekursiv murojaatlar kerak bo'lmas edi. Biz bu misolni o'quvchilarga yaxshi tanish bo'lgani uchun tanladik va rekurrent munosabatlarni ishlash printsipini bayon etdik xolos.

Shuni e`tiborga olish kerakki, $n!$ ni hisoblash uchun algoritmning vaqtbay samaradorligi unchalik muhim emas. Chunki faktorial juda ham tez o'sishni ta`minlaydi va shu sababli uni n ning unchalik katta bo'lmagan qiymatlari uchun hisoblanadi.

Yuqorida keltirilgan misol asosida rekursiv algoritmlarni matematik tahlil qilishning umumiy rejasini bayon qilamiz.

Rekursiv algoritmlarni matematik tahlil qilishning umumiy rejasi:

1. Algoritmning kiruvchi ma`lumotlari o`lchamini baholashda e`tiborga olinishi lozim bo'lgan parametr (yoki parametrlar) tanlanadi.

2. Algoritmning asosiy amallarini aniqlang. Odatda bu amallar tsikl ostida joylashadi.

3. Bazaviy amallar sonini kiruvchi ma`lumotlar o'lchamiga bog'liq ekanligini tekshiring. Agar shunday bog'liqlik boshqa omillarda ham mavjud bo'lsa, ular uchun ham algoritm samaradorligini tahlil qiling.

4. Bazaviy amallarning bajarilishlar sonlari yig'indisini hisoblash uchun rekkurent formula ishlab chiqing hamda unga mos boshlang'ich shartlarni ko'rsating.

5. Rekkurent tenglamani yoching yoki agar buning iloji bo'lmasa, hech bo'lmaganda o'sish tartibinri aniqlang.

2-misol. (Hanoy minorasi) Uchta sterjen-o'q hamda n ta turli o'lchamdagi halqalar berilgan bo'lsin. Halqalar kengliklarining kamayi shi tartibida bitta sterjenga o'tkazilgan. Bir vaqtning o'zida faqat bitta halqani olish mumkin. Halqa ustiga faqat o'lchami undan kichik bo'lgan halqalarni qo'yish mumkin. Shu shartlar ostida halqalarni kamayish tartibida uchinchi sterjenga o'tkazish talab qilinadi. Bunda ikkinchi sterjendan yordamchi strejen sifatida foydalanishga ruxast beriladi.

Bu masalani chiroyli rekursiv algoritm yordamida hal qilish mumkin (5.1-rasm). Halqalarni (n>1) birinchidan uchinchi sterjenga o'tkazish uchun avval n-1 ta halqani uchinchi sterjendan yordamchi sifatida foydalangan holda ikkinchi sterjenga rekkurent o'tkaziladi. Shunday keyin eng katta halqani uchinchi sterjenga o'tkaziladiva nihoyat, qolgan n-1 ta halqani uchinchi sterjenga rekursiv o'tkaziladi.

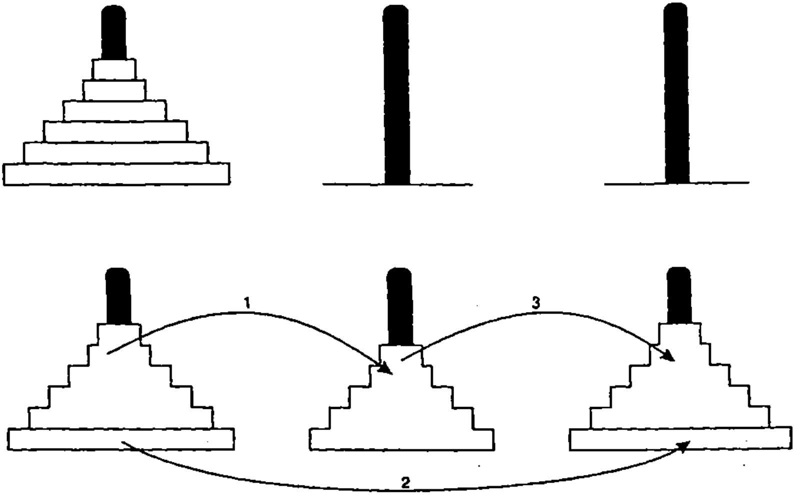
Tabiiyki, algoritm uchun kiruvchi ma`lumotlar o'lchami halqalar soni bilan baholanadi. Shu sababli, bir sterjendan ikkinchisiga olib o'tishlar soni $M(n)$ faqat n soniga bog'liq bo'ladi va ihtiyoriy n>1 uchun quyidagi rekkurent munosabatlar ham o'rinli bo'ladi:

$$M(n) = M(n-1) + 1 + M(n-1).$$

Bu munosabatga $M(1) = 1$ ni qo'shamiz:

$$\begin{cases} M(n) = 2M(n-1), & agar\ n > 1; \\ M(1) = 1, & agar\ n = 1. \end{cases}$$

Bu tenglamani yechish uchun avvalgi bobdagi kabi, teskarisidan qo'yish usulidan foydalanamiz.



1.1-rasm. Hanoy minorasi masalasini rekursiv yechish sxemasi.

$M(n) = 2M(n-1)+1 =$

$M(n-1) = 2M(n-2)+1$ ni qo'yamiz.

$= 2[2M(n-2)+1]+1 =$

$= 2^2 M(n-2)+2+1 =$

$M(n-2) = 2M(n-3)+1$ ni qo'yamiz.

$= 2^2[2M(n-3)+1]+2+1 = 2^3 M(n-3)+2^2+2+1.$

Bu yechimning dastlabki uchta satrida oddiy qonuniyat ko'rsatiladi, shuning uchun to'rtinchi satr quyidagicha ko'rinishga ega bo'ladi:

$2^4 M(n-4)+2^3+2^2+2+1.$

Yechim uchun satr nomeri o'rniga i ni qo'yib, quyidagidagi umumlashtirilgan formulaga ega bo'lamiz:

$M(n) = 2^i M(n-i)+2^{i-1}+...+2+1 = 2^i M(n-i)+2^i-1.$

Boshlang'ich shartlar $n=1$ bo'lgan hol uchun berilganligini e`tiborga olinsa, shu satrga mos rekkurent munosabat yechimini topish maqsadida $i = n-1$ ni qo'yilsa, quyidagi munosabat hosil bo'ladi:

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

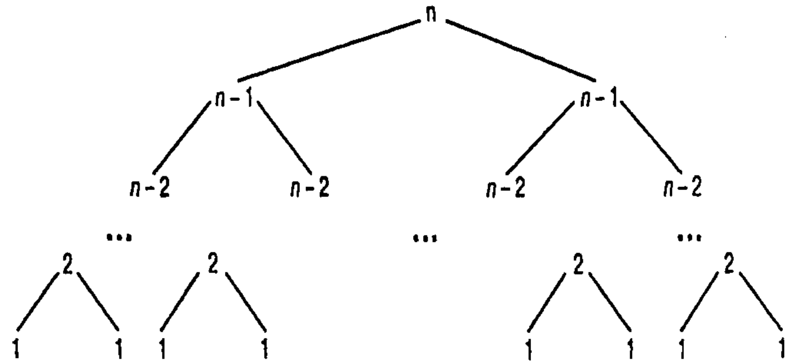
Ko'rinib turibdiki, qaralayotgan algoritm eksponentsial algoritm- lar sinfiga kiradi. Shuning uchun u n ning unchalik katta bo'lmagan qiymatlari uchun ham juda uzoq ishlaydi. Ammo, bu degani yomon algoritm qurildi degani emas. Algoritmning uzoq ishlashi shunchaki uning murakkabligi bilan bog'liq xolos.

Shuni ta`kidlash joizki, rekursiv algoritmlar bilan ehtiyot bo'lish lozim, chukni ularning "sirti chiroyli" bo'lgani bilan samaradorligi juda ham past bo'lishi mumkin.

Agar rekursiv algoritmda uning o'ziga bir nechta marta murojaat qilinsa, bunday algoritmlarni tahlil qilish uchun rekursiv murojaatlar daraxtini qurish maqsadga muvofiq hisoblanadi. Daraxtning tugunlari rekursiv murojaatlarga mos keladi. "Hanoy minorasi" uchun rekursiv murojaatlar daraxti 5.2-rasmdagi kabi quriladi. Undagi tugunlar sonini sanab, algoritmda bajarialdigan amallar sonini aniqlash mumkin bo'ladi:

$$C(n) = \sum_{l=0}^{n-1} 2^l = 2^n - 1,$$

bu yerda i – 5.2-rasmda tasvirlangan daraxt tuganlari darajasini bildiradi. Ko'rinib turibdiki, bu son halqalarni bir sterjendan ikkkinchisiga olib o`tishlar sonini anglatadi.



5.2-rasm. Hanoy minorasi uchun rekursiv murojaatlar daraxti.

6-§. Dekompozitsiya metodi

6.1. Umumiy tushunchalr. Dekompozitsiya metodi (uni "ajrat va hukmronlik qil" metodi ham deb ataladi) algoritmlar qurishda eng ko'p qo'llanadigan metodlardan hisoblanadi. Yana boshqa bir qator samarali algoritmlar o'z strategiyalarida bu usuldan keng qo'llanadi.

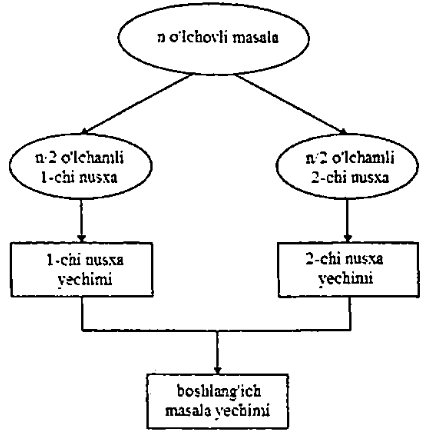
Dekompozitsiya metodiga asoslangan algoritmlarning ishlash rejasi quyidagi bosqichlardan iborat bo'ladi:

1. Berilgan boshlang'ich masala shu masalaning bir nechta kichik nusxalariga ajratiladi. Odatda bu nusxalarning o'lchamlari bir hil bo'ladi.

2. Masalaning kichik nusxalari hal qilinadi (masalan, rekursiv asosda), boshqa hollarda esa boshqa algoritmlardan foydalanish mumkin.

3. Boshlang'ich masalaning yechimi kichik masala nusxalari uchun olingan yechimlarning kombinatsiyalari shaklida topiladi.

Dekompozitsiya metodining ishlash sxemasi 6.1-rasmda tasvirlangan. Unda boshlang'ich masala o'lchamlari bo'yicha teng bo'lgan ikkita kichik masala nusxalariga ajratiladi. Bunday yondoshuv amaliyotda (masalan, bitta protsessorli kompyuterlar uchun ishlab chiqilgan algoritmlarda) juda ham ko'p uchraydi. Namuna sifatida $a_0, \ldots, a_{n-1}$ sonlarning yig'indisini hisoblash masalasini ko'raylik. Agar $n>1$ bo'lsa, bu masalani ikkita kichik nusxalarga ajratish mumkin: dastlabki $\lfloor n/2 \rfloor$ ta sonlar va qolgan $\lceil n/2 \rceil$ sonlar yig'indilarini *hisoblash*. Tabiiyki, agar $n=1$ bo'lsa, u holda yig'indi a_0 ga teng bo'ladi. Bu yig'indilarning har biri hisoblanganidan (rekursiv asosda) keyin yakuniy natijaga ega bo'lish uchun ularning qiymatlarini qo'shib qo'yiladi:



6.1-rasm. Dekompositsiya metodi.

$$a_0 + \cdots + a_{n-1} = \left(a_0 + \cdots + a_{\lfloor n/2 \rfloor - 1}\right) + \left(a_{\lfloor n/2 \rfloor} + \cdots + a_{n-1}\right).$$

Mazkur algoritm n ta sonlar yig'indisini hisoblashning eng samarali usuli bo'la oladimi? Bu algoritmni qo'pol kuchga asoslangan algoritm bilan taqqoslaganda (masalan, to'rtta son yig'indisi uchun) uchun osongina "yo'q" degan javobni berish mumkin.

Shunday qilib, har qanday holda ham dekompozitsiya metodidan foydalanish maqsadga muvofiq bo'lavermas ekan. Shunday bo'lsada, dekompozitsiya metodi kibernetikaga bir qator samarali va muhim algoritmlarni taqdim etgan. Bu metod ayniqsa parallel hisoblashlarni tashkil qilishda (bunda masalaning xar bir nusxasi alohida protsessorlarda bajariladi) juda ham qulay qulay usul hisoblanadi.

6.2. Birlashtirib tartiblash masalasi. Bu masala (mergesort) dekompozitsiya metodi ishini namoyish qiluvchi eng ajoyib masalalardan biri hisoblanadi. Bu usul g'oyasiga ko'ra berilgan A[0..*n*-1] massiv ikkiga bo'lish ($A[0..\lfloor n/2 \rfloor]$ hamda $A[\lceil n/2 \rceil..n-1]$ massivlar) orqali tartiblanadi. Bunda rekursiv asosda har bir qism massivlar tartiblanadi va ish yakunida tartiblangan qism massivlar birlashtiriladi. Mazkur algoritm uchun psevdokod quyidagicha yoziladi:

Algoritm Mergesort (A [0..*n* - 1])
// Rekursiv asosda A[0..*n* - 1] massiv tartiblanadi
// **Kiruvchi ma'lumotlar**: tartiblanadigan A[0..*n*-1] massiv
// **Chiquvchi ma'lumotlar:** tartiblangan A[0..*n*-1] massiv
if n > 1
$A[0..\lfloor n/2 \rfloor]$ ni $B[0..\lfloor n/2 \rfloor]$ ga ko'chirilsin
$A[\lceil n/2 \rceil..n-1]$ ni $C[\lceil n/2 \rceil..n-1]$ ga ko'chirilsin
Mergesort($B[0..\lfloor n/2 \rfloor]$)
Mergesort($C[\lceil n/2 \rceil..n-1]$)
Merge(B, C, A)

Ikkita tartiblangan masivlarni birlashtirish quyidagicha amalga oshiriladi. Ikkita ko'rsatkich (massiv indekslari) birlashtirilishi talab qilingan massivlarning birinchi elementlarini ko'rsatib turadi. So'ngra bu elementlar taqqoslanadi va ularning kichigi yangi massivga kiritiladi. Shunday keyin kichik elementni ko'rsatib turgan ko'rsatkich birga orttiriladi va mos massivning navbatdagi elementini ko'rsata boshlaydi. Bu amal birilashtirilayotgan massivlardan birida bitta ham element qolmaguncha davom etadi. Shunday so'ng ikkinchi massivdagi elemenrtlar to'g'ridan – to'g'ri yangi massivga joylanadi.

ALGORITM *Merge (V[0..r-1], S[0..q-1], A[0..r+q-1])*
// tartibdangan ikkita massivni bittaga birlashtirish
// **Kiruvchi ma`lumotlar:** *V[0..r-*1] va *C[p..q-1]* massivlar
// **Chiquvchi ma`lumotlar:** tartiblangan *A[0..r+q-*1] massiv
$i \leftarrow 0; j \leftarrow 0; k \leftarrow 0$
while $i < r$ **and** $j < q$ **do**
 if B[i}<C[j]
  $A[k] \leftarrow B[i];\ i \leftarrow i+1$
 else
  $A[k] \leftarrow C[j];\ j \leftarrow j+1$
if $i = p$
  *C[j..q-*1] ni *A[k..p +q-*1] ga ko'chirilsin
else
  *B[i..p-*1] ni *A[k..p +q-*1] ga ko'chirilsin

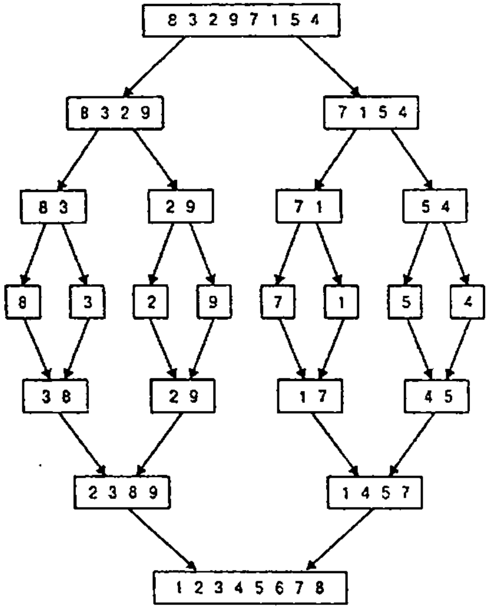
8, 3, 2, 9, 7, 1, 5, 4 sonlardan iborat ro'yhat uchun algoritmning ishlash jarayoni 6.2-rasmda tasvirlangan. Birlashtirib tartiblash algoritmi qay darajada samarali? Aytaylik, soddalik uchun n soni 2 ning darajasidan iborat bo'lsin. U holda bajariladigan rekurrent munosabat uchun taqqoslashlar soni $C(n)$

$$C(n) = 2C(n/2) + C_{merge}(n),\ n > 1$$

$$C(1) = 0;$$

ga teng bo'ladi.

Birlashtirish jarayonida kalitlarni taqqoslashlar soni bo'lgan $C_{merge}(n)$ ni tahlil qilib ko'raylik. Algoritmning har bir qadamida bitta taqqoslash bajariladi va shunday keyin birlashtirilayotgan massivlardagi elementlarning umumiy soni bittaga kamayadi. Eng yomon holda massivlarning birortasi ham boshqasida faqat bitta element qolmaguncha tugamaydi. Demak, Eng yomon hol uchun $C_{merge}(n) = n-1$. Bu bilan biz quyidagi rekkurent munosabatga ega bo'lamiz:



6.2-расм. Алгоритмнинг ишлаш жараёни

$$C_{eyh}(n) = 2C_{eyh}(n/2) + n - 1, \quad n > 1 \textit{ lar uchun},$$
$$C_{eyh}(1) = 0.$$

Birlashtirib tartiblash algoritmi bajariladigoan kalitlarni taqqoslashlar soni eng yomon hollarda tartiblashning ihtiyoriy algoritmi uchun o'rnatilgan taqqoslashlar soniga juda ham yaqin bo'ladi. Bu algoritmningg kamchiligi shundaki, uni amaliyotga tatbiq etish uchun qo'shimcha xotira talab qilinadi. Albatta, tartiblash masalasini qo'shimcha xotirasiz ham hal qilish mumkin, ammo xotirani tejashga bo'lgan urinish algoritm tezligiga salbiy ta`sir ko'rsatadi.

6.3. Tez saralash algoritmi. Bu algoritm (quicksort) - dekompozitsiya metodiga asoslangan yana bir muhim tartiblash algoritmlaridan biri sanaladi. Massiv elementlarining o'rinlarini e`tiborga olib massivlarga ajratish orqali birlashtirib saralashdan farqli o'laroq, tez saralash algoritmida massiv ularning qiymatlariga bog'liq ravishda kichik massivlarga ajratiladi. U berilgan $A[0..n-1]$ massivni kichik massivlarga ajratishda (partition) uning elementlari o'rnini shunday almashtiradiki, massivning barcha elementlari qandaydir s pozitsiyagacha $A[s]$ dan katta bo'lmaydi, s pozitsiyadan keyingi elementlar esa $A[s]$ dan kichik bo'lmaydi:

$$\underbrace{A[0]\dots A[s-1]}_{\text{barcha elementlar } \leqslant A[s]} \quad A[s] \quad \underbrace{A[s+1]\dots A[n-1]}_{\text{barcha elementlar } \geqslant A[s]}.$$

Tabiiyki, massivlarga ajratilgandan so'ng, $A[s]$ element deyarli yakuniy o'rniga joylashadi. Shundan keyin biz $A[s]$ gacha va undan keyin turgan ikkita qism massivlarni bir-biriga bog'liq bo'lmagan holda tartiblashimiz (aynan shu usulning o'zi bilan) mumkin.

ALGORITM *Quicksort (A[l..r])*

// Massivni tez saralash metodi bilan tartiblaydi

// **Kiruvchi ma`lumotlar:** A[0..*n*-1] ning *A*[l..*r*] qism massivi

// **Chiqurvchi ma`lumotlar:** tartiblangan A[l..r] massiv

if $l < r$

s←Partition(A[l..r]) // s - qism massivlarga ajratish pozitsiyasi

Quicksort(A[l..s-1])

Quicksort(A[s + 1..*r*])

A[0..*n*-1] massivni va umumiy holda uning $A[l..r]$ $(0<l<n-1)$ qism massivlarini quyidagicha ajratish mumkin. Dastlab qaysi elementga nisbatan ajratish amalini bajarish aniqlanadi. Bu elementni muhimligi sababli ***tayanch*** (pivot) deb ataymiz. Tayanch elementni tanlash uchun bir qator strategiyalar mavjud. Biz bu strategiyalarga algoritm samaradorligini tahlil qilish jaranyonida qaytamiz. Hozircha, eng sodda strategiyadan foydalanamiz, ya`ni qism massivning birinchi

elementini tayanch element sifatida tanlaymiz: $r=A[1]$.

Massivlarni qism massivlarga ajartishning bir qator protseduralari mavjud. Biz ana shu usullarning ichida qism massivlarni ikki marta (bir marta chapdan o'ngga, keyin o'ngdan chapga qarab) ko'rib chiqishga asoslangan samarali bir usuldan foydalanamiz. Bu o'tishlarning har birida joriy element tayanch element bilan taqqoslanadi. Chapdan o'ngga o'tish ikkinchi elementdan boshlanadi. Biz tayanch elementlar kichik bo'lgan elementlar qism massivning chap tomonida joylashishini hohlaganimiz uchun, birinchi o'tishda tayanch elmentlar kichik bo'lgan elementlarga "tegilmaydi" va birinchi katta element uchraganda ishdan to'xaydi. O'ngdan chapga qarab ko'rib chiqishda esa qarab chiqish o'ng tomondan boshlanadi. Bunda ham tayanch elementdan katta bo'lgan elementlarga "tegilmaydi" va o'tish tayanch elementdan kichik bo'lgan birinchi element topilganidan keyin to'htaydi.

Agar bu elementlarning indekslari kesishmasa, ya`ni $i<j$ bo'lsa, u xolda bu elementlarning o'rinlari almashtiriladi va jarayon i ni oshirgan, j esa kamaytirgan holda davom ettiriladi:

		$\rightarrow i$		$j\leftarrow$	
p	barcha elementlar $\leq p$	$\geq p$	...	$\leq p$	barcha elementlar $\geq$ p

Agar indekslar kesishib qolsa, ya`ni $i>j$ bo'lsa, u holda tayanch elemetni A[j] bilan almashtirgan holda ajratishni davom ettiramiz.

Vanihoyat, agar o'tish davrida olingan indekslar bitta elementda to'xtab qolsa, $i=j$ bo'lsa, u xolda bu elementning qiymati p ga teng bo'ladi. Demak, biz quyidagicha ajratilgan massiqlarga ega bo'amiz:

		$\rightarrow i=j\leftarrow$	
p	barcha elementlar $\leq p$	$=p$	barcha elementlar $\geq$ p

Oxirgi holatni indekslar kesishadigan hoalt bilan birlashtirish mumkin, buning uchun tayanch elementni $i\geq j$ bo'lganda $A[j]$ bilan almashtirishga to'g'ri keladi.

Ajratishning yuqorida tavsiflangan ajarayonini quyidagi psevdokod amalga oshiradi.

Algoritm Partition (*A* [*l*..*r*]) // qism to'plamga ajratish

// **Kiruvchi ma`lumotlar:** A[0..n-1] massiv

// **Chiqurvchi ma`lumotlar:** A[*l*..*r*]; // bunda ajratish pozitsiyasi

// funksiya qiymati sifatida qaytariladi

$i \leftarrow l;$

$j \leftarrow r + 1$

repeat repeat

$i \leftarrow r + 1$

until $A[i] \geq r$ **repeat**

until $A[j] \geq$ p

swap (A[*i*], A[*j*])

until $i \geq j$

swap (*A*[*i*], *A*[*j*]) //$i \geq j$ bo'lganda oxirgi almashtirish bekor qilinadi

swap(*A*[*i*], *A*[*j*])

return *j*

Shuni ta`kidlash joizki, bunday psevdokodda *i* indeksning ruhsat berilgan diapazondan chetga chiqishi mumkin. Har bir qadamda *i* indeksning chetga chiqqan yoki chiqmaganligini tekshirish o'rniga *A*[0..*n*-1] massivga "to'siq" qo'yish mumkin va u *i* indeksni chetga chiqib ketishiga yo'l qo'ymaydi. Shuni ham aytish mumkinki, qism massiv oxirida tayanch elementni tanlash bu to'siqni inkor qiladi.

Tez saralash algoritmi yordamida massiv elementlarini tartiblashga namuna 6.3-rasmda keltirilgan.

Tez saralash algoritmining samaradorligini agar massiv qism massivlarga ajratilguncha bajarilgan taqqoslashlar soni agar indekslar kesishadigan bo'lsa $n+1$ ga. agar ustma-ust tushsa n ga teng bo'ladi.

Agar barcha ajratma qism massivlar mos massivlarning qoq o'rtasida bo'lganda algoritm eng yaxshi holatga ega bo'ladi. Bu holda taqqoslar soni quyidagi rekkurent munosabatga teng bo'ladi:

$$C_{eyh}(n) = 2C_{eyh}(n/2) + n,\ \ n > 1,\ \ C_{eyh}(1) = 0\ .$$

Eng yomon holatda esa barcha qism massivlarning biri bo'sh, ikkinchisining o'lchami ajratilayotgan massiv o'lchamidan birga kichik bo'ladi. bunday holatlar o'sish tartibida berilgan massivlarda, ya`ni masala berilgan kiruvchi ma`lumotlar uchun to'g'ridan – to'g'ri yechilgan hollarda yuzaga keladi. Haqiqatdan ham, agar $A[0..n\text{-}1]$ – qat`iy o'suvchi bo`lib, tayanch element sifatida $A[0]$ element olinsa, u xolda elementlarni chapdan o'ngga qarab chiqish A[1] elementda, o'ngdan chapga qarab chiqish esa $A[0]$ da to'xtaydi va ajratish 0-chi pozitsiyadan amalga oshiriladi:

$\rightarrow i$			$j \leftarrow$
A[0]	A[1]	. . .	A[n-1]

Shunday qilib, $n+1$ ta taqqoslash va $A[0]$ elementni o'zini-o'zi bilan almashtirilganidan keyin m a`lum bo'ladiki, end qat`iy o'suvchi $A[1..n\text{-}1]$ massivni tartiblashga to'g'ri keladi. Qat`iy tartiblangan bunday massivlarni tartiblash oxirgi $A[n\text{-}2,\ n\text{-}1]$ elementgacha davom etadi. Bu holda taqqoslashlarning umumiy soni quyidagicha bo'ladi:

$$C_{eyh}(n) = (n+1) + n + \ldots + 3 = \frac{(n+1)(n+2)}{2} - 3 \in \Theta\left(n^2\right).$$

O'rtacha taqqoslarlar soni uchun quyidagi rekkurent munosabat hosil bo'ladi:

$$C_{avg}(n) = \frac{1}{n}\sum_{s=0}^{n-1}\left[(n-1) + C_{avg}(s) + C_{avg}(n-1-s)\right],\ \ n > 1$$

$$C_{avg}(0) = 0,$$

$$C_{avg}(1) = 0.$$

Bu munosabatning yechimlari sodda, ammo eng yaxshi va eng yomon holatlar uchun yetarlicha murakkab hisoblanadi.

Hulosa qilib aytish mumkinki,

$$C_{avg}(n) \approx 2n \ln n \approx 1{,}38 \log_2 n \,.$$

Ko'rinib turibdiki, tez saralash algoritmi o'rtacha holat uchun kalitlarni taqqoslash amalini eng yaxshi holdagiga qaraganda 38% ortiq bajaradi[4].

6.3. Ikkiga bo'lib izlash algoritmi. Ikkiga bo'lib izlash algoritmi tartiblangan massivdan izlashning enrg yaxshi usuli hisoblanadi. Bu algoritm quyidagi g'oya ostida ishlaydi. Berilgan K kalitni massivning o'rtasida joylashgan $A[m]$ element bilan taqqoslaydi. Agar ular teng bo'lsa, algoritm o'z ishini to'xtatadi. Aks holda algoritm izlashni agar n<$A[m]$ bo'lsa chap qism massivdan, n>$A[m]$ bo'lganda o'ng qism massivdan rekursiv asosda davom ettiradi. Buning ma`nosi shuki, kalit o'zi mavjud bo'lishi mumkin bo'lgan qism massivdan izlanadi. Izlashning har bir qadamida qism massiv elementlari soni avvalgi massiv elementlari sonining yarmiga teng bo'ladi, ya`ni izlash oralig'i ikki marta kichrayadi. Taqqoslashlarning umumiy soni $\log_2 n$ ga teng bo'ladi. Qiyoslash uchun, 1000 ta elementli massivdan uzog'i bilan 10 ta taqqoslash yordamida izlangan kalitning mavjud yoki mavjud emasligini aniqlash mumkin. Namuna tariqasida $K = 70$ kalitni quyidagi massivdan izlab ko'raylik:

3	14	27	31	39	42	55	70	74	81	85	93	98

Algoritmning ishlash jarayoni (iteratsiyasi) quyidagi jadvalda keltirilgan.

---

[4] Левитин А. И.

Indexlar	0	1	2	3	4	5	6	7	8	9	10	11	12
Qiymatlar	3	14	27	31	39	42	55	70	74	81	85	93	98
1-chi iteratsiya	*l*						*m*						*r*
2-chi iteratsiya								*l*		*m*			*r*
3-chi iteratsiya								*l,m*	*r*				

Ikkiga bo'lib izlash algoritmi rekursiv hisoblansada, uni norekursiv usul bilan ham amalga oshirish mumkin. Quyida norekursiv algoritm psevdokodi keltirilmoqda.

Algoritm *Binary_Search (A [0..*n-1], *K)* // norekursiv izlash

// **kiruvchi ma`lumotlar:** o'sish tartibidagi *A[0..n*-1] massiv

// izlashning K kaliti

// **Chiquvchi ma`lumotlar:** ***K*** ga teng bo'lgan element indeksi,

// agar bunday element mavjud bo'lmasa -1

$l \leftarrow 0;$

$r \leftarrow p\text{-}1;$

while l $\le$ ***r do***

$t \leftarrow \lfloor (l+r)/2 \rfloor$

if $K = A[t]$ ***return m***

else if $K < A[t]$

i ←t-1 else

$l \leftarrow m + 1$

return -1

Yuqoridagi algoritm samaradorligini tahlil qilishning standart usuli izlangan kalitni massiv elementlari bilan taqqoslashlar sonini aniqlashdan iborat bo'ladi. bunda uch karra taqqoslash amalga oshirilayotganligini (*K* va *A[m]* larni taqqoslash *K* ning *A*[*m*] dan kichik, katta yoki tengligini aniqlashga yordam berishini) esdan chiqarmaslik lozim.

Massiv elementlari *n* ta bo'lganda taqqoslashlar soni qanday bo'ladi? Javob nafaqat *n* ga, balki konkret massivga ham bog'liq bo'ladi. Eng yomon holat massivda izlangan kalit mavjud bo'lmagan holda sodir bo'ladi. Bitta taqqoslash bajarilganidan keyin masala yana shu masalaning o'ziga qaytadi, faqat keyingi massiv avvalgisiga qaraganda ikki marta kichik bo'ladi. Shuning uchun taqqoslashlar soni

$$C(n) = 2C\lfloor n/2) \rfloor + 1,\ n > 1,$$
$$C\ (1) = 1.$$

Ushbu rekkurent munosabatni hal qilish uchun $n = 2^k$ deb olish va tenglamani teskarisidan qo'yish usuli bilan yechishga to'g'ri keladi. Bunda

$C(2^k) = k + 1 = \log_2 n + 1$ ekanligini yodda tutish lozim.

7-§. Rekurrent munosabatlar metodi

Agar algoritm o'zini-o'zi yordamchi algoritm sifatida foydalanadigan bo'lsa, bunday algoritmlar rekursiv deyiladi.

Rekursiv algoritmlar ikki turga bo'linadi:

a) to'g'ri rekursiya. Bunda algoritm o'ziga-o'zi murojaat qiladi.

b) yondosh rekursiya. Bunda *A* algoritm *B* ga, *B* algoritm *A* ga murojaat qiladi.

Rekursiv algoritm yozish uchun avvalo quyidagi shartlat o'rinli bo'lishi zarur:

1) rekkurent munosabat aniqlangan bo'lishi;

2) shu munosabat uchun boshlang'ich holatning mavjud bo'lishi.

Rekkurent munosabat deganda qaralayotgan jarayonga doir muayyan bosqichlarni avvalgi bosqichlar bilan bog'lovchi munosabatlar tushuniladi. Masalan, $N! = N \cdot (N-1)!$ formulani $N!$ uchun rekkurent munosabat deb qarash mumkin. Boshlang'ich holat esa 1!=1 bo'ladi.

Rekkurent munosabatlar metodining g'oyasi yetaricha sodda bo'lib, qo'yilgan masala uchun qandaydir mulohazalar yordamida qo'yilgan masala o'lchamlari kichikroq bo'lgan nusxalari orqali ifodalash talab qilinadi. Bu holda har bir nusha-masala o'zining nushasi uchun asosiy masalaga aylanadi. Mazkur jarayon davom ettirilib, masalaning yechimi ma`lum bo'lgan eng kichik o'lchamli nushasi qurilmaguncha davom ettiriladi. Bunday holatni rekkurent munosabatlar usulida "algoritmni o'z ichiga cho'kishi" deb ataladi. Eng kichik o'lchamli masala nushasi "cho'kish" jarayonini to'htatish uchun xizmat qiladi, chunki odatda bunday nushaning yechimi ma`lum bo'ladi. Shundan keyin eng kichik nusha-masalaning yechimi ma`lum bo'lgandan foydalanib, unga asosiy bo'lgan masalaning yechimi topiladi. Bu yechim navbatdagi bosqich asosiy masalasini hal qilish uchun poydevor bo'lib hizmat qiladi. Umuman aytganda, *i*-chi bosqichdagi asosiy masala *i*-1-chi bosqich yechimidan foydalangan holda topiladi. Bu jarayonni "algoritmni suzib chiqishi" tarzida tavsiflanadi.

Yuqoridagi ma'lumotlarni hisobga olsak, faktorialni hisoblash masalasi uchun rekkurent va boshlang'ich munosabatlar quyidagicha bo'ladi:

$$N! = \begin{cases} N \cdot (N-1)!, & agar \;\; N > 1 \\ 1, & agar \;\; N = 1. \end{cases}$$

Ko'rinib turibdiki, *N*! ni hisoblash uchun *(N-1)*! ma'lum bo'lishi kerak. Lekin, $(N-1)! = (N-2)! \cdot (N-1)$ bo'lgani uchun o'z navbatida $(N-2)!$ *(N-2)*! ni topish talab qilinadi. (*N*-2)! esa (*N*-3)!·(*N*-2) ga teng va hokazo. Bu yerda *N*! ni hisoblash algoritmi o'zining ichiga o'zi "cho'kib" borishi hodisasi ro'y bermoqda. Cho'kish jarayoni boshlang'ich holat sodir bo'lgunga qadar, ya'ni 1! gacha davom etadi. Shundan keyin, "cho'kish" jarayoni to'xtaydi, $1! = 1$ ekanligi haqida ko'rsatma olgan kompyuter yuqoriga qarab "suzib" chiqish bosqichini boshlaydi. Ya'ni, 2!=1, 2!=1!·2=2, 3!=2!·3=6 va hokazo. Bu holat to *N*! hisoblanmaguncha davom etaveradi.

Yuqorida keltirilgan rekursiv algoritmning psevdokodi quyidagicha yoziladi:

```
algoritm fak(int m)
// Kiruvchi ma`lumotlar: natural m soni
// Chiquvchi ma`lumotlar: qiymati m! ga teng bo'lgan son
 if (m=1) then fak←1
        else fak←fak(m-1)*m;
 return fak;
```

N=4 uchun "cho'kish" va "suzib chiqish" jarayoni quyidagicha:

N ning qiymatlari	4 4 3 2 1	Rekursiyadan chiqish
N=1 sharti natijasi	yo'q yo'q yo'q ha	
	fak(3)*4 fak(2)*3 fak(1)*2 1	$p := p$*4=24 $p := p$*3=6 $p := p$*2=2 p:=1

Tabiiyki, ushbu masalani rekursiyasiz ham hal qilish mumkin. Biz rekkurent munosabatlar metodini o'quvchilarga yaxshi tanish bo'lgan shu misol orqali bayon etsak, uning mohiyatini tushunish qiyin bo'lmaydi degan mulohazaga asoslandik xolos.

Qo'yilgan bu masala uchun bazaviy amal $fak(n-1)*n$ dan iborat bo'lib, algoritm samaradorligi faktorialni oddiy usulda hisoblashdan ortiqcha farq qilmaydi.

Shunday masalalar sinfi mavjudki, ularni rekursiyadan foydalanmay turib yechishning boshqa samarali usullari yo'q.

7.1. Ketma-ketlikning n-chi xadini hisoblash. $f(n)$ funksiyaning qiymatlari $f(0)=1$, $f(2n)=f(n)$ va $f(2n+1)=f(n)+1$ ifodalar yordamida topiladi. Berilgan k natural soni uchun $f(k)$ ni toping.

Yechish g'oyasi. Yuqoridagi masalani k ta elementli massiv yordamida yechish ko'pchilikning nazarida oson usulga o'xshaydi.

Lekin bu to‘g‘ri emas. Chunki, k yetarlicha katta son bo‘lsa, k ta elementli massiv kompyuter xotirasiga sig‘may qolishi mumkin. Qolaversa, siqqan taqdirda ham, bu elementlarning hammasidan foydalanilmaydi. Masalan, $k=1000$ bo‘lganda bu masalani yechish uchun massivning 1000 ta elementidan ko‘pi bilan 11 tasi kerak bo‘ladi (nima uchunligini o‘ylab ko‘ring), qolganlari esa kompyuter xotirasini befoyda band qiladi. Bunda xotiradan noo’rin foydalanish holati yuz beradi va u keyinchalik salbiy oqibatlarga olib kelishi mumkin. Shuning uchun qo‘yilgan masalani massivdan foydalanmay yechish eng yaxshi usullaridan biri rekursiya mexanizmidan foydalanishni nazarda tutadi.

Zarur bo‘lgan rekkurent munosabat va boshlang‘ich holatlarning masala shartida keltirilganligi ishni yanada osonlashtiradi.

Mazkur masala uchun ishlab chiqilgan algoritmning psevdokodi quyidagicha yoziladi:

ALGORITM fun(int m)

// **kiruvchi ma`lumotlar:** m natural soni

// **Chiquvchi ma`lumotlar:** $f(n)$ funksiyaning qiymati

if *(m=0)* **then** *f←1*

else

h←m/2;

if *(m / 2=0)* **then** *fun←fun(h)*

else *fun←fun(h)+1;*

return f;

Yuqoridagi ma’lumotlardan ko‘rinib turibdiki, rekursiya oddiy protsedura yoki funksiyaga nisbatan murakkabroq tushuncha, lekin u mohir dasturchi qo‘lida juda ham yaxshi vositaga aylanishi mumkin.

7.2. Natural sonni qo’shiluvchilarga ajratish masalasi. M natural soni berilgan bo’lsin. Uni qo’shiluvchilarga ajratishlarning umumiy soni topilsin.

M natural sonini qo’shiluvchilarga ajratish deganda uni musbat natural sonlarning yig'indisi tarzida ifodalash nazarda tutiladi. Quyida namuna uchun $M = 6$ bo'lgan hol uchun yig'indilar keltirilmoqda. Bu yig'indilarni sanab chiqish qo'yilgan masala yechimini beradi.

$6;$

$5+1;$

$4+2,\ 4+1+1;$

$3+3,\ 3+2+1,\ 3+1+1+1;$

$2+2+2,\ 2+2+1+1,\ 2+1+1+1+1;$

$1+1+1+1+1+1.$

Mazkur masalani hal qilish uchun qo'yiladigan birinchi qadam $P(m)$ funksiyani $Q(m, n)$ funksiya orqali ifodalashdan iborat bo'ladi. $Q(m, n)$ funksiya natural m sonini n dan katta bo'lmagan qo'shiluvchilarga ajratishlar sonini ko'rsatadi. Agar ihtiyoriy m argument uchun $Q(m, n)$ funksiyani hisoblay olsak, u holda bu funksiya ***P(m)*** ni ifodalayi, chunki $P(m) = Q(m, m)$.

Ikkinchi qadamda metod g'oyasiga ko'ra $Q(m, n)$ funksiya qiymatini bevosita hisoblashga yordam beradigan argumentlarning qiymatlarini hisoblanadi. Shuningdek, $Q(m, n)$ funksiyani oz'idan avvalgi qiymatlari orqali hisoblash qinunuyati belgilanadi.

Yuqoridagi mulohazalar quyidagi 5 ta tenglamaga olib keladi:

1. $Q(m, 1) = 1$, chunki m sonini eng katta qo'shiluvchisi 1 ga teng bo'lgan qo'shiluvchilarga ajratish usuli faqat bitta, ya'ni $m = 1+1+...+1$.

2. $Q(1, n) = 1$, bu tabiiy, chunki 1 sonini faqat bitta usul bilan qo'shiluvchilarga ajratish mumkin holos.

3. $Q(m, n) = Q(m, m)$, $m < n$. Chunki m ning hech bir yoyilmasida m dan katta bo'lgan n qo'shiluvchilarning bo'lishi mumkin emas.

4. $Q(m, m) = 1 + Q(m, m-1)$, ya'ni, m sonini m ga teng bo'lgan qo'shiluvchili yoyilmasi faqat bitta bo'ladi, m boshqa hamma yoyilmalarida eng katta qo'shiluvchi $n<m-1$.

5. $Q(m, n) = Q(m, n-1) + Q(m-n, n)$. Bu mulohaza rekursiyaning asosini tashkil etadi. Unga ko'ra m sonining eng katta qo'shiluvchisi n ga teng yoki kichik bo'lgan yoyilmasi yoki qo'shiluvchi sifatida n ni o'z ichiga olmaydi (bu holda mazkur yoyilma $Q(m, n-1)$ funksiya yordamida hisoblanadi) yoki n ni o'z ichiga oladi (ammo bu holda qolgan qo'shiluvchilar $m-n$ soni yoyilmasini tashkil qiladi va $Q(m-n, n)$ funksiya yordamida hisoblanadi.

Yuqorida keltirilgan tenglamalar asosida rekursiv $Q(m, n)$ funksiyani quyidagi rekkurent munosabatlar orqali ifodalash mumkin:

$$\begin{cases} Q(m, 1) = 1; \\ Q(1, n) = 1; \\ Q(m, n) = Q(m, m), \ m \le n; \\ Q(m, m) = 1 + Q(m, m-1), \ n = m; \\ Q(m, n) = Q(m, n-1) + Q(m-n, n). \end{cases}$$

Yuqorida bayon etilgan rekkurent munosabatlar asosida $Q(m, n)$ protseduraga (m, m) argumentlar uchun murojaat qiladigan rekursiv algoritm quyidagicha yoziladi.

Algoritm qush_ajratish Q(m, n)
// **kiruvchi ma`lumotlar**: m va n natural sonlar
// **Chiquvchi ma`lumotlar:** $Q(m, m)$
if (m=1) **or** (n=1) // rekursiya to'xtashini nazorat qilish
then Q←1 // Q ni to'g'ridan – to'g'ri hisoblash
else if (m<n)
  then Q←Q(m, m) // m<n bo'lganda rekursiv murojaat
else if (m=n)

then Q←1+Q(m, m-1) //m=n, rekursiv murojaat
else Q←Q(m, n-1)+Q(m-n, n) //rekursiv murojaat
return (Q)

Shuni ta`kidlash joizki, $Q(m, n)$ funksiya ikkita argumentga ega bo'lgani uchun, algoritm tarkibidagi uchta turli hil rekursiv murojaatlar yuzaga kelishi mumkin: argumentlarning tengligi, birinchining ikkinchi argumantdan kichik yoki katta bo'lishi. Bu holatlar ikki argumentli funksiyalarning umumiy hususiyati emas, balki $Q(m, m)$ funksiya va demak, qo'yilgan masalaning o'ziga hos hulqini aks ettiradi.

Rekursiv algoritm holi uchun mazkur algoritmning samaradorligini baholash mushkul masala. Hususiy hollarda, masalan, Fibonachi sonlari uchun algoritm samaradorligi yetarlicha past bo'ladi. Faktoriallarni rekursiv algoritm yordamida hisoblashda algoritm samaradorligi asimptotik bahoga ega bo'ladi.

8-§. Masala o'lchamlarini pasaytirish metodi

Masala o'lchamlarini pasaytirish ("kichraytir va hukmronlik qil") metodi berilgan masala va o'lchami unga qaraganda kichikroq bo'lgan masala nusxasi o'rtasidagi bog'lanishlarga asoslanadi. Agar shunday bog'lanish mavjud bo'lsa, undan yuqoridan quyidagi (rekursiv asosda) yoki quyidan yuqoriga (norekursiv) tarzida foydalanish mumkin.

Masala o'lchamlarini pasaytirishning uch hil usuli mavjud:

- o'zgarmas miqdorga pasaytirish;
- o'zgarmas ko'paytuvchi miqdorida pasaytirish;
- o'zgaruvchan miqdorga pasaytirish.

O'zgarmas miqdorga pasaytirishda masala joriy nusxasining o'lchami algoritmning har bir iteratsiyasida o'zgarmas miqdorga pasayadi. Odatda bu miqdorga 1 ga teng bo'ladi.

Misol uchun a^nni hisoblash masalasini olaylik. Bu masalada o'lchovlari n va n-1 bo'lgan masala nusxalari $a^n = a^{n-1} \cdot a$ munosabat orqali bog'lanadi. Shuning uchun $f(n) = a^n$ ni hisoblash masalasi quyidagi rekkurent munosabat yordamida hal qilinishi mumkin:

$$f(n) = \begin{cases} f(n-1) \cdot n, & agar\ n > 1, \\ 1, \ agar\ n = 1. \end{cases}$$

Masala o'lchamini o'zgarmas ko'paytuvchiga pasaytirish metodi algoritmning har biri iteratsiyasida bir hil o'zgarmas ko'paytuvchi miqdorga pasaytirishni nazarda tutadi.

Namuna uchun yana darajaga ko'tarish masalasini olish mumkin. Bunda o'lchami n bo'lgan masala (a^n) o'lchami uning yarmiga teng bo'lgan $a^{n/2}$ masala bilan $a^n = \left(a^{n/2}\right)^2$ munosabati orqali bog'langan. Agar n soni toq bo'lsa, $\left(a^{(n-1)/2}\right)^2$ ni a ga ko'paytirish lozim bo'ladi. Shuning uchun bu masalani hal qilishda quyidagi munosabatdan foydalanish mumkin:

$$a^n = \begin{cases} \left(a^{n/2}\right)^2, & agar\ n\ musbat\ va\ juft, \\ \left(a^{(n-1)/2}\right)^2 \cdot a, & agar\ n\ toq\ va\, 1\ dan\ katta, \\ a, & agar\ n = 1. \end{cases}$$

Masalani mazkur formula bilan hal qilishda algoritm samaradorligi $O(log_2 n)$ ga teng bo'ladi.

O'lchamini o'zgaruvchan miqdorga pasaytirish metodida masala o'lchami algoritmning har bir qadamida turli miqdorlarga pasaytiriladi. Misol sifatida EKUB uchun Yevklid algoritmini ko'rish mumkin. Bu masala $Ekub(m,n) = Ekub(n,\ m \bmod n)$formulasiga asoslanadi.

8.1. Orasida qo'yib tartiblash masalasi.

$A[0..n\text{-}1]$ massivni tartiblash talab qilingan bo'lsin. Bu masalani o'lchamini pasaytirib yechishga xarakat qilamiz.

Faraz qilaylik, berilgan massivning dastlabki n-2 ta elementlari tartiblangan bo'lsin: A[0]≤...A[n-2]. Ular orasiga o'sish tartibini buzmagan holda $A[n\text{-}1]$ elementni joylashtirish talab qilinadi. Buning uchun massivning tartiblangan qismiga tartibni buzmagan holda, $A[n\text{-}1]$ element uchun kerakli pozitsiyani topib joylashtirish lozim bo'ladi.

Bu masalani oddiy tartiblash yoki binar izlash algoritmi yordamida hal qilish mumkin.

Orasiga qo'yib tartiblash algoritmini rekursiv qurish mumkin. Bu usul tartiblashni quyidan yuqoriga qarab tashkil qilgani uchun boshqalaridan samaraliroq hisoblanadi. Har bir iteratsiyada $A[i]$ element (birinchisidan boshlab to n-1 gacha) o'zi uchun mos pozitsiyaga joylashtiriladi, ammo bu o'rin bu element uchun yakuniy bo'lmaydi va algoritmning navbatdagi qadamlarida o'zgarishi mumkin.

$$\underbrace{A[0] \le \dots A[j]}_{A[i]\text{ dan kichik yoki teng}} < \underbrace{\overbrace{A[j+1] \le \dots A[i-1]}_{}}_{A[i]\text{ dan katta}} \le A[i] \backslash \le \dots \le |A[n-1]$$

Quyida shu algoritmning psevdokodi keltirilgan.

ALGORITM *Insertionsort (A[0..n* - 1])

// **Kiruvchi ma'lumotlar:** n ta elementli $A[0..n\text{-}1]$ massiv

// **Chiquvchi ma'lumotlar:** tartiblangan A[0..n-1] massiv

for i←1 to n - 1 do

v← $A[i]$

j←i - 1

while $j \ge 0$ **and** $A[j] > v$ **do**

$A[j + 1]$←$A[j]$

$j \leftarrow j-1$

$A[j+1] \leftarrow v$

Algoritmning ishlash jarayoni 8-1 rasmda tasvirlangan. Unda vertikal chiziq massivning hozirgacha tartiblangan qismini anglatadi. Orasiga qo'yiladigan element rasmda qoraytirib ko'rsatilgan.

89 |	**45**	68	90	29	34	17
45	89 |	**68**	90	29	34	17
45	68	89 |	**90**	29	34	17
45	68	89	90 |	**29**	34	17
29	45	68	89	90 |	**34**	17
29	34	45	68	89	90 |	**17**
17	29	34	45	68	89	90

8.1-rasm. Orasiga qo'yib tartiblash algoritmiga misol.

Algoritm uchun $A[j] > v$ taqqoslash amali bazaviy hisoblanadi. Bu amallar soni kiruvchi ma`lumotlar tabiatiga ham bog'liq bo'ladi. Eng yomon holat, ya`ni berilgan massiv elementlari kamayish tartibida joylashganda $A[j] > v$ taqqoslashlar soni eng katta bo'ladi. bu holda taqqoslashlarning umumiy soni $C(n) = \frac{n(n-1}{2} \in \Theta(n^2)$ ga teng bo`ladi.

Eng yaxshi holatda tashqi tsiklning har bir iteratsiyasida $A[j] > v$ taqqoslash bir marta bajariladi. Bu holat boshlang`ich massiv elementlari oldindan tartiblangan bo'lsa sodir bo'ladi va bu holda taqqoslashlar soni $C(n) = n-1 = \Theta(n)$ bo'ladi.

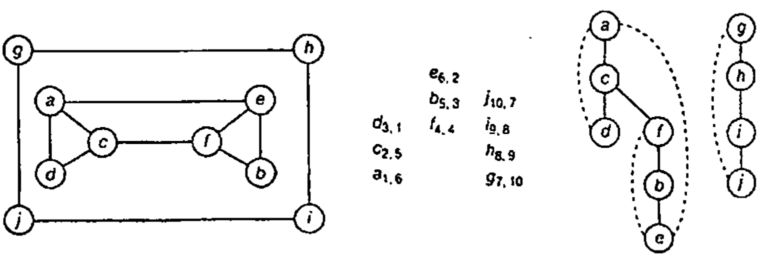
Algoritm samaradorligi o'rtacha taqqoslashlar soniga ko'ra baholanadi. Bu miqdorni boshlang'ich massiv elementlari tartibsiz beriladigan hollarda tahlil qilinadi. Bunday vaziyatlarda taqqoslashlar soni taxminan eng yomon holga qaraganda ikki marta ko'p bajariladi, ya`ni $C(n) = \frac{n^2}{4} \in \Theta(n^2)$ ga teng bo'ladi.

8.2. Ichkariga va eniga qarab izlash masalalari

Ichkariga qarab izlash (yoki aylanib chiqish) algoritmi oriyentirlangan va oriyentirlanmagan graflarni ko'rib chiqib, boshlang'ich uchdan boshlab mumkin bo'lgan barcha uchlarni aylanib chiqishni nazarda tutadi. Algoritm g'oyasiga ko'ra grafning ihtiyoriy bir uchini ko'rib chiqilgan deb belgilab qo'yiladi. So'ngra algoritmning har bir qadamida joriy uch bilan qo'shni bo'lib, hozircha ko'rib chiqilmagan uchlar qayta ishlanadi. Agar bunday uchlar bir nechta bo'lsa, ko'rib chiqsh uchun ularning ihtiyoriy birini tanlab olish mumkin. Bu jarayonda agar "boshi berk" uchga to'g'ri kelib qolinsa, u holda joriy uchni ko'rib chiqilmagan uchlar qatoriga qo'shiladi va avvalgi uchga "qaytiladi". Jarayon barcha uchlarni aylanib chiqilganidan keyin tugaydi.

Bunday jarayonlarni tashkil qilishda steklardan foydalanish (ko'rib chiqilgan uchlarni steklarga joylash, "boshi berk" uchlarni stekdan olib tashlash qabilida) maqsadga muvofiq hisoblanadi.

Ichkariga qarab izlashda graf uchlarini aylanib chiqish jarayonini o'rmon tarzida ifodalash qulay hisoblanadi. Grafning boshlang'ich uchi o'rmondagi birinchi daraxtning ildizi bo'lib hizmat qiladi. Ko'rib chiqilgan uchlarni bog'lovchi qirralar daraxt qirrasi deb ataladi.



a) b) c)

8.2-rasm. *a)* graf, *b)* aylanib chiqish stegi (birinchi indeks stekka kiritish tartibi, ikkinchi indeks boshi berk yo'llar), *c)* ichkariga qarab izlash o'rmoni, orqaga qaytuvchi qirralar punktir chiziq.

Ichkariga qarab izlash algoritmining psevdokodi quyidagicha yoziladi:

ALGORITM *DFS (G)*

// **Kiruvchi ma`lumotlar:** *G*= (*V*, *E*) grafi

// **Chiquvchi ma`lumotlar:** aylanib chiqish tartibiga mos graf

// barcha uchlarni 0 bilan belgilaymiz (ko'rib chiqlmagan uchlar)

count ← 0

for *V* dan olingan har bir *v* uch (uchun) do

if *v* uch 0 bilan belgilangan bo'lsa

dfs(v)

dfs(v)

// *v* bilan bog'langan barcha qarab chiqilmagan uchlarni rekursiv

// ko'rib chiqiladi va tartibini *count* da saqlab qolinadi:

count ←*count* + 1;

// *v* ni *count* soni bilan belgilab qo'yiladi

for *V* dan olingan va *v* ga qo'shni bo'lgan har bir *w* (uchun)

do if *w* 0 bilan belgilangan

dfs(w)

DFS protsedurasining qisqa va soddaligi mazkur algoritmning murakkabligi haqidagi yolg'on tasavvurlarni uyg'otadi. Algoritmning haqiqiy quvvat va chuqurlik (ichkarilash) darajalarini baholash uchun qo'shnilik matritsasi yoki qo'shni uchlarning bog'langan ro'yhatlaridan foydalanib, qadam-baqadam ko'rib chiqishga to'g'ri keladi.

Algoritm samaradorligini baholashda shuni e`tiborga olish kerakki, uning ishlash vaqti grafni ifodalash uchun qo'llanilgan ma`lumotlar strukturasi o'lchamiga proportsional bo'ladi. Agar graf qo'shnilik matritsasi tarzida ifodalangan bo'lsa, uni aylanib chiqish algoritmining samaradorligi $O(|V|^2)$, bog'langan ro'yhatlardan foydalangan holda esa $O(|V|+|E|)$ ga teng bo'ladi, bu yerda $|V|$ hamda $|E|$ lar mos ravishda grafning uchlari va qirralari.

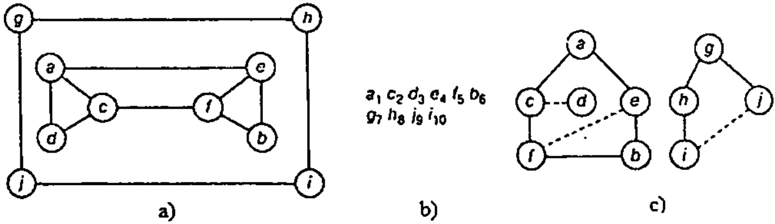
Ichkariga qarab izlash va grafni o'rmon tarzida ifodalash graflarning ko'plab muhim hususiyatlarini tekshirib ko'rish bilan bog'langan'liq masalalarda qo'l kelishi mumkin. Shuni ta`kidlash joizki, ichkariga qarab izlash algoritmi uchlarning ikki hil tartibini ko'rsatadi: uchlarni aylanib chiqishda birinchi marta o'tiladigan uchlar hamda duch kelgan boshi berk yo'llar tartibi.

Eniga qarab izlash algoritmi. Bu algoritm ham avvalgi masala uchun qo'llanadi. Uning g'oyasi bo'yicha, dastlab boshlang'ich uch bilan qo'shni bo'lgan barcha uchlar qarab chiqiladi. So'ngra boshlang'ich uch bilan ikkita qirra orqali bog'langan uchlar qarab chiqiladi va h.k. Jarayon boshlang'ich uch bilan bog'langan hamma uchlarni qarab chiqilguncha davom etadi.

Eniga qarab izlash uchun navbatlardan foydalanish maqsadga muvofiq. Navbat boshlang'ich nuqta bilan initsializiya qilinadi va uni qarab chiqilgan uchlar qatoriga qo'shiladi. Har bir iteratsiya qadamida algoritm navbat boshida turgan uch bilan qo'shni bo'lgan, ammo hozircha qarab chiqilmagan barcha uchlarni aniqlaydi va ularni ko'rib chiqilgan uchlar sifatida belgilaydi, shundan keyin navbatdan boshda turgan uch o'chirib tashlanadi.

Eniga qarab izlash jarayonini ham izlash o'rmoni sifatida ifodalash maqsadga muvofiq hisoblanadi. bo'ladi. Izlash jarayonida birinchi uchragan ko'rib chiqilmagan uch o'ziga o'tilgan uch bilan qirra yordamida birlashtiriladi va uni daraxt qirrasi deb ataladi. O'zidan avvalgi qirradan farq qiluvchi uchlarga olib boruvchi qirralarni ko'ndalang qirralar deb ataladi.

Quyida eniga qarab izlash algoritmininng psevdokodi keltirilgan.



8.3-rasm. Eniga qarab izlashga namuna. *a)* graf, *b)* izlash navbati (indekslar ko'rib chiqilgan uchlar navbatini anglatadi), *c)* Izlash o'rmoni

ALGORITM *BFS* (G)

// **Kiruvchi ma`lumotlar:** *G* q (V, ye*)* grafi

// **Chiquvchi ma`lumotlar:** qarab chiqish tartibiga mos graf

// *V* ning har bir uchini 0 ga teng, deb belgilymiz

` *count*←O

for *V* dan olingan har bir c uch (uchun) **do**

if *v* uch) bilan belgilangan bo'lsa

bfs(v)

bfs(v)

// *v* bilan bog'langan'langan barcha uchlarni aylanib chiqish va

// ularga qarab chiqish tartib nomerini global *count*

// o'zgaruvchishiga berish

count←*count*+ 1

v ga *count* ninh qiymatini berish hamda

v ning navbatdagi uchini aniqlash

while (toki) navbat bo'sh emas **do**

for *V* dan olingan va navbat boshida turgan uch bilan

qo'shni bo'lgan har gir *w* uch (uchun) **do**

if *w* uch 0 bilan belgilangan bo'lsa,

count←*countQ*1

w ga *count* qiymatini bberamiz.

w ni navbatga qo'shamiz.

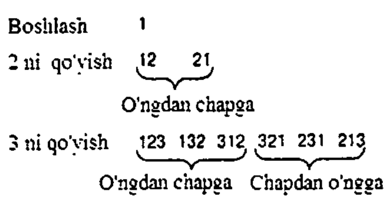
Navbatdan birinchi turgan uchni o'chiramiz

Eniga qarab izlash algoritmining samaradorligi ham ichkariga qarab izlash algoritmi samaradorligi bilan bir hil.

8.3. Kombinatorik ob`ektlarni generatsiya qilish algoritmlari

O'rin almashtirishlarni generatsiya qilish. Faraz qilaylik, 1 dan n gacha bo'lgan sonlarning barcha o'rin almashtirishlarini topish talab qilingan bo'lsin. Umumiy holda bu sonlar $\{a_1, ..., a_n\}$ massiv elementlarining indekslari ham bo'lishi mumkin.

$\{1, \ldots, n\}$ sonlarning barcha o'rin almashtirishlarini topish masalasiga o'lchamlarni pasaytirish usulini qo'llaymiz. Aytaylik, (1, ..., n-1) sonlar uchun mumkin bo'lgan barcha $(n-1)!$ ta o'rin almashtirishlar topilgan bo'lsin. U holda berilgan masalani yechish uchun (1, ..., n-1) sonlarning har bir o'rin almashtirishidagi mumkin bo'lgan barcha pozitsiyalariga n sonini yozib chiqamiz. Bunda o'rin almashtirishlarning umumiy soni $n\cdot(n-1)!=n!)$ ga teng bo'ladi. Biz n sonini ilgari tashkil qilingan o'rin almashtirishlarga chapdan o'ngga yoki o'ngdan chapga qarab joylashtirishimiz mumkin. Tahlillar shuni ko'rsatdiki, n sonini 1, 2, ... (n-1) ketma-ketlikka o'ngdan chapga qarab joylashtirish va (1...n-1) to'plamning yangi o'rin almashtirishiga o'tilganda joylashtirish yo'nalishini o'zgartirish samarali bo'lar ekan (8.4-rasmda namuna keltirilgan).



8.4-rasm.

Bunday o'rin almashtirishlarning afzalligi uning minimal o'zgarishlar talabini qanoatlantirishi bilan belgilanadi. Ular o'zidan avvalgi o'rin almashtirishdagi ikkita element o'rinlarini almashtirish orqali hosil qilinadi.

O'rin almashtirishlarni bunday tartibini n ning kichikroq qiymatlari uchun o'rin almashtirishlarni oshkor generatsiya qilmasdan ham topsa bo'ladi. Buning uchun har bir elementni o'rin almashtirish yo'nalishiga bog'lashga to'g'ri keladi. Yo'nalishni qaralayotgan element ustiga strelka qo'yib ko'rsatiladi, masalan: $\overleftarrow{3}\ \overleftarrow{2}\ \overrightarrow{4}\ \overleftarrow{1}$. Agar strelka o'zining kichik qo'shinisini ko'rsatsa, bunday o'rin almashtirishdagi k elementni mobil deb ataladi. Masalan, $\overleftarrow{3}\ \overleftarrow{2}\ \overrightarrow{4}\ \overleftarrow{1}$ o'rin almashtirish- dagi 4 soni mobil hisoblanadi. Mobil element tushunchasidan foydalanib o'rin almashtirishlarni generatsiya qilish uchun Djonson- Tritter algoritmiga ega bo'lish mumkin.

ALGORITM *JohnsonTrotter (n)*

// **Kiruvchi ma`lumotlar:** n natural soni

// **Chiquvchi ma`lumotlar:** $\{1..n\}$ o'rin almashtirishlar ro'yxati

Birinchi o'rin almashtirishni initsializatsiya qilamiz: $\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3}\ \overleftarrow{4}$

while (toki) k mobil soni mavjud bo'lsa **do**

eng kichik mobil k soni topiladi

k va uning strelka ko'rsatayotgan qo'shnisi o'rnini almashtiramiz

k dan katta bo'lgan barcha element strelkalarini o'zgartiramiz.

$n=3$ uchun mazkur algoritmning ishlash jarayonidagi mobil element qalin shrift bilan ko'rsatilgan:

$\overleftarrow{1}\,\overleftarrow{2}\,\overleftarrow{\mathbf{3}}\quad \overleftarrow{1}\,\overleftarrow{\mathbf{3}}\,\overleftarrow{2}\quad \overleftarrow{3}\,\overleftarrow{1}\,\overleftarrow{\mathbf{2}}\quad \overrightarrow{\mathbf{3}}\,\overleftarrow{2}\,\overleftarrow{1}\quad \overleftarrow{2}\,\overrightarrow{\mathbf{3}}\,\overleftarrow{1}\quad \overleftarrow{2}\,\overleftarrow{1}\,\overrightarrow{3}$.

Ushbu algoritm o'rin almashtirishlarni generatsiya qilishning eng samarali usullaridan biri sanaladi.

Ro'yxatdagi eng oxirgi o'rin almashtirish $n\ (n-1)\ ...\ 1$ bo'lishi tabiiyroq. Agar o'rin almashtirishlarni leksikografik tartibda amalga oshirilsa, bunday tartibga ega bo'lish mumkin. Agar raqamlarni xarflar sifatida qabul qilinsa, bunday tartib lug'atlarda keltiriladigan tartib bilan ustma-ust tushadi:

123 132 213 231 312 321.

Leksikografik tartibda berilgan $a_1\ a_2\ ...\ a_{n-1}$ dan keyingi o'rin almashtirishni qanday tashkil qilish mumkin? Agar $a_{n-1} < a_n$ bo'lsa, u holda oxirgi ikki element o'rinlari almashtiriladi (masalan, 123 dan keyin 132). Aks holda a_{n-2} elementga o'tiladi. Agar $a_{n-2} < a_{n-1}$ bo'lsa, oxirgi uchta element o'rinlari a_{n-2} ni minimal o'zgartirgan holda almashtiriladi, ya`ni bu o'ringa a_{n-1} va a_n elementlar orasidan a_{n-2} dan kattasi yoziladi, navbatdagi ikki o'ringa esa qolgan ikki element o'sish tartibida joylashtiriladi. Masalan, 132 dan keyin 213, 231 dan keyin eesa 312 yoziladi. Umumiy holda o'ngdan chapga qarab joriy o'rin almashti- rishdan $a_i < a_{i+1}$ shartni qanoatlantiruvchi (va demak, $a_{i+1} > ... > a_n$) qo'shni elementlar izlanadi. So'ngra, ro'yhat oxiridan a_i dan katta bo'lgan eng kichik element topiladi va uni i-chi o'ringa joylashtiriladi; i-1 dan n gacha bo'lgan pozitsiyalarga $a_i, a_{i+1}, ..., a_n$ ro'yxatning qolgan elementlari joylashtiriladi.

Qism to'plamlarni qurish. Berilgan $A = \{a_1, a_2, ..., a_n\}$ to'plamning barcha 2^n qism to`plamlarini topish masalasi qo'yilgan bo'lsin. Bu masalani o'lchamini birga pasaytirish metodi orqali hal qilish mumkin.

Barcha $A = \{a_1, a_2, ..., a_n\}$ to`plamni ikkita guruhga ajratish mumkin: a_n ni o'z ichiga olgan va olmagan qism to'plamlar. Birinchi guruhga $\{a_1, ..., a_{n-1}\}$ to'plamning barcha qism to'plamlari, ikkinchi guruh esa $\{a_1, ..., a_{n-1}\}$ to'plamning har bir qism to'plamiga a_n ni qo'shish orqali hosil qilinadi. Demak, biz $\{a_1, ..., a_{n-1}\}$ to'plamning barcha qism to'plamlari ro'yhatini hosil qilganimizdan so'ng, ro'yxatga uning a_n qo'shilgan barcha elementlarini yozib, $\{a_1, ..., a_n\}$ to'plamning barcha qism to'plamlariga ega bo'lish mumkin.

Bu algoritmni $\{a_1, a_2, a_3\}$ to'plamga nisbatan amalda qo'llash jarayoni 8.1-jadvalda keltirilgan.

8.1-jadval. Qism to'plamlarni generatsiya qilish

n	qism to'plamlar							
0	$\varnothing$							
1	$\varnothing$	$\{a_1\}$						
2	$\varnothing$	$\{a_1\}$	$\{a_2\}$	$\{a_1, a_2\}$				
3	$\varnothing$	$\{a_1\}$	$\{a_2\}$	$\{a_1, a_2\}$	$\{a_3\}$	$\{a_1, a_3\}$	$\{a_2, a_3\}$	$\{a_1, a_2, a_3\}$

$A = \{a_1, a_2, ..., a_n\}$ to'plamning barcha qism to'plamlarga uzunligi n bo'lgan $b_1, ..., b_n$ bitli satrlarni mos qo'yish qo'yilgan masalani hal qilishning eng qulay usullaridan biri hisoblanadi. Bunda agar $b_1 = 1$ bo'lsa a_i element qism to'plamga tegishli bo'ladi, aks holda – yo'q. Masalan, 000 bitli satr bo'sh to'plamga mos keladi.

$n = 3$ bo'lganda quyidagi ro'yhat hosil bo'ladi.

Bitli satr	000	001	010	011	100	101	110	111
qism to'plam	$\varnothing$	$\{a_1\}$	$\{a_2\}$	$\{a_1, a_2\}$	$\{a_3\}$	$\{a_1, a_3\}$	$\{a_2, a_3\}$	$\{a_1, a_2, a_3\}$

8.5. O'lchamni o'zgarmas ko'paytuvchi miqdorga pasaytirish. Bu usul eng samarali algoritm qurish usullaridan biri hisoblanadi. Uning g'oyasi berilgan masala o'lchamlarini o'zgarmas miqdorga (masalan, 2 marta) pasaytirib, yangi masala nusxalarini shakllantirish va ular orqali asosiy masalaning yechimini topishga asoslangan. Avvalgi boblarda ko'rilgan binar izlash yoki darajaga ko'tarish masalalari ana shunday masalalar toifasiga kiradi. Odatda bunday algoritmlar logarifmik xarakterga ega bo'ladi va yetarlicha katta tezlikka erisha oladi.

Qalbaki tanga haqidagi masala. Faraz qilaylik, n ta bir hil qiymatga va shaklga ega bo'lgan tangalar berilgan bo'lib, ularning biri qalbaki va boshqalaridan yengilligi bilan ajralib turadi. Ana shu tangani tarozi yordamida ajratib olishning samarali algoritmini ishlab chiqish talab qilinadi.

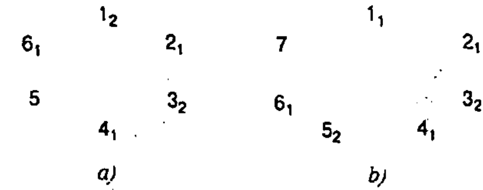
Masalani hal qilishning eng oson usuli – bu ularni ikki qismga ajratish va tarozi yordamida qalbaki tanga qaysi qismda yotganligini aniqlash, so'nga yengil qismdagi tangalarni yana ikki qismga bo'lish va bu jarayonni toki har bir qismda bittadan tanga qolguncha davom ettirishdan iborat bo'ladi. Ammo, bu usul algoritmlar ichida eng yaxshisi emas.

Masalaning yaxshi algoritmini qurish uchun tangalarni uchta qismga ajratish talab qilinadi. Tarozida tortilganda ulardan ikkitasining bir hil og'irlikka ega bo'lishi qalbaki tanganing uchinchi qismda yotganligini, aks holda tarozining yengil pallasiga qo'yilganligini anglatadi. Qalbaki tanga yotgan qismni yana uchga bo'linadi va jarayon qismlarning har birida bittadan tanga qolguncha davom ettiriladi. Ana shu tangalarni tortib, masala yechimini topish mumkin. Bu usulda masala o'lchami algoritmning har iteratsiyasida uch marta pasayishi ko'rinib turibdi. Demak, taroizda tortishlar soni $\log_3 n$ ga teng bo'ladi.

Doirada turgan odamlar masalasi. Faraz qilaylik, n ta odam doirada turgan bo'lsin. Ularning har biri o'z tartib nomeriga ega. Sanash 1 nomerli odamdan boshlanadi. Har bir o'tishda sanoq birdan boshlanadi va har ikkinchi odam doiradan chiqariladi. Sanash doirada faqat bitta odam qolguncha davom etadi. Oxirgi qolgan odamning nomerini topish talab qilinadi.

Agar doirada 6 ta odam turgan bo'lsa, birinchi o'tishda 2, 4, 6, ikkinchi o'tishda 3 va 1 nomerli odamlar doiradan chiqariladi va oxirgi qolgan odamning nomeri 5 ga teng. Agar doirada 7 ta odam turgan bo'lsa, birinchi o'tishda 2, 4, 6 va 1, ikkinchi o'tishda esa 5 va 3 nomerli odamlar doiradan chiqariladi va oxirgi qolgan odamning nomeri 7 bo'ladi (8.5-rasm).

Masalani n ning juft va toq holatlari uchun alohida tahlil qilishga to'g'ri keladi. Agar $n=2k$ bo'lsa, u xoldda birinchi o'tishdan keyin biz yana shu masalaga kelamiz, ammo uning o'lchami avvalgisiga qaraganda ikki marta kichik bo'lib, faqat nomerlari bilan farq qiladi holos. Osongina ko'rish mumkinki, birinchi o'tishda 3 nomerli odam ikkinchi o'tishda 2, 5 nomerli odam 3 nomerlarni oladi. Demak, odamning boshlang'ich holatdagi nomerinn topish uchun uning yangi nomerini 2 ga ko'paytirish va 1 ni ayirish lozim: $f(2k)=f(k)-1$. Bu munosabat doirada oxirgi qolgan odam uchun ham o'rinli bo'ladi.



8.5-rasm. Indekslar shu odamni nechanchi o'tishda doiradan chiqishini ko'rsatadi.

Endi n ning toq son bo'ladigan, ya`ni $n>1, n=2k+1$ holatini ko'raylik. Birinchi o'tishda juft nomerli odamlar doiradan chiqariladi. Agar bunga 1 nomeri odam ham qo'shils, u xolda boshlang'ich masalaning k o'lchamli nusxasi hosil bo'ladi. Bunda odamlarning yangi pozitsiyasiga ko'ra avvalgi nomerini topish uchun $f(2k+1)=2f(k)+1$ formuladan foydalanish mumkin.

Shunday qilib, masalani yechish uchun quyidagi rekkurent munosabat qurildi:

$$f(n)=\begin{cases}2f(k)-1, & agar\ \ n=2k\\ 2f(k)+1, & agar\ \ n=2k+1\ va\ n>1,\\ 1, & agar\ \ n=1\end{cases}$$

Masalan hal qilishning yana bir usul – bu *f(n)* ning dastlabki 15 ta qiymatini topish, mavjud qonuniyatlarni aniqlash va bu qonuniyatlarni matematik induktsiya yordamida isbotlashdan iborat. Boshqa bir ajoyib usul berilgan odamlar soni *n* ni ikkilik sanoq sistemasida ifodalash va hosil bo'lgan bitlar ketma ketligini tsiklik ravishda chapga bitta pozitsiyaga surishdan iborat. Masalan, $f(6)=110_2=101_2=5,\ \ f(7)=111_2=111_2=7$.

8.6. O'lchamni o'zgaruvchan miqdorga pasaytiradigan algoritmlar

Bunday algoritmlarning har bir qadamida masala o'lchami turli miqdorlarga pasayadi va oxirgi qadamda asosiy masala yechimi topiladi. Ikki natural son uchun eng katta umumiy bo'luvchini topishning Yevklid algoritmi ana shunday algoritmlardan hisoblanadi.

Medianani hisoblash va tanlash masalasi.

Tanlash masalasi berilgan *n* ta sonlar orasidan *k*-chi eng kichik elementiini topishdan iborat. Bunday son k-chi tartibli satistika deb ataladi. Tabiiyki, $k=1$ yoki $k=n$ bo'lganda berilgan elementlar ro'yxatini eng katta yoki eng kichik elementni topish uchun to'liq ko'rib chiqish mumkin. Ammo, bunday masalalar ichida $k=\lceil n/2\rceil$ bo'lib, ro'yhatning birinchi yarmidan katta, ikkinchi yarmidan kichik bo'lgan elementni topish masalasi diqqatga sazovor hisoblanadi. Mediana deb ataladigan bunday o'rta qiymat matematik statistikada muhim ahamiyat kasb etadi. Albatta, berilgan sonlar massivini tartiblagandan so'ng uning *k*-chi elementini olish mumkin. Bu holda algoritm samaradorligi $O(n \log n))$ ga teng bo'ladi.

Masalani hal qilishning eng samarali usullaridan biri tartiblanmagan massivdan uning yagona k-chi elementini ko'rsatishdan iborat. Tabiiyki, bu holda massivni tartiblash ortiqcha ish hisoblanadi. Buning uchun massiv ikkita qismga shunday ajratish talab qilinadiki, qandaydir tayanch p element ularning birinchisidagi elementlardan kichik, ikkinchisidagi elementlardan esa katta bo'lmaydi

$$\underbrace{a_{i_1} \dots a_{i_{s-1}}}_{\leqslant p}\; p\; \underbrace{a_{i_{s+1}} \dots a_{i_n}}_{\geqslant p}.$$

Bu usul quyidagicha amalga oshiriladi. Aytaylik, s - ajratish pozitsiyasi bo'lsin. Agar $s = k$ bo'lsa, u holda tayanch element p tanlash masalasining yechimi bo'ladi. Agar $s > k$ bo'lsa, element ro'yxatning chap qismida joylashadi. Agar $s < k$ bo'lsa, u holda ro'yxatning o'ng qismidan $(k-s)$ - chi eng kichik elementni topish kerak bo'ladi. Shunday qilib, agar algoritmning joriy qadamida masala yechimi hosil bo'lmasa, hech bo'lmaganda masalaning o'lchami pasaytirilgan nusxasi shakllanadi. Bu nusxani rekursiv hal qilish mumkin.

Aslida bu masalani rekursiyasiz ham hal qilish mumkin. Bunda to'g'ridan – to'g'ri $s = k$ shart o'rinli bo'lmaguncha tekshirish davom ettiriladi.

1-misol. Quyidagi sonlar medianasini toping: 4, 1, 10, 9, 7, 12, 8, 2, 15.

Bu holda $k = \lceil 9/2 \rceil = 5$ bo'ladi va massivning 5-chi eng kichik elementini topish talab qilinadi. Ro'yhat elementlari 1 dan 9 gacha nomerlangan. Huddi tez saralash algoritmidagi kabi tayanch element sifatida birinchi element olinadi va massivni qarama-qarshi yo'nalishda qarab chiqishdagi elementlarning o'rinlari almashtiriladi.

4 1 10 9 7 12 8 2 15

2 1 4 9 7 12 8 10 15

$s=3<k=5$ bo'lgani uchun, ishni ro'yxatning o'ng qismida davom ettiriladi.

9 7 12 8 10 15

8 7 9 12 10 15

$s=6>k=5$ bo'lgani uchun ro'yhatning chap qismi bilan ishlanadi.

8 7

7 8

Endi $s=k=5$ bo'lib qoldi va ishni tugatish mumkin. Biz topgan mediana 8 bo'lib, u 2, 1, 4 va 7 dan katta, 9, 12, 10 va 15 dan esa kichik.

Algoritm tez saralashga qaraganda o'rtacha samaraliroq hisoblanadi, chunki algoritmning har bir iteratsiyasida ro'yhatning bir qismi bilan ish olib boriladi. Agar ro'yhatni qismlarga ajratish massivning qolgan qismi o'rtasi uchun bajarilishini hisobga olsak, taqqoslashlar sonini sanash uchun quyidagi rekkurent munosabatdan foydalanish mumkin:

$$C(n)=C(n/2)+(n+1).$$

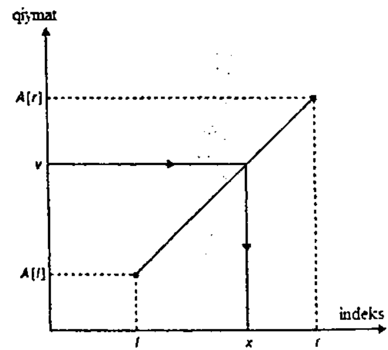
Ushbu masalada massiv o'lchami oldindan aytib bo'lmaydigan miqdorda (ayrim hollarda ikki marta, ayrim xollarda ko'proq) pasaymoqda. Tahlillarning ko'rsatishicha, bunday algoritmning o'rtacha samaradorligi o'lchamlarning xar gal ikki marta pasayishiga qaraganda kattaroq bo'lar ekan. Eng yomon holda algoritm samaradorligi $\Theta(n^2)$ gacha tushadi.

Interpolyatsion izlash. Tartiblangan massivdan kalit ma`lumotni izlash jarayonini interpolyatsion deb ataladigan usul bilan amalga oshirish mumkin. Izlashning bu usulida kalitning qiymati u bilan taqqoslanishi kerak bo'lgan element indeksini aniqlashda hisobga olinadi, ya`ni unga yaqinroq indeks tanlashga xarakat qilinadi.

$A[l]$ (chap chegara) va $A[r]$ (o'ng chegara) lar o'rtasidan izlash iteratsiyasini bajarishda algoritm massiv qiymatlari chiziqli ravishda ortib boradi degan nuqtai-nazarga asoslanadi. Shu farazga ko'ra, v kalit qiymat bilan taqqoslanadigan element indeksi $(l, A[l])$ va $(r, A[r])$ nuqtalar orqali o'tuvchi to'g'ri chiziqda yotuvchi x nuqtaning yahlitlangan koordinatasi tarzida aniqlanadi. Bu nuqtaning y koordinatasi v qiymatga teng bo'ladi.

$(l, A[l])$ va $(r, A[r])$ nuqtalar orqali o'tuvchi to'g'ri chiziqning standart tenglamasini yozgandan keyin, undagi y ni r bilan almashtirib, x ga nisbatan yechsak, quyidagi tenglamaga ega bo'lamiz:

$$x = l + \left\lfloor \frac{(v - A[l])(r - l)}{A[r] - A[l]} \right\rfloor.$$



8.6-rasm. Interpolyatsion izlashda indekslarni hisoblash.

Ushbu metod ostida yotgan g'oya juda ham sodda. $A[l]$ va $A[r]$ elementlarning o'sishi bizga ma`lum, ammo uning qanday o'sishini bilmaymiz. Aytaylik, bu o'sish chiziqli bo'lsin. Bu holda yuqoridagi formula bilan hisoblangan indeks qiymati v ga teng bo'lgan kalitli elementning kutilmasi bo'ladi.

$A[x]$ bilan v taqqoslanganidan keyin algoritm yoki o'z ishini tugatadi (agar ular teng bo'lsa) yoki xuddi shu usul bilan indekslari l dan to $x+1$ gacha yoki $x+1$ dan r gacha bo'lgan elementlar orasidan izlashni davom ettiradi. Shunday qilib, har bir iteratsiyasida masalaning o'lchami qandaydir miqdorga pasayadi, ammo bu miqdor oldindan ma`lum bo'lmaydi.

Algoritm samaradorligini tahlil qilinganda, interpolyatsion izlashda o'rtacha $\log_2 \log_2 n+1$ dan ozroq bazaviy taqqoslash amallari bajariladi. Bu funksiya shu qadar sekin o'sadiki, amaliy jihatdan uni konstantaga teng deb olish ham mumkin.

9-§. Dinamik programmalash

Dinamik programmalash mashhur amerikalik matematik Richard Bellman (Richard Bellman) tomonidan 1950 yillarda ishlab chiqilgan bo'lib, qarorlarni qabul qilishga qaratilgan ko'p bosqichli optimallashtirish masalalarini yechishning umumiy metodi sifatida qabul qilingan. Bu o'rinda "programmalash" so'zi "rejalashtirish" ma`nosida talqin qilinadi va kompyuter programmalariga hech qanday aloqasi yo'q. Mazkur metod amaliy matematikaning eng muhim vositalaridan biri bo'lib, kibernetiklar o'rtasida algoritmlar loyihalarini ishlab chiqishning faqat optimallashtirish masalalari bilan cheklanib qolmaydigan metodi sifatida qarala boshlandi. Biz ham ushbu bitiruv malakaviy ishida dinamik programmalash metodiga aynan shu nuqtai-nazardan qaraladi.

Dinamik programmalash bir-birini yopuvchi ost masalalardan tashkil topgan masalalarni yechish metodi hisoblanadi. Odatda bunday ost masalalar berilgan masalala yechimlarini ko'rinishi xuddi shunday, ammo kichikroq ko'lamdagi masalalarning yechimlari bilan bog'lovchi rekkurent munosabatlar natijasida yuzaga keladi. Bir-birini yopuvchi ost masalalarni takror va takror yechish o'rniga dinamik programmalash xar bir kichik masalani faqat yuir marta yechishni taklif etadi va bu jarayonda kichik masalalarning yechimlarini jadvalga yig'ib boradi. Ishning so'ngida bu jadvaldan boshlang'ich masala yechimlarini xosil qilinadi.

Ushbu metodning ishlash jarayonini Fibonachchi sonlari misolida ko'rib chiqamiz. Ma`lumki, Fibonachchi sonlari

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \dots$$

ketma-ketligilan iborat bo'lib, quyidagicha ko'rinishda rekkurent munosabatlar yordamida ifodalanishi mumkin:

$$F(n) = F(n-1) + F(n-2), \quad n \geq 2. \qquad (1)$$

Bu munosabatga boshlang'ich shart bo'lib quyidagilarni olish mumkin:

$$F(0) = 0, \quad F(1) = 1. \qquad (2)$$

Fibonachchi sonlarining n-chi xadini oddiy usul bilan hisoblaganda, *F(n)* funksiya qiymatini ko'p martalab hisoblashga va olingan qiymatlarni bir o'lchovli massivga yozib borishga to'g'ri keladi. Ko'rish mumkinki, *F(n)* ni hisoblash masalasi bir-birini yopuvchi va o'lchamlari boshlang'ich masaladan past bo'lgan *F(n-1)* xamda *F(n-2)* ost masalalardan iborat. Shunday qilib, bir o'lchovli massivni *F(n)* ning (1) formula bilan hisoblanadigan qiymatlari bilan to'ldiriladi. Ish esa (2) formuladan boshlanadi. Massivning oxirgi elementi *F(n)* ning qiymatini saqlashi tabiiy.

Shuni ta`kidlash joizki, ushbu masalani yechishda qo'shimcha massivdan foydalanmaslik ham mumkin. Buning uchun yangi hadning qiymatini hisoblashda undan avvalgi ikkita element qiymatlaridan foydalaniladi. Bunday vaziyatlarda dinamik programmalash usulini qo'llash xotirani tejash evaziga vaqtdan yutishga imkon beradi.

Dinamik programmalashga asoslangan algoritmlar uchun mazkur masalaning ost masalalar yechimlarini topish odatiy hisoblanadi. Bu metod variantlaridan yana biri keragi bo'lmagan ost masalalarni yechishdan qochishha yordam beradi. Umuman aytganda, dinamik programmalash algoritmining asosiy bosqichi masala yechimlarini o'lchamlari kichikroq bo'lgan ost masala yechimlari bilan bog'lovchi rekkurent munosabatlarni o'z ichiga oladi.

9.1. Binomial koeffitsientlarni hisoblash

Binomial koeffitsientlarni hisoblash - optimallashtirish masalasi bo'lmasada, dinamik programma- lashga yaqqol namuna bo'la oladigan standart masala hisoblanadi. Elementar kombinatorika kursidan ma`lumki, C_n^k ko'rinishida yozish qabul qilingan binomial koeffitsient deb n ta elementli to'plamdan k ta ($0 \le k \le n$) elementli kombinatsiyalar miqdoriga aytiladi. "Binomial koeffitsientlar" atamasi shu sonlarni binom formulasidagi ishtirokidan kelib chiqqan:

$$(a+b)^n = C_n^0 a^n + \cdots + C_n^k a^{n-k} b^k + \cdots + C_n^n b^n.$$

Binomial koeffitsientlar bir qator hususiyatlarga ega bo'lib, biz ulardan faqat ikkitasi ustida to'xtalib o'tamiz:

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k, \quad n > k > 0 \ bo'lganda, \qquad (3)$$

$$C_n^0 = C_n^n = 1. \qquad (4)$$

C_n^k ni hisoblash formulasini ifodalovchi (3) rekkurent munosabatning tabiati o'lchami kichikroq bo'lgan va bir-birini yopuvchi C_{n-1}^{k-1} xamda C_{n-1}^k larni hisoblashdan iborat masalalar vositasida C_n^k ni hisoblash masalasi bizni dinamik program- malash metodiga muro- jaat qilishga undaydi. Buning uchun binomial koeffitsientlar qiymatlarini $n+1$ ta satr va $k+1$ ta ustundan iborat jadvalga yoziladi (9.1-rasm).

	0	1	2	$\cdots$	$k-1$	k
0	1					
1	1	1				
2	1	2	1			
$\vdots$						
k	1					1
$\vdots$						
$n-1$	1				C_{n-1}^{k-1}	C_{n-1}^k
n	1					C_n^k

9.1-rasm. Binomial koeffisiyentlarni dinamik programmalsh yordamida hisoblash jadvali

C_n^k ni hisoblash uchun

1-rasmdagi jadvalni 0 – chi satrdan boshlab, n-chi satrgacha satrma-satr to'ldiriladi. Har bir i ($0 \le i \le n$) satr 1 dan boshlab chapdan o'ngga qarab to'ldiriladi, chunki $C_n^0 = 1$. Jadvalning bosh dagonalida 0 dan k-chi satrgacha 1 lar joylashadi, chunki $C_i^i = 1$ $(0 \le i \le k)$. Jadvalning qolgan elementlari (3) formula yordamida, avvalgi satr elementlarini qo'shish orqali hisoblanadi. Bu jadval Paskal jadvalini ifodalaydi xamda quyidagi psevdoalgoritm yordamida tavsiflanadi.

Algoritm *Binomial koeffitsientlar*
// *dinamik programmalash koeffitsientlarini hisoblash*
// **kiruvchi ma`lumotlar**: *Ikkita nomanfiy son* $0 \le k \le n$
// **chiquvchi ma`lumotlar:** *S(p, k) ning qiymati*
for i ← 0 **to** *n* **do**
for *j* ←*0* **to** *min(i, k)* **do**
if *j = 0 or j =i* **then** *C[t, j]←1*
else *C[t, j]←C[i-l, j-1] + C[t-1, j]*
return *C[n,k]*

Mazkur algoritmning vaqt bo'yicha samaradorligi qanday? Bu yerda bazaviy deb qo'shish amali olinadi. *C(n, k)* ni hisoblash uchun bajarish talab qilingan amallar sonini *A(n, k)* bilan belgilanadi. Ma'lumki, Har bir elementni (3) formula yordamida hisoblash uchun faqat bitta qo'shish amali bajariladi. Qolaversa, jadvalning dastlabki $k+1$ ta satri uchburchak, qolgan $n-k$ ta satrlar esa to'g'ri to'rtburchak tashkil qilgani uchun, *A(n, k)* ni hisoblash formulasini ikkita qismga ajratish mumkin:

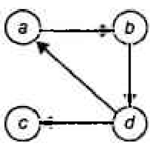
$$A(n,k) = \sum_{i=1}^{k}\sum_{j=1}^{i-1} 1 + \sum_{i=k+1}^{n}\sum_{j=1}^{k} 1 = \sum_{i=1}^{k}(i-1) + \sum_{i=k+1}^{n} k =$$
$$= \frac{(k-1)\,k}{2} + k\,(n-k) \in \Theta\,(nk)\,.$$

9.2. Vorshal va Floyd algoritmlari

Vorshal algoritmi. Eslatib o'tamizki, orientirlangan graflarda qo'shnilik matritsasi $A=\{a_{ij}\}$ bul matritsasidan iborat bo'lib, i-chi satr va j-chi ustun kesishmalaridagi yacheykalar faqat va faqat i-chi uchdan j-chi uchga yo'nalgan qirra mavjud bo'lganda 1, qolgan xollarda esa 0 qiymatga ega bo'ladi. Bunday matritsalardan tashqari, grafning ihtiyoriy uchlari o'rtasida mavjud bo'lishi mumkin bo'lgan yo'l haqidagi ma`lumotlarni saqlovchi matritsa ham diqqatga sazovor.

***Ta`rif-1. Tranzitiv tutashuv***ni uchlarining soni $N$ ta bo'lgan orientirlangan grafda $i$-chi satr va $j$-chi ustunlar kesishmasi agar $i$-chi uchdan $j$-chi uchga notrivial orientirlangan yo'l mavjud bo'lsa 1 ga (ya`ni, orientirlangan yo'l uzunligi musbat), aks xolda 0 ga teng bo'lgan $n\times n$ o'lchamli $T=\{a_{ij}\}$ bul matritsasi tarzida aniqlash mumkin.

9.2– rasmda orientirlangan grafga namuna xamda uning qo'shnilik matritsasi va tranzitiv tutashuvlari tasvirlangan.



a)

$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

b)

$$T = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

c)

9.2-rasm. *a*) Orientirlangan graf; *b*) qo'shnilik matritsasi; *c*) tranzitiv tutashuv.

Orientirlangan grafdagi tranzitiv tutashuvni eniga yoki ichkariga qarab izlash orqali qurish mumkin. i-chi uchdan boshlab izlash usullaridan birini qo'llash shu uchdan o'tish mumkin bo'lgan uchlar haqida ma`lumot olishga imkon beradi. Bundan tranzitiv tutashuv matritsasining i-chi satri bilan kesishadigan ustunlar 1 ga teng bo'lishi kelib chiqadi. Shunday qilib, tranzitiv tutashuv to'liq matritsasini grafning har bir uchini boshlang'ich sifatida qaragan xolda aylanib chiqib topish mumkin.

Bunday metod bitta orientirlangan grafning o'zini bir necha marta aylanib chiqsa-da, yanada samaraliroq algoritmning mavjudligiga umid tug'dirishi mumkin. Bunday metod mavjud va uni Vorshal algoritmi deb ataladi. Bu algoritm n-ta uchli orientirlangan grafning tranzitiv tuta- shuvini $n\times n$ o'lchovli bul matritsalari sifatida topishga imkon beradi:

$$R^{(0)}, ..., R^{(k-1)}, R^{(k)}, ..., R^{(n)} . \qquad (5)$$

Bu matritsalarning har biri grafdagi yo'llar yo'nalishi haqidagi ma`lumotlarni saqlaydi. Hususan, $R^{(k)}$, $(k = 0, 1, ..., n)$ matritsaning i-chi satri va j-ustunlari kesishmasidagi $r_{ij}^{(k)}$ element faqat va faqat i-chi uchdan j-uchga olib keluvchi musbat uzunlikdagi orientirlangan yo'ldagi barcha oraliq uchlarining nomerlari k dan katta bo'lmasa 1 ga teng bo'ladi. Shunday qilib, $R^{(0)}$ dagi yo'llar oraliq uchlarga ega bo'la olmaydi va demak, $R^{(0)}$ orientirlangan grafning qo'shnilik matritsasini ifodalaydi. $R^{(1)}$ matritsa oraliq uchlar sifatida 1-chi nomerli uch ishtirok etadigan yo'llar haqidagi ma`lumotlarni saqlaydi. Demak, aytish mumkinki, $R^{(0)}$ ga qaraganda ko'proq 1 larga ega bo'lishi lozim. Shunday qilib, umuman aytganda, (5) ketma-ketlikdagi har bir navbatdagi matritsa avvalgisiga qaraganda 1 ta ko'p oraliq uchga ega bo'ladi va demak, ko'proq sondagi birlarga ega bo'ladi (ammo, bunday bo'lishi shart emas). Ketma-ketlikning oxirgi $R^{(n)}$ matritsasi oraliq uchlar sifatida orientirlangan grafning barcha uchlarini olishi mumkin (n ta) va demak orientirlangan grafning tranzitiv tutashuvini ifodalaydi.

Algoritmning o'ziga xosligi shundaki, har bir $R^{(k)}$ matritsanirng barcha elementlarini (5) qatorda o'zidan oldin kelgan $R^{(k-1)}$ asosida hiisoblab topish mumkin. Aytaylik, i-chi satr va j – chi ustunlar kesishgan joydagi $r_{ij}^{(k)}$ yacheyka 1 ga teng bo'lsin. Bu shuni anglatadiki, i-chi v_i uchdan j-chi v_j uchga yo'l mavjud va bu yo'ldagi barcha oraliq uchlarning nomerlari k dan katta emas:

$v_i \to$ nomeri k dan katta bo'lmagan uchlar ro'yxati$\to v_j$. (6)

Bunday yo'lda ikki hil holat sodir bo'lishi mumkin. Birinchidan, oraliq uchlar ro'yhatida k-chi uch kirmagan. Bu xolda v_i dan v_j ga eltuvchi bu yo'l nomerlari k-1 dan katta bo'lmagan uchlardan iborat bo'ladi va demak, $r_{ij}^{(k-1)}$ element ham 1 ga teng bo'ladi. Ikkinchidan, (6) yo'l boshqa oraliq uchlar bilan birgalikda k-chi uch v_k ni ham o'z ichiga oladi. Umuman aytganda, v_k ro'yxatda faqat bir marta uchraydi xolos (agar bunday bo'lmasa, v_i dan v_j ga eltuvchi yo'lni qurish mumkin va unda v_k larning birinchi va oxirgi m arta uchrashlari o'rtasidagi barcha uchlar o'chirib tashlanadi). Bu fikrlarni e`tiborga olib, (6) ni quyidagi qayta yozib olish mumkin:

$v_i \to$ nomeri $\leq k$ -1 bo'lgan uchlar,

v_k, nomeri $\geq k$-1 bo'lgan uchlar $\to v_j$. (7)

Bu ifodaning birinchi qismi v_i dan v_k ga eltuvchi va barcha oraliq uchlarining nomerlari k-1 dan katta bo'lmagan (demak, $r_{ij}^{(k-1)} = 1$) yo'l mavjudligini anglatadi. Ikkinchi qismi esa v_k dan v_j ga eltuv - chi va barcha oraliq uchlarining nomerlari k-1 dan katta bo'lmagan (demak, $r_{kj}^{(k-1)} = 1$) yo'lning mavjudligini anglatadi.

Demak, agar $r_{ij}^{(k)} = 1$ bo'lsa, u xolda yoki $r_{ij}^{(k-1)} = 1$ yoki $r_{ik}^{(k-1)} = 1$ xamda $r_{kj}^{(k-1)} = 1$ bo`ladi. Osongina ko'rish mumkinki, bunga teskari tasdiq ham o'rinli. Shunday qilib, biz $R^{(k-1)}$ matritsadan $R^{(k)}$

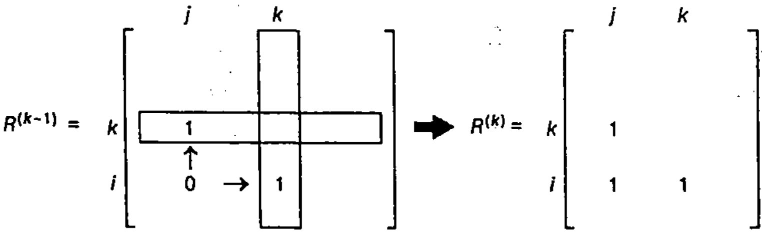
matritsa elementlarini hosil qilish uchun formulaga ega bo'ldik:

$$r_{ij}^{(k)} = r_{ij}^{(k-1)} \ or \ r_{ik}^{(k-1)} \ and \ r_{kj}^{(k-1)} . \qquad (8)$$

(7) formula Vorshal algoritmi asosini tashkil qiladi. Undan $R^{(k-1)}$ matritsadan $R^{(k)}$ matritsa elementlarini hosil qilish uchun quyidagi qoida (xattoki "qo'lda " hisoblashga ham imkon beradigan) kelib chiqadi:

agar $R^{(k-1)}$ da r_{ij} element 1 ga teng bo'lsa, u xolda $R^{(k)}$ da xam r_{ij} element 1 ga teng bo'ladi;

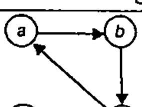
$R^{(k-1)}$ da r_{ij} element 0 ga teng bo'lsa, u xolda $R^{(k)}$ da r_{ij} element faqat va faqat $R^{(k-1)}$ da r_{ik} element ham, r_{kj} element ham 1 ga teng bo'lgandagina 1 ga teng bo'ladi (bu qoida 9.3-rasmda tasvirlangan).



9.3-rasm. Vorshal algoritmida nollarni almashtirish qoidasi

9.3-rasmda tasvirlangan orientirlangan graf uchun Vorshal algoritmini qo'llash 9.1-jadvalda keltirilgan.

Vorshal algoritmining psevdokodi quyidagicha ko'rinishga ega.

	Berilgan graf.

$R^{(0)}=\begin{array}{cc} & \begin{array}{cccc}a & b & c & d\end{array} \\ \begin{array}{c}a\\b\\c\\d\end{array} & \begin{bmatrix}0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 0 & 0\\1 & 0 & 1 & 0\end{bmatrix}\end{array}$	Bu yerda birlar oraliq uchlarsiz yo'llarning mavjudligini aks ettiradi ($R^{(0)}$- qo'shnilik matritsasi); to'g'ri to'rtburchakdagi ustun va satrlar $R^{(1)}$ ni hisoblashda foydalaniladi.
$R^{(1)}=\begin{array}{cc} & \begin{array}{cccc}a & b & c & d\end{array} \\ \begin{array}{c}a\\b\\c\\d\end{array} & \begin{bmatrix}0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 0 & 0\\1 & 1 & 1 & 0\end{bmatrix}\end{array}$	Birlar oraliq uchlarining nomerlari 1 dan katta bo'lmagan yo'llar, ya`ni, faqat *a* uchli (d dan b ga) yo'llar mavjudligini anglatadi; to'g'ri to'rtburchak bilan ajratilgan satr va ustunlar $R^{(2)}$ ni topishda foydalaniladi
$R^{(2)}=\begin{array}{cc} & \begin{array}{cccc}a & b & c & d\end{array} \\ \begin{array}{c}a\\b\\c\\d\end{array} & \begin{bmatrix}0 & 1 & 0 & 1\\0 & 0 & 0 & 1\\0 & 0 & 0 & 0\\1 & 1 & 1 & 1\end{bmatrix}\end{array}$	Birlar oraliq uchlarining nomerlari 2 dan katta bo'lmagan yo'llarning, ya`ni, a va b uchli (ikkita yangi yo'l) mavjudligini anglatadi; ajratilgan satr va ustunlar R(3) ni topish uchun foydalaniladi
$R^{(3)}=\begin{array}{cc} & \begin{array}{cccc}a & b & c & d\end{array} \\ \begin{array}{c}a\\b\\c\\d\end{array} & \begin{bmatrix}0 & 1 & 0 & 1\\0 & 0 & 0 & 1\\0 & 0 & 0 & 0\\1 & 1 & 1 & 1\end{bmatrix}\end{array}$	Birlar oraliq uchlarining nomerlari 3 dan katta bo`lmagan yo'llarning, ya`ni, *a, b* va *c* uchli (yangi yo'llar yo`q) mavjudligini anglatadi; ajratilgan satr va ustunlar $R^{(4)}$ ni topish uchun foydalaniladi
$R^{(4)}=\begin{array}{cc} & \begin{array}{cccc}a & b & c & d\end{array} \\ \begin{array}{c}a\\b\\c\\d\end{array} & \begin{bmatrix}1 & 1 & 1 & 1\\1 & 1 & 1 & 1\\0 & 0 & 0 & 0\\1 & 1 & 1 & 1\end{bmatrix}\end{array}$	Birlar oraliq uchlarining nomerlari 4 dan katta bo'lmagan yo`llarning, ya`ni, *a, b. c va d* uchli (beshta yangi yo'l) mavjudligini anglatadi.

9.1-jadval. Vorshal algoritmini orientirlangan grafga nisbatan qo'llash. Yangi birlar qora shrift bilan ajratib ko'rsatilgan.

Algoritm Vorshal $\{A\ [1..n, 1..n]\}$

// Tranzitiv tutashuvni hisoblash uchun qo'llanadi

// **kiruvchi ma`lumotlar:** *n*-uchli graf A qo'shnilik matritsasi

// **chiquvchi ma`lumotlar:** Grafning tranzitiv tutashuvi

$R^{(0)} \leftarrow A$

for $k \leftarrow 1$ to n do

for $i \leftarrow 1$ to n do

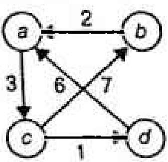
for $j \leftarrow 1$ to n do

$R^{(k)}[i, j] \leftarrow R^{(k-1)}[i, j]$ **or** $R^{(k-1)}[i, k]$ **and** $R^{(k-1)}[k, j]$

return $R^{(n)}$

Vorshal algoritmiga nisbatan quyidagilarni aytish mumkin. Birinchidan, u o'ta qisqa. Ikkinchidan, uning vaqt bo'yicha murakkabligi $O(n^3)$ ga teng. Vorshal algoritmi ichki tsiklini o'zgartirib, tezlashtirish mumkin. Algoritmni tezlashtirishning yana bir usuli matritsa satrlarini bitli satrlar sifatida qarash xamda ***or*** bitli amalidan foydalanishni nazarda tutadi. Vanihoyat ko'rish mumkinki, Vorshal algoritmi asosida yotadigan g'oyani orientirlangan graflardagi eng qisqa yo'llarni topish uchun ham qo'llash mumkin.

Floyd algoritmi (uchlar juftligi o'rtasidagi eng qisqa yo'lni topish uchun). Barcha uchlar juftligi o'rtasidagi eng qisqa yo'lni topish masalasi berilgan graf uchun (orientirlangan graf bo'lishi shart emas) har bir uchdan boshqa barcha uchlargacha bo'lgan eng qisqa masofani topishdan iborat. Eng qisqa yo'l uzunliklarini yozish uchun o'lchamlari $n \times n$ bo'lgan D matritsadan foydalanish qulay. Bu matritsani ***masofalar matritsasi*** deb ataymiz: bu matritsaning i-chi satri va j-chi ustunlari kesishmasida joylashgan d_{ij} element i-chi uchdan j-chi uchgacha bo'lgan eng qisqa yo'l uzunligini ko'rsatadi. $(1 \leq i, j \leq p)$. Bunday matritsaga namuna 9.5- rasmda tasvirlangan.



a)

$$W = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

b)

$$D = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

c)

9.5- rasm. *a*) orientirlangan graf; *b*)-uning og'irlik matritsasi; *c*) – masofalar matritsasi.

Masofalar matritsasini Floyd algoritmi yordamida qurish mumkin. Bu algoritmni orientirlangan va orientirlanmagan graflarga nisbatan qo'llash mumkin. Bunda asosiy e`tibor grafda manfiy uzunlikka ega bo'lgan tsikllarning mavjud bo'lmasligiga qaratiladi.

Floyd algoritmi n-uchli grafning masofalar matritsasini quyidagi ketma-ketlikni qurish yordamida hisoblaydi:

$$D^{(0)}, ..., D^{(k-1)}, D^{(k)}, ..., D^{(n)} . \qquad (9)$$

Bu matritsalarning har biri ma`lum bir cheklovlarga ega bo'lgan eng qisqa yo'l uzunliklarini saqlaydi. Hususan, $D^{(k)}$ $(k = 0, 1, ..., n)$ matritsaning i-chi satri va j-chi ustunlari kesishmasida joylashgan $d_{ij}^{(k)}$ element oraliq uchlarining nomerlari k dan katta bo'lmagan i-chi uchdan j-chi uchgacha bo'lgan yo'llar orasida eng qisqa yo'l uzunligiga teng bo'ladi. Hususan, ketma-ketlik oraliq uchlari mavjud bo'lmagan $D^{(0)}$ matritsadan boshlanadi (ya`ni, $D^{(0)}$ matritsa shunchaki og'irliklar matritsasidan iborat bo'ladi). ketma-ketlikning oxirgi $D^{(n)}$ matritsasi oraliq uchlari grafning n-uchlaridan ihtiyoriylaridan iborat bo'lgan eng qisqa yo'l uzunliklariga teng bo'lishi mumkin. Va demak, $D^{(n)}$ matritsa izlangan graf masofalar matritsasidan iborat bo'ladi.

Vorshal algoritmidagi kabi Floyd algoritmida ham har bir $D^{(k)}$ matritsa elementlarini (9) ketma-ketlikda undan oldin turadigan $D^{(k-1)}$ matrtsa elementlaridan foydalanib topish mumkin. Aytaylik, $D^{(k)}$ matritsaning i-chi satr va j-ustunlari kesishmasida turgan element $d_{ij}^{(k)}$ bo'lsin. Bu shuni anglatadiki, $d_{ij}^{(k)}$ element i-chi uchdan j-uchgacha bo'lgan eng qisqa yo`l uzunligiga teng bo`ladi va bu yo`ldagi barcha oraliq uchlarning nomerlari $k$ dan katta bo`lmaydi:

$$v_i \to \text{nomeri } k \text{ dan katta bo`lmagan uchlar ro'yhati} \to v_j . \qquad (10)$$

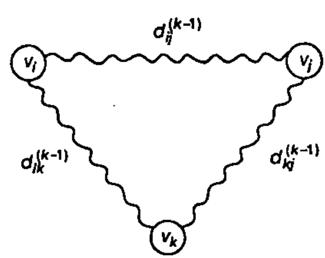
Barcha bunday yo'llarni kesishmaydigan ikkita qism to'plamga ajratish mumkin: oraliq sifatida k-chi uch v_k ishtirok etmaydigan yo'llar; v_k oraliq sifatida ishtirok etadigan yo'llar. Birinchi qism to'plam o'z ichiga nomer k-1 dan katta bo'lmagan oraliq uchlarni olgani uchun, bu uchlar o'rtasidagi eng qisqa yo'l uzunligi $d_{ij}^{(k-1)}$ ga teng.

Ikkinchi qism to'plamdagi eng qisqa yo'l uzunligi nimaga teng? Agar graf manfiy uzunlikdagi tsikllarga ega bo'lmasa, u xolda ikkinchi qism to'plamdagi yo'llarni v_k uchlar faqat bir martadan kiradigan yo'llarnri qarash bilan cheklanish mumkin. Barcha bunday yo'llar quyidagicha ko'rinishga ega bo'ladi:

$$v_i \to \text{uchlarning nomeri} \leq k\text{-}1,$$
$$v_k, \text{ uchlarining nomeri} \leq k\text{-}1 \to v_j. \qquad (11)$$

Boshqacha aytganda, bu yo'llarning har biri v_i dan v_k gacha bo'lgan va oraliq uch nomerlari k-1 dan katta bo'lmagan yo'llardan iborat bo'ladi. Shuningdek, v_k dan v_j gacha bo'lgan yo'llarning ham oralqi uchlarining nomerlari k-1 dan katta bo'lmaydi. Bu vaziyatni sxemalar orqali 9.6-rasmdagi kabi tasvirlash mumkin.

v_i dan v_k gacha bo'lgan masofa nomerlari k-1 dan katta bo'lmagan oraliq uchlardan foydalan maydigan yo'llar orasida eng qisqa yo'l uzunligi $d_{ik}^{(k-1)}$, v_k dan v_j gacha bo'lgan eng qisqa yo'l uzunligi $d_{kj}^{(k-1)}$ bo'lgani uchun v_i dan v_j gacha bo'lgan eng qisqa masofa $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$ ga teng. Xar ikki qism to'plamlardagi eng qisqa yo'l



9.6-rasm. Floyd algoritmi asosidagi g'oya.

uzunliklarini e`tiborga olib, quyidagi munosabatni yozish mumkin:

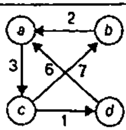
$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} \qquad k \geq 1,\ d_{ij}^{(0)} = \omega_{ij} \ \ lar\ uchun\ . \tag{12}$$

Ya`ni, joriy $D^{(k-1)}$ masofalar matritsasining i-chi satr va j-chi ustunlar kesishmasida joylashgan element i-chi satr va k-chi ustun xamda k-chi satr va j-chi ustun kesishmasidagi element yig'indisi bilan faqat va faqat bu yig'indi joriy qiymatdan kichik bo'lgandagina almashtiriladi.

Floyd algoritmini 9.5-rasmdagi grafga nisbatan qo'llash 9.7-rasmda tasvirlangan.

Floyd algoritmining psevdokodi quyidagicha ko'rinishga ega. Unda uidagi faktdan foydalaniladi: (9) ketma-ketlikdagi har bir navbatdagi matritsa o'zidan avvalgi matritsa ustiga yozilishi mumkin.

```
Floyd algoritmi (W[1..n, 1..n])
// Floyd algoritmi grafning barcha uchlari juftliklari o`rtasidagi
// eng qisqa yo'l uzunligini topish uchun qo`llanadi
// Boshlang'ich ma`lumotlar: Grafning W og'irliklar matritsasi
// Chiquvchi ma`lumotlar: eng qisqa yo'l uzunligi
D←W  // Agar W ni qayta yozish talab qilinmasa
for k←1 to n do
        for i ←1 to n do
    for j←1 to n do
D[i, j] ← min {D[i, j], D[i, k]+D[k, j]}
     return D
```

	Qaralayotgan graf.

$D^{(0)}=\begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$	Oraliq uchlarsiz eng qisqa yo'l uzunligi oddiy og'irlik matritsasidan iborat bo'ladi. Bunda D(0) shunchaki og'irlik matritsasidan iborat bo'ladi.
$D^{(1)}=\begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \end{array}$	Oraliq uchlarining nomerlari 1 dan katta bo'lmagan yo'llar ichida eng qisqa yo'l uzunligi, ya`ni faqat a, (ikkita yangi eng qisqa yo'l yo'llar: b dan c gacha xamda d dan c gacha).
$D^{(2)}=\begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ 9 & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \end{array}$	Oraliq uchlarining nomerlari 2 dan katta bo'lmagan, ya`ni a va b bo'lgan eng qisqa yo'llarning uzunligi (c dan a ga yangi yo'l) .
$D^{(3)}=\begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 9 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$	Oraliq uchlarining nomerlari 3 dan katta bo'lmagan, ya`ni *a*, *b* va *c* bo'lgan eng qisqa yo'llarning uzunligi (*a* dan *b* ga, *a* dan *d* ga, *b* dan *d* ga, *d* dan *b* ga to'rtta yangi yo'l).
$D^{(4)}=\begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$	Oraliq uchlarining nomerlari 4 dan katta bo'lmagan, ya`ni *a*, *b*, *c* va *d* bo'lgan eng qisqa yo'llarning uzunligi (*c* dan *k* ga yangi eng qisqa yo'l).

7-rasm. Floyd algoritmini orientirlangan grafga nisbatan qo'llash. Unda yangilangan elementlar qoraytirilgan shrift bilan ajratib ko'rsatilgan.

Floyd algoritmning vaqt bo'yicha murakkabligi xuddi Vorshal algoritmi kabi kubik darajaga ega.

10-§. Ochko'z algoritmlar

Ochko'z algoritmlar g'oyasi ostida har bir iteratsiyada masalaning lokal optimal yechimini topish yotadi. Ammo, bu lokal yechimlar umumiy masalaning optimal yechimi bo'lmasligi mumkin.

Masalan, bajarilgan ish uchun mijoz tomonidan aytaylik, 128,7 ming so'mni naqd to'lash talab qilingan bo'lsin. Mijoz o'z oldiga bu to'lovni eng kam kupyuralar yordamida amalga oshirishni hohlaydi. Bugungi kunda muomalada yurgan 50000, 10000, 5000, 1000, 500, 200, 100, 50 va 25 so'mlik kupyuralardan foydalangan holda bu to'lov summasini qanday amalga oshirish mumkin?

Masalaning javobi juda ham sodda: to'lov 2 ta 50000, 2 ta 10000, 1 ta 5000, 3 ta 1000, 1 ta 500 va 1 ta 200 so'mlik kupyuralardan iborat bo'ladi. Bu yechimni optimal deb hisoblash mumkin.

Odatda ochko'z algoritmlar optimallashtirish masalalariga nisbatan qo'llaniladi. Bunday yondoshuvda masala yechimini qurish bosqichlar ketma-ketligi tarzida amalga oshiriladi va har bir bosqichda masala yoki uning joriy nusxasi uchun hususiy optimal yechim topiladi. Bu jarayon toki masalaning to'liq yechimi hosil bo'lmaguncha davom etadi.

Har bir bosqich uchun topiladigan hususiy yechimlar quyidagi shartlarni qanoatlantirishi lozim:

mumkin bo'lgan yechim, ya'ni masala cheklovlariga mos kelishi;

lokal optimal yechim, ya'ni har bir bosqich uchun mumkin bo'lgan variantlardan eng yaxshisi bo'lishi;

qat'iy, ya'ni yechim qabul qilinganidan keyin, algoritmning navbatdagi qadamlarida o'zgarmaydigan bo'lishi.

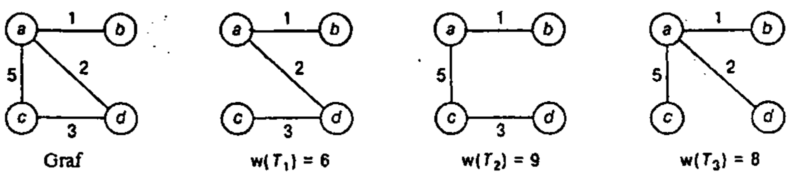
Aynan shu talablar algoritm nomini asoslab beradi. Oxir-oqibat masalaning kutilgan yechimiga olib boradi degan niyatda algoritmning har bir qadamida mumkin bo'lgan yechimlar orasidan eng yaxshisi "ochko'zlarcha" tanlanadi. Bunday yonoshuv bir qator masalalar uchun ochko'z algoritm yaxshi natija bersada, boshqa masalalar uchun kutilgan natijani ta'minlay olmaydi.

Prim algoritmi. N ta aholi yashaydigan nuqtalar berilgan bo'lib, ularning har bir jufti o'rtasidagi masofa ma'lum bo'lsin. Bu nuqtalarni shunday birlashtirish talab qilinadiki, har bir nuqtadan boshqasiga yo'l mavjud bo'lsin hamda umumiy masofa eng kichik bo'lsin.

Agar nuqtalarni grafning uchlari, ular o'rtasidagi masofalarni qirra uzunliklari deb qaralsa, qo'yilgan masala minimal sinchli daraxt masalasiga aylanadi.

Ta`rif-1. Bog'langan grafni ***sinchli daraxt*** deb ataladi, agar u grafning barcha uchlarini o'z ichiga oluvchi tsiklik bo'lmagan ost grafdan iborat bo'lsa.

Grafning vazni deganda uning barcha qirra uzunliklarining yig'indisi tushuniladi. Demak, minimal sinchli daraxt eng kichik vaznga ega bo'ladi. 10.1-rasmda keltirilgan mulohazalarni izoxlash uchun namunalar keltirilgan.



10.1-rasm. Graf va uning sinchli daraxtlari

Prim algoritmi ushbu masalani samarali hal qilish usullaridan biri hisoblanadi. Unga ko'ra minimal sinchli daraxt qadamba-qadam kengayib boruvchi daraxt shaklida quriladi. Algoritmning har bir qadami ochko'zlik printsipi ostida bajariladi va daraxtga hozircha unga kirmagan eng yaqin uch qo'shiladi. Agar eng yaqin uchlar bir nechta bo'lsa, tanlov ihtiyoriy tarzda amalga oshiriladi.

Algoritmning har bir iteratsiyasida daraxtga bitta uch qo'shilgani uchun, iteratsiyalarning umumiy soni n-1 ga teng bo'ladi.

Quyida mazkur algoritm uchun qurilgan psevdokod keltirilmoqda.

ALGORITM *Prim (G)*

// **kiruvchi ma`lumotlar**: Bog'langan $G = (V, E)$ graf

// **Chiquvchi ma`lumotlar:** minimal sinchli daraxtni tashkil
// qiluvchi qirralarning E_T to'plami
$V_T \leftarrow \{v_0\}$
// daraxt uchlari to'plami ihtiyoriy uch bilan tashkil qilindi
$E_T \leftarrow \varnothing$
for $i \leftarrow 1$ **to** $|V|$ - 1 **do**
$v \in V_T$ va $u \in V-V_T$ bo'lgan barcha (v, u) qirralar orasidan
vazni eng kichik bo'lgan E*=(v^*, u^*) qirra izlanmoqda
$V_T \leftarrow V_T \cup \{u^*\}$
$E_T \leftarrow E_T \cup \{e^*\}$
return E_T

Prim algoritmiga ko'ra darazxtga kirmagan uchlar uchun ikkita ma`lumotni nazarda tutadi: eng yaqin uchning nomeri va mos qirraning uzunligi. Daraxtning uchlari bilan qo`shni bo`lmagan uchlar $\infty$ belgisi bilan belgilab qo'yiladi. Bunday uchlarning vaznini 0 ga teng deb qabul qilinadi. daraxtga qo'shilishi lozim bo'lgan $u^*$ uch topilganidan keyin, quyidagi ikki amalni bajarish kerak bo`ladi:

u^* uchni $V-V_T$ to'plamdan o'chirib, daraxtning V_T to'plamiga qo'shish;

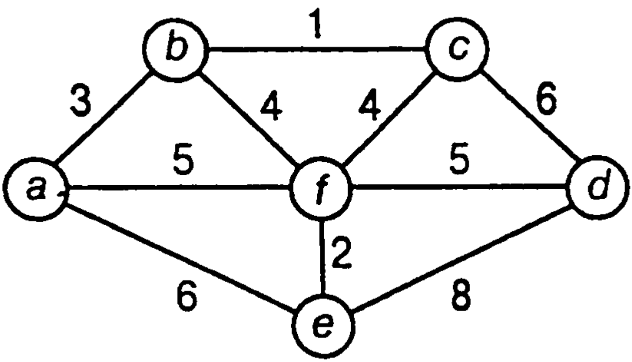
$V-V_T$ to'plamda qolgan va u^* bilan eng kichik qirra orqali bog'langan har bir uch uchun uninh nomini u^* bilan, uzunligini esa u^* gacha bo'lgan masofa bilan almashtirish.

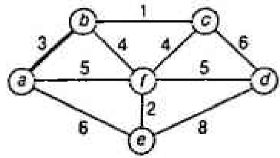
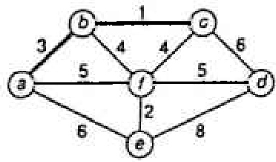
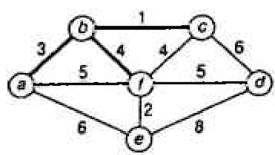
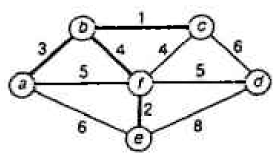
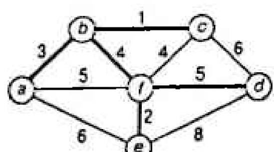
10.2-rasmda Prim algoritmini ko'rsatilgan grafga tatbiq etish jarayoni tasvirlangan.

Prim algoritmining to'g'ri ishlashini tekshiramiz. Induksiyaga ko'ra Prim algoritmi asosida tashkil qilingan har bir T_i, $i = 0, ..., n-1$ qism daraxt izlanayotgan minimal sinchli daraxtning bir qismi bo`lishini qo'rsatamiz.

Induksiyaning T_0 bazisi o'rinli, ya`ni u minimal sinchli daraxtga kiradi. Aytaylik, T_{i-1} minimal sinchli daraxtga kirsin. Ana shu daraxt yordamida qurilgan T^* daraxtning mirnimal sinchli daraxt bo'lishini ko'rsatamiz. Teskarisidan faraz qilamiz: T_i ni o'z ichiga olgan daraxt mavjud bo'lmasin. $e_i = (v, u)$ - qirra T_{i-1} daraxt uchidan T_{i-1} daraxtga kirmagan uchlar o'rtasidagi minimal masofa va T_{i-1} ni T_i gacha kengaytirish uchun Prim algoritmidan foydalanilgan bo'lsin. Farazga ko'ra e_i birorta ham minimal sinchli daraxtga kira olmaydi. Shunday qilib, agar e_i ni T daraxtga qo'shadigan bo'lsak, tsikl hosil bo'lishi kerak (10.3-rasm).

Tsikl $e_i = (v, u)$ o'z ichiga $v' \in T_{i-1}$ uchni $u' \notin T_{i-1}$ bilan birlashtiruvchi boshqa (v', u') uchni o'z ichiga olishi lozim. Bunda v' va v, u' *va* v lar ustma-ust tushishi mumkin, ammo bir vaqtda emas. Agar tsikldan (v', u') ni olib tashlasak, og'irligi T daraxt vaznidan katta bo'lgan boshqa minimal sinchli daraxtga ega bo'lamiz, chunki e_i ning vazni (v', u') dan katta bo'lmaydi.

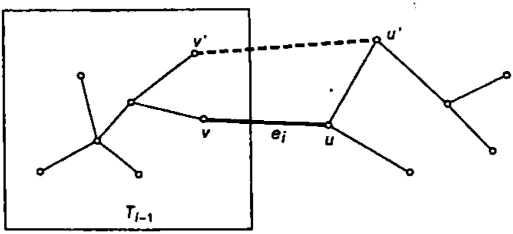


daraxtning uchlari	qolgan uchlar	tasviri
a(-,-)	**b(a,3)** c(-,∞) d(-,∞) e(a,6) f(a,5)	
b(a,3)	**c(b,1)** d(-,∞) e(a,6) f(b,4)	
c(b,1)	d(c,6) e(a,6) **f(b,4)**	
f(b,4)	d(f,5) **e(f,2)**	
e(f,2)	**d(f,5)**	
d(f,5)		

qavslarda daraxtning eng yaqin uchi va qirrasi vazni berilgan; daraxtni kengaytirish uchun tanlanadigan uchlar qoraytirilgan shriftda berilgan.

10.2-rasm. Prim algoritmni amalda qo'llash

Demak, bu daraxt minimal sinchli daraxt bo'ladi va T_i ni o'z ichga oluvchi minimal sinchli daraxt mavjud emas degan farazimizga zid keladi. Bu esa Prim algoritmning to'g'ri ekanligini tasdiqlaydi.



10.3-rasm. Prim algoritmi to'g'riligining isboti.

Algoritm samaradorligi daraxtni ifodalash uchun foydalanilgan ma`lumotlarning qanday tuzilmalari tanlanganligi hamda V-V_T to'plamga kirgan uchlar ustuvorligi (uchlarning ustuvorligi deganda ulardan joriy daraxt uchlarigacha bo'lgan eng yaqin masofa nazarda tutiladi) ga bog'liq.

Xaffman daraxtlari. Biror n ta belgidan iborat bo'lgan alfavit yordamida yozilgan matnni shifrlash talab qilingan bo'lsin. Bunda har bir belgiga kod deb ataluvchi qandaydir bitlar ketma-ketligi tayinlangan bo'lsin.

Shifrlashda har bir belgiga bir hil uzunlikdagi bitli satrlarni mos qo'yib, fiksirlangan uzunlikdagi kodlash usulidan foydalanish mumkin. Shifrlangan eng qisqa bitlar ketma-ketligini hosil qilish uchun eng ko'p uchraydigan belgilarga qisqaroq, kam uchraydiganlariga esa uzunroq bitlarni mos qo'yish (masalan, *e* (·), *a* (·-), *q* (--·-), *z* (--··)) g'oyasi XIX asrda Samuel Morze tomonidan taklif etilgan va amaliyotda juda ham keng qo'llanilgan.

O'zgaruvchan uzunlikda kodlashdan foydalanishda uchraydigan muammoni (k-chi belgi necha bitdan iborat ekanligini aniqlash) hal qilish uchun prefiksli kodlardan foydalanish mumkin. Bu usulda birorta ham kod boshqa kodning prefiksi bo'la olmaydi. Demak, bitli satr berilgan bo'lsa, undan biror belgining kodi bilan ustma-ust tushadigan bitlar uchramaguncha tekshiriladi va bu bitlarni mos belgi bilan almashtiriladi. Bu jarayon bitli satr tugamaguncha davom etadi.

Biror alfavit uchun binar prefiksli kod ishlab chiqish jarayonida uning belgilarini binar daraxt yaproqlari tarzida ifodalash maqsadga muvofiq hisoblanadi. Daraxtning chap qirralarini 0, o'ng qirralarini esa 1 bilan begilab qo'yish mumkin. Belgi kodini daraxt ildizidan yaproqqacha bo'lgan yo'l orqali hosil qilinadi. Bu usulda ikkita yaproq uchun bitta yo'l mavjud bo'lmaydi. Barcha bunday daraxtlar prefiksli kodni ifodalaydi.

Amalda qo'llanish chastotalari ma'lum bo'lgan belgilar uchun qurish mumkin bo'lgan daraxtlar orasida ko'p uchraydigan belgilarga qisqa, kam uchraydiganlariga uzunroq bitlar ketma-ketligini mos qo'yish usuli Devid Xaffman tomonidan ochko'z algoritm yordamida ishlab chiqilgan. U quyidagi qadamlardan iborat:

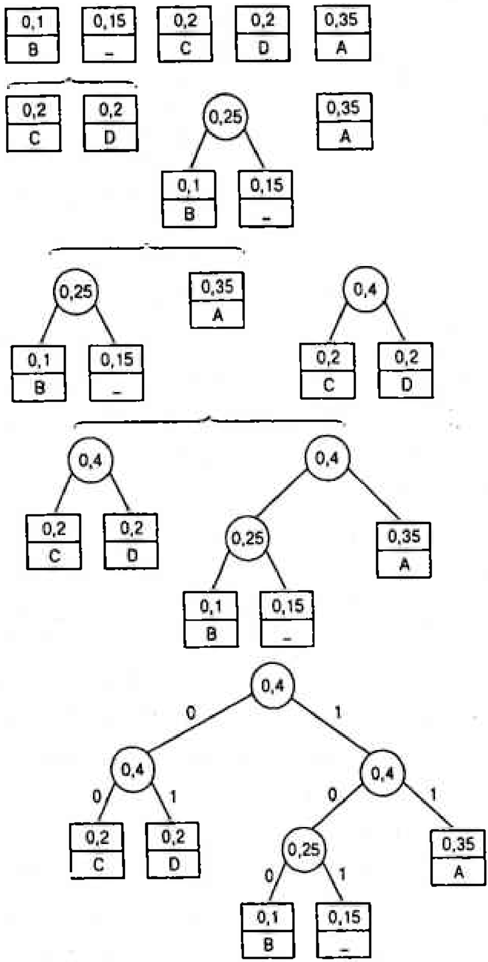
1-qadam. n – ta birtugunli daraxtlarni tashkil qilinadi va ularni alfavit belgilar orqali nomlanadi. Har bir belgi chastotasini uning daraxti tagiga vazn sifatida yozib qo'yiladi. Umumiy holda daraxtning vazni uning yaproqlarida ko'rsatilgan vaznlar yig'indisiga teng bo'ladi.

2-qadam. Quyidagi amal yagona daraxt hosil bo'lmaguncha davom ettiriladi. Vaznlari eng kichik bo'lgan ikkita daraxt topiladi (agar bunday daraxtlar ko'p bo'lsa, ihtiyoriy ikkitasi olinadi) va ularni yangi daraxtning chap va o'ng qism daraxtlari tarzida joylashtiriladi, daraxt ostiga ularning vaznlari yig'indisi yozib qo'yiladi. Bunday algoritm ostida qurilgan daraxtlarni Haffman daraxtlari, daraxtlar aniqlaydigan kodlar esa Haffman kodlari deb ataladi.

1-misol. Beshta belgidan iborat alfavit {A, B. C. D. ...} va ularning chastotasi quyidagicha bo'lsin:

Simvol	A	B	C	D	...
Ehtimolligi	0.35	0.1	0.2	0.2	0.15

Bu ma`lumotlar asosida Haffman daraxtini qurish jarayoni 10.4-rasmda tasvirlangan.



10.4-rasm. Xaffman usulida kodlashga namuna.

Natijada belgilarning quyidagi kodlariga ega bo'lamiz:

simvol	*A*	*B*	*C*	*D*	_
ehtimolligi	0.35	0.1	0.2	0.2	0.15
Kodi	11	100	00	01	101

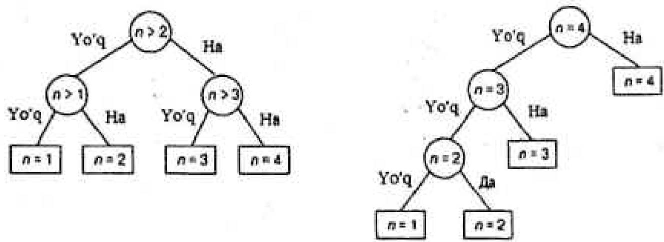
Demak, *DAD* – 011101 tarzida kodlanadi, 1001101101101 esa *BAD_AD* kabi qayta kodlanadi. Belgilarning berilgan ehtimolliklari hamda olingan kodlarga ko'ra bitta belgini kodlash uchun bitlarning matematik kutilmasi $2 \cdot 0.35 + 3 \cdot 0.1 + 2 \cdot 0.2 + 2 \cdot 0.2 + 3 \cdot 0.15 = 2.25$ ga teng bo'ladi.

Agar shu alfavit uchun fiksirlangan uzunlikdagi kodlar qo'llanga- nida, har bir belgi uchun kamida uchta bitlardan foydanishga to'g'ri kelgan bo'lar edi. Demak, ko'rish mumkinki, Xaffman kodlari bitlar ketma-ketligini ma`lum bir miqdorda qisishga yordam beradi. Bu miqdor $(3-2{,}25)/3 \cdot 100\% = 25\%$ ga teng. Boshqacha aytganda, ma`lumotlar Xaffman usulida kodlansa, fiksirlangan uzunlikka qaraganda 25% kam xotira talab qilinadi. Tahlillarning ko'rsatishicha, Xaffman usulida kodlashda ma`lumotlar o'z xarakteriga ko'ra 20% dan 80% gacha miqdorda qisilar ekan va berilgan matndan navbatdagi belgi (masalan, 1010) o'qilganidan so'ng kodlash daraxti har gal qayta quriladi. Shuni ta`kidlash joizki, Haffman algoritmi ma`lumotlarni qisish bilan chegaralanmaydi.

Faraz qilaylik, bizga *n* ta musbat son berilgan bo'lsin: $\omega_1, \omega_2, ..., \omega_n$. Bu sonlar binar daraxtning har bir tuguniga bittadan, *n* ta yaprog`iga mos qo'yilgan bo'lsin. Agar yo'l uzunligi $\sum_{i=1}^{n} l_i\omega_i$, (bu yerda $l$ –ildizdan $i$-chi yaproqqacha bo'lgan masofa) tarzida aniqlangan bo'lsa, uzunligi minimal bo`lgan daraxtni qanday qurish mumkin?

Bu Xaffman algoritm yordamida yechish mumkin bo'lgan umumiy masala hisoblanadi.

Bunday masalalar qaror qabul qilish bilan bog'liq bir qator masalalarda yuzaga kelishi mumkin. Masalan, n ta buyumlardan (masalan, 1 dan n gacha bo'lgan natural sonlar) biri tanlangan bo'lsa, uni javobi "ha" yoki "yo'q" qabilidagi savollar yordamida topish o'yinini olish mumkin. O'yin vaqtida qo'llash mumkin bo'lgan strategiyalardan birini qaror qabul qilish daraxti yordamida tanlash mumkin. $n = 4$ uchun bunday daraxt uchun namunalar 10.5-rasmda keltirilgan.



10.5-rasm. 1 dan 4 gacha bo'lgan sonlarni topish uchun ikkita daraxt.

Bunday daraxtdagi yo'l uzunligi o'ylangan sonni topish uchun kerak bo'ladigan savollar miqdoriga teng bo'ladi. Agar i sonini p_i ehtimollik bilan tanlansa, ildizdan i-chi yaproqqacha bo'lgan masofani l_i desak, savollarning o'rtacha miqdori $\sum_{i=1}^{n} l_i p_i$ ga teng. Agar berilgan sonlarning har birini bir hil ehtimollik (1G'n) bilan tanlansa, eng yaxshi strategiya huddi binar izlash algoritmidagi kabi ketma-ketlikning yarmini chiqarib tashlashdan iborat bo'ladi. Ammo, p_i ihtiyoriy (masalan, $n = 4,\ p_1 = 0{,}1,\ p_2 = 0{,}2,\ p_3 = 0{,}3,\ p_4 = 0{,}4$) bo'lganda bunday strategiyadan foydalanish yaramaydi. Bu holda minimal uzunlikdagi yo'l 10.4-rasmning o'ng tomonida tasvirlangan.

11-§. HAMMA IMKONIYATLARNI KO'RIB CHIQISH

P, NP va NP- to'liqligidagi masalalar. Dasturlash uchun masalalarni shartli ravishda uchta sinfga ajraish qabul qilingan: *P*, *NP* va *NP*- to'liqligidagi masalalar.

P-klass masalalarini polinomial vaqt mobaynida yechish mumkin bo'ladi. Boshqacha aytganda bu masalalarni $O(n^k)$ vaqt ichida hal qilish mumkin, bu yerda n – kiruvchi ma'lumotlarning o'lchami, k-qandaydir konstanta. Biz hozirgacha ko'rib chiqqan masalalarning ko'pchiligi ana shunday masalalardan hisoblanadi.

NP-klassidagi masalalarni polinomial vaqt davomida tekshirish mumkin bo'ladi. Bu yerda shuni ta'kidlash joizki, agar qandaydir usul bilan bu klass masalalari yechilgan bo'lsa, olingan yechimlar korrektligini (to'g'riligini) kiruvchi ma'lumotlar o'lchamiga polinomial bog'liq bo'lgan vaqt mobaynida tekshirish mumkin bo'ladi. *P*-klassdagi ihtiyoriy masala *NP*-klassiga ham taalluqli bo'ladi.

NP-klassga tegishli bo'lgan va yetarlicha katta murakkablikka ega bo'lgan masalalarni ***NP-to'liqligidagi*** masalalar deb ataladi. Bu masalalar *NP*-klassi masalalaridan polinomial vaqt ichida hal qilish algoritmlarining mavjud emasligi bilan farqlanadi.

NP-to'liqligidagi masalalarni hal qilishda kiruvchi ma'lumotlar o'lchamining kichik miqdorda o'zgarishi bir necha marta ko'p vaqt talab qiladi. Masalan, n-ta elementning o'rin almashtirishi bilan bog'liq masalalarda quyidagi miqdordagi variantlarni ko'rib chiqish talab qilinadi:

$$n=4 \quad uchun \quad 4!=24,$$
$$n=5 \quad uchun \quad 5!=120$$
$$n=6 \quad uchun \quad 6!=720.$$

NP-to'liqligidagi masalalar yechish jarayonining o'ta murakkab ekanligi bilan boshqa turdagi masalalardan farq qiladi. Shu sababli, bunday masalalarni yechishning tez ishlovchi algoritmlarini izlash o'rniga, ularning taqribiy bo'lsada yechishga yoki mahsus hollarini qarab chiqishga uringan ma`qul. Bir qator masalalar ham mavjudki, ular bir qaraganda soddaga o'xshaydi, ammo *NP*-to'liqlikka ega bo'ladi. Masalan, tarmoqdagi oqimni aniqlash, grafdan izlash masalalari ana shunday *NP*-to'liqligida deb hisoblanadi.

Masalalarning *NP*-to'liqlikka ega ekanligini isbotlashda quyidagi g'oyadan foydalaniladi: polinomial vaqt mobaynida yechish talab qilingan va qarorlar qabul qilishga bag'ishlangan A masala berilgan bo'lsin. Alohida olingan kiruvchi ma`lumotlar uchun olingan bunday masalalarni shu masalaning nusxasi deb ataladi. Aytaylik, yechimi qaror qabul qilish bilan bog'liq boshqa B masala mavjud va uni oldindan polinomial vaqtda yechish usuli ma`lum bo'lsin. Faraz qilaylik, A masalaning ihtiyoriy α nushasini B masalaning β nushasiga keltira oladigan protsedura mavjud va u quyidagi xarakteristikalarga ega bo'lsin:

Bunday almashtirish polinomial vaqt talab qilsin;

Javoblar bir hil, ya`ni α nusha yechimlari va β nusha yechimlari bir hil bo'lsin.

Bunday protseduralarni polinomial vaqtli keltirish algoritmlari deb ataladi va polinomial vaqt mobaynida A masalani yechish usulini taqdim etadi.

A masalaning α nushasi keltirish algoritmi yordamida B masalaning β nushasi bilan almashtiriladi.

B masalaning β nushasini yechish algoritmi ishga tushiriladi.

B masalaning β nushasi yechimidan A masala α nushasining yechimi o'rnida foydalaniladi.

Sanab o'tilgan bu bosqichlarning har birini polinomial vaqt davomida bajarish mumkin bo'lgani uchun, B masalaga keltirish yo'li bilan A masalaning "sodda"ligi isbotlanmoqda.

Ammo, yuqorida ta`kidlanganidek, *NP*-to'liqligidagi masalalar uchun ularning "sodda"ligini emas, balki juda qatta qiyinchilik bilan yechish mumkinligini isbotlanadi.

Hamma imkoniyatlarni ko'rib chiqish algoritmlari orqali hal qilinadigan masalalar *NP*-to'liqligidagi masalalar toifasiga kiradi va ularni polinomial vaqt davomida yechish uchun umumiy algoritmlar mavjud emas. Boshqacha aytganda, bu masalalarni yechish uchun eksponentsial vaqt talab qilinadi. Bu holat kiruvchi ma`lumot o'lchamlarining kichik miqdorda ortishiga algoritmda bajarish talab qilingan amallar sonining katta miqdorda ortishi sabab bo'ladi.

Agar kiruvchi ma`lumotlarning o'lchami katta miqdorda ortadigan bo'lsa, u holda bunday masalalarni xamma imkoniyatlarni qarab chiqish algoritmi bilan to`g'ridan – to'g'ri yechishning iloji bo'lmaydi. Hususiy hollarda bunday masalalarni qarab chiqilishi kerak bo'lgan variantlarni qisqartirishga erishgan holda yechishga xarakat qilish mumkin.

NP-to'liqligidagi masalalarga namuna qilib shahmat taxtasida figuralarni (masalan, farzinni) joylashtirish, r yukzak, kommivoyajer, labirint va boshqa bir qator masalalarni ko'rsatish mumkin.

Qaytarib izlash usuli. Ko`pincha, *NP*-to`liqligidagi masalalarni xamma imkoniyatlarni qarab chiqish usuli bilan hal qilishga to'g'ri keladigan hollarda qaytarib izlash deb ataladigan usuldan foydalanish mumkin bo'ladi.

Agar qaralayotgan masalaning yechimlari bir nechta bo`lsa, xamma imkoniyatlarni qarab chiqish orqali uning mumkin bo'lgan barcha yechim variantlarini hosil qilinadi. So`ngra bu variantlarning har biri uchun masala shartini tekshirish talab qilinadi. Tabiiyki, *NP*-to'liqligidagi masalalarni kiruvchi ma`lumotlar o'lchami katta bo`lganda yoki uzoq vaqt talab qiladi yoki umuman yechib bo'lmaydi.

Ayrim hollarda, qaytarib izlash usulini qo'llagan holda komponentalar orasidan yechimlarni qurish va bu yechimlar uchun masala shartini tekshirish mumkin. Agar qurilgan yechim masala shartini qanoatlantirmasa, u holda dastlabki bunday fiksirlangan komponentalar uchun mumkin bo'lgan boshqa yechimlarni ko'rib chiqishning qizig'i yo'q va komponentalarning navbatdagi variantiga o'tish mumkin bo'ladi. Bunday hollarda algoritm oxirgi qurilgan yechimga qaytadi va mumkin bo'lgan boshqa variant bilan almashtiradi.

Quyida qaytarib izlash usuli algoritmning umumlashtirilgan sxemasi keltirilmoqda.

Algoritm *qaytarib_izlash (X[1..i])*

// **kiruvchi ma`lumotlar:** dastlabki *i* ta yechimlarni qurish uchun

// *X[1..i]* massiv

// **Chiquvchi ma`lumotlar:** masalaning yechimi bo'lgan barcha kortejlar

if *X[1..i]* masalaning yechimi bo'lsa

then write *X[1..i]*

else

begin

for *X[1..i]* ga mos keladigan va masala shartidagi cheklovlarini qanoatlantiradigan har bir mumkin bo'lgan x_i uchun

begin

X[i+1]← x

Backtrack(X[1.. i + 1])

end

end

Shaxmat taxtasida farzinlar joylashtirish haqidagi masalasini yuqorida keltirilgan algoritm yordamida hal qilish mumkin.

Namuna tariqasida Eynshteyn taklif etgan quyidagi boshqotirma- ni keltirish mumkin. Bu misol qaytarib izlash izlash algoritmiga yaqqol misol bo'lib hizmat qiladi.

Masala: Har hil rangdagi 5 ta uy bo'lib, ularning har birida bittadan millat vakili yashaydi. Odamlarning har biri boshqalarga o'xshamagan ichimlik ichadi va har xil sigaret chekadi xamda uyida har hil hayvonlarni saqlaydi. Bu odamlar haqida quyidagi ma`lumotlar ma`lum:

1. Ingliz qizil uyda yashaydi.
2. Shved it boqadi.
3. Daniyalik choy ichadi.
4. Yashil uy oq uyning chap tomonida joylashgan.
5. Yashil uyda yashovchi odam kofe ichadi.
6. PallMall chekadigan odam qush boqadi.
7. O'rtadagi uyda yashovchi odam sut ichadi.
8. Norvegiyalik birinchi uyda yashaydi.
9. Sariq uyda yashovchi odam Dunhill chekadi.
10. Marlboro chekadigan odam mushuk boquvchi odamning yonida yashaydi.
11. Ot boqadigan odam Dunhill chekuvchining yonida yashaydi.
12. Winfield sigaretasini chekuvchi pivo ichadi.
13. Norvegiyalik ko'k uy yonida yashaydi.
14. Nemis Rothmans sigaretasini chekadi.
15. Marlboro chekuvchi odam suv ichuvchi odamning yonida yashaydi.

Savol: Kim baliq boqadi?

Bu masala yechimini mantiqiy fikrlash orqali 10-15 minut vaqt mobaynida topish mumkin. ammo. xamma imkoniyatlarni qarab chiqish usuli bilan bu masalani hal qilish uchun vaqt yetmaydi, chunki bu holatda hammasi bo'lib $(5!)^5 = 24883200000$ ta variantlarni qarab chiqishga to'g'ri keladi.

Masalani qaytarib izlash orqali hal qilishga urinib ko'ramiz. yechim uyning rangi, unda yashovchi odamning millati, ularning ichimliklari, sigaret nomlari va boqadigan hayvonlarni o'z ichiga oluvchi 5-ta komponentadan iborat bo'ladi.

Har bir bosqich uchun tanlov amalga oshirilganidan so'ng, bu tanlovlar uchun yechimni hosil qilish mumkinligini tekshirish lozim bo'ladi. Buning uchun, dastlab fiksirlab qo'yilgan shartlardan foydalaniladi. Masalan, 8-chi shartning o'zi qaraladigan ortiqcha variantlarni kesib tashlab, qarab chiqish lozim bo'lgan variantlar sonini $(5!)^4 = 207360000$ taga keltiradi.

Tabiiyki, dastlabki bosqichlarda qancha ko'p tekshirishlar bajarilsa, shuncha ko'p nokorrekt variantlarni kesib tashlash imkoniyati tug'iladi va bu holat masalaning yechish vaqtini sezilarli darajada qisqartirishga imkon beradi.

Mantiqiy tahlillar orqali masalaning mumkin bo'lgan yechimlaridan birini quyidagicha qurish mumkin[5].

Norvegiya	Daniya	Ingliz	Nemis	Shved
Sariq	Ko'k	Qizil	Yashil	Oq
Suv	Choy	Sut	Kofe	Pivo
Dinhill	Marlboro	Pallmall	Rotmans	Winfield
Mushuk	Ot	Qush	Baliq	It

Tahlillarning ko'rsatishicha[6], mazkur masala uchun ishlab chiqlgan dastur qarab chiqishladigan variantlarning umumiy soni 254 milliarddan ziyod bo'lgani holda, kesib tashlangandan keyin qoladigan

[5] Yechim muallaifga tegishli.
[6] И. В. Красиков, И. Е. Красикова. Алгоритмы. Просто как дважды два. 231-стр.

5474 ta variantlardan bor-yo'g'i 3965 variantni tekshirib, to'g'ri yechimga kelgan.

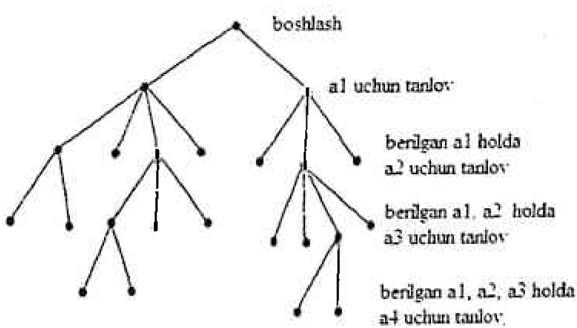
Shuni ta`kidlash joizki, barcha imkoniyatlarni qarab chiqish yordamida yechiladigan hamma masalalarni ham kesib tashlash usuli bilan hal qilish mumkin emas. Bu strategiyaning muvaffaqiyati juda ham keng diapazonda bo'lib, bir hil masalalar uchun yaxshi natija bersa, boshqa masalalar uchun xamma imkoniyatlarni qarab chiqishni taqozo etishi mumkin.

Odatda, qandaydir maqsad funksiyani optimallashtirish (minimallashtirish yoki maksimallashtirish) masalasini oldindan bir qator shart yoki cheklanishlar berilgan holdagina hal qilinishi mumkin.

Qaytish orqali xamma imkoniyatlarni qarab chiqish (umumiy sxema). N ta tartiblangan U_1, U_2, ..., U_N (N — oldindan noma`lum) to'plam berilgan bo'lsin. Ma`lum bir cheklov va shartlarni qanoatlantiruvchi

$A=(a_1, a_2,..., a_N)$, $a_1 \in U_1, a_2 \in U_2$, ..., $a_N \in U_N$ vektorni qurish talab qilinadi.

Xamma imkoniyatlarni qarab chiqish algoritmida A vektorni komponentalar bo'yicha chapdan o'ngga qarab quriladi. Faraz qilaylik, dastlabki k-1 ta komponentalarning qiymatlari topilgan bo'lsin:



$A=(a_1, ..., a_{k-1}, ?, ..., ?)$. U holda berilgan shartlar to'plami navbatdagi a_n komponentani tanlash imkoniyatini $S_k \subset U_k$ shart yordamida cheklab qo'yadi. Agar $S_k <> []$ (bo'sh emas) bo'lsa, a_k sifatida S_k ning eng kichik qiymatini olish mumkin. Shundan so'ng navbatdagi $k+1$ - chi komponentaga o'tish mumkin va h.k. Ammo, agar S_k bo'sh bo'lsa, u holda k-1 chi komponentani tanlashga o'tib, a_{k-1} komponenta tashlab yuboriladi va a_{k-1} ning yangi qiymati sifatida hozirgina tashlab yuborilgan elementdan keyin joylashgan S_{k-1} element olinadi. Bunda shunday yuo'lishi ham mumkinki, a_{k-1} ning yangi qiymati uchun masala sharti bo'sh bo'lmagan S_k ga ham ruxsat berishi mumkin va bu holda yana a_k ni tanlashga urinib ko'rish mumkin. Agar a_{k-1} ni tanlashning imkoni bo'lmasa, u holda yana bir qadam orqaga qaytiladi va yangi a_{k-2} ni tanlashga o'tiladi va h.k.

Mazkur jarayonni izlash daraxti tarzida ifodalash mumkin. Uning ildizi (0 –chi bosqich) bo'sh vektordan iborat. Uning tarmoqlari a_1 uchun nomzodlardan iborat. Umumiy holda f_k –chi bosqich tugunlar $a_1, a_2, ..., a_{k-1}$ tanlovlar amalga oshirilganidan keyin a_k ni tanlash uchun nomzodlarni ifodalaydi. Masala yechimining mavjud bo'lish yoki bo'lmasligi daraxtning qaysidir tarmoqlari masalaning yechimi bo'la olish yoki olmasligi bilan teng kuchli. Barcha yechimlarni izlar ekanmiz, biz hamma ana shunday tugunlarni aniqlashga xarakat qilamiz.

Taklif etilgan jarayonning rekursiv algoritmi uchun psevdokod quyidagicha yoziladi.

Procedure Backtrack (<vektor, i>) ;

begin

if <vektor yechim bo'lsa >

```
then <uni qayd etish >
else begin
<Si ni hisoblash>;
for <a∈Si> do Backtrack (<vektor|| a>, i+1);
{*// - vektorga komponentalarni qo'shib qo'yish *}
end;
end;
```

Tavsiflangan algoritmning vaqt bo'yicha murakkabligini baholaymiz. Xamma imkoniyatlarni qarab chiqishni bu usulda tashkil qilish vaqt bo'yicha eksponentsial algoritmlarga olib keladi. Haqiqatdan ham, aytaylik, barcha yechimlar N uzunlikka ega bo'lsin. U holda daraxtning $|U_1|*|U_2|*...*|U_N|$ tartibli tarmoqlarni ko'rib chiqishga to'g'ri keladi. Agar U_i ning qiymatlari qandaydir C konstanta bilan chegaradangan bo'lsa, u holda C^N miqdordagi tarmoqlarga ega bo'lish mumkin.

Farzinlarni joylashtirish haqidagi masala. $N*N$ o'lchovli shaxmat taxtasida N ta farzinni shunday joylashtirish talab qilinadiki, ularning har biri boshqalariga havf solmasin.

Farzinlarni shaxmat taxtasida joylashtirishning mumkin bo'lgan barcha variantlari - $C_{N^2}^{N}$ ($N=8$ bo'lgan hol uchun $4{,}4*10^9$) ga yaqin. Har bir ustunda ko'pi bilan bitta farzinni joylashtirish mumkin va bu holda variantlar soni N^N ($N=8$ uchun $1{,}7*10^7$) ga teng bo'ladi. Bitta satrga ikkita farzinni qo'yish mumkin emas, shuning uchun (1, 2, ..., N) sonlarning o'rin almashtirishlaridan iborat bo'lgan $(a_1, a_2, ..., a_N)$ vektor masalaning yechimi bo'lishi uchun qaprab chiqiladigan variantlar soni $N!$ ($N=8$ uchun $4{,}0*10^4$) ga teng. Har bir diagonalda ham ikkita farzinni joylashtirish mumkin emasligini e'tiborga olinsa qarash lozim bo'lan variantlar 2056 ta qoladi).

Shunday qilib, $N*N$ o'lchovli shaxmat taxtasida N ta farzinni joylashtirishning mumkin bo'lgan katta sondagi variantlarini qisqartirishga erishdik. Xamma imkoniyatlarni qarab chiqishda masalalarni bunday usul bilan tahlil qilish cheklovlar asosida izlash yoki daraxtdan uning qism daraxtlarini kesish usuli deb ataladi. Yana bir takomillashgan usullardan biri - bu shoxlarni birlashtirish yoki yopishtirish usulidir. Bu usulning g'oyasi bir marta bajarish mumkin bo'lgan amallarni takroran bajarish oldini olishdan iborat: agar daraxtning ikki yoki undan ortiqroq shoxlari o'xshash bo'lsa, faqat ulardan bittasini tahlil qilinadi xolos. Farzin haqidagi masalada birlashtirish amalidan foydalanish mumkin. Bunda agar $a_1 > \lceil N/2 \rceil$ bo'lsa, topilgan yechimni aks ettirish orqali $a_1 \leq \lceil N/2 \rceil$ bo'lgan hol uchun ham yechimni hosil qilish mumkin. Demak, $a_1 = 2$ va $a_1 = N-1$ bo'lgan hol uchun qurilgan daraxtlar o'xshash.

Quyidagi ma`lumotlar yuqorimdagi mulohazalarni tasdiqlaydi va kiritiladigan ma`lumotlar strukturasini izoxlab beradi.

Ma`lumotlar strukturasi:

Up: Array [2. .16] of boolean ;

// birinchi tipdagi diagonallarning bandlik alomati//

Down:Array[-7. 7] of boolean;

// ikkinchi tipdagi diagonallarning bandlik alomati//

Vr: Array [1..8]of boolean;

// vertikallarning bandlik alomati//

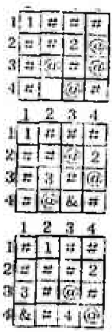
X: Array [1. .8] of integer;

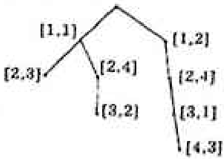
// har bir gorizontalda joylashgan farzinning vertikal nomeri//

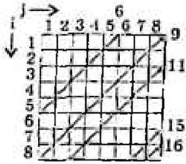
Quyida yechimlarni pastdan yuqoriga texnologisi asosida tashkil qiluvchi "g'ishtchalar"ni izoxi keltirilmoqda.

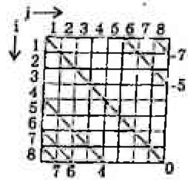
ALGORITM Yurish (i, j)

// **kiruvchi ma`lumotlar:** *i, j : integer*









// **chiquvchi ma`lumotlar :** navbatdagi yurish koordinatalari

X[d] ←j; Vr[j] ← False;

Up [i+j]←False;

Down[i-j]←False;

End;

Algoritm bekor (i , j :Integer) ; //Oxirgi yurishni bekor qilish

// **Kiruvchi ma`lumotlar:** oxirgi yurish koordinatalari

// **Chiquvchi ma`lumotlar:** yurishni bekor qilish

Vr [j]←True;Up[i+j]←True;

Down[i-j]←True;

End;

Algoritm tekshirish (i, j : Integer)

// (i, j) pozitsiyaga yurish mumkinligini tekshirish

// **kiruvchi ma`lumotlar:** *(i, j) pozitsiya*

// **Chiquvchi ma`lumotlar:** mantiqiy qiymat *(false* yoki *true)*

tekshirish←Vr[j] And Up[i+j] And Down [i-j] ;

end;

Farzinlarni joylashtirishning bitta variantini izlashning asosiy psevdokodi quyidagicha yoziladi:

Algoritm Solve (i : Integer ;Varq: Boolean) ;

// **kiruvchi ma`lumotlar:** yangi pozitsiya nomerlari

// **Chiquvchi ma`lumotlar:** yangi pozitsiya nomerlari

Oraliq kattalik: *Vcir j: Integer;*

J←0;

repeat

Inc (j) ;

q←False; // vertikal bo'yicha tsikl

if *tekshirish (i, j) Then*

Begin Yurish(i, j) ;

if *i<8* **then begin** *Solve (i+1 , q) ;*

if *Not q* **then** *bekor(i, j);* **end**

else *q←True; // yechim topildi*

end;

until *q or 0=8 ;*

end;

Masalaning qo'yilishini o'zgartiramiz. Aytaylik, *N*N* o'lchamli shaxmat taxtasida farzinlarning mumkin bo'lgan barcha joylashtirishlarini topish talab qilingan bo'lsin. Oldindan aytish mumkinki, 8*8 o'lchamli shaxmat taxtasi uchun joylashtirishlar soni 92 ga teng. Bu muammoni quyidagi algoritm yordamida hal qilish mumkin:

Algoritm *Solve(i: integer)*;

// **kiruvchi ma`lumotlar:** yangi pozitsiya nomerlari

// **Chiquvchi ma`lumotlar:** yechimlar soni

Oraliq kattalik: *j: Integer;*

if *i<=N* **then begin**

for *j←1* **to** *N* **do**

if *tekshirish (i, j)* **then begin** *Yurish(i, j) ; Solve (i+1) ;*
bekor (i, j) ;

end; end else

```
begin inc (S);// yechimlar soni, global o'zgaruvchi
print; // yechimlarni chop qilish
end;
end;
```

Masalani o'rganishni davom ettiramiz. Endi faqat nosimmetrik yechimlarni izlaymiz. 8*8 o'lchamdagi taxta uchun javob 12 ga teng. Farzin haqidagi masalaning hamma yechimlarini dasnlabki topilgan yechimlarni shaxmat taxtasini 90°, 180° va 270° hamda taxtani teng ikkiga bo'luvchi chiziqlarga nisbatan ko'zguviy burishlar (bunda koordinatalar sistemasi qo'zg'almas deb qaraladi) orqali hosil qilish mumkin.

Tahlillar shuni ko'rsatdiki, shaxmat taxtasida *N* ta farzinlarni joylashtirish jarayonida quyidagi vaziyatlar yuzaga kelishi mumkin:

- taxtani bir marta ko'zguviy aks ettirishda farzinlarning yangi joylashuvi yuzaga keladi, burishlarda esa yangi yechimlar hosil bo'lmaydi;
- 90° ga burish va uni aks ettirish farzinlarning yana ikkita yangi joylashuvini ta`minlaydi;
- uchta burish va to`rtta aks ettirish yangi joylashuvlarni beradi.

Mumkin bo'lgan yechimlardan simmetrik bo`lganlarini kesib tashlash uchun tartibning ayrim munosabatlarini belgilab olish talab qilinadi. yechimlarni koordinatalari 1 dan *N* gacha bo`lgan sonlardan iborat *N* o`lchovli vektor sifatida ifodalaymiz; *i*-chi satrda turgan farzinning ustuni koordinatasi vektorning *i*-chi elementiga teng bo`ladi. Simmetrik yechimlarni hisobga olmaslik uchun dan uchun simmetriya orqali hosil qilish mumkin bo`lgan barcha vektorlar orasidan minimal vektorni topamiz.

Quyida keltirilayotgan *Sim1, Sim2, Sim3* algoritmlar yechimlar vektorini gorizontal, vertikal va bitta diagonal o`qqa nisbatan ko`zguviy burishlarni amalga oshiradi. Geometriya kursidan ma`lumki, bu simmetriyalar kompozitsiyasi yuqorida keltirilgan barcha shaxmat taxtasi simmetriyalarini kafolatlaydi. yechim vektorining minimallikka tekshirish *Str* funksiyasi tomonidan bajaroiladi. Mumkin bo`lgan variantlardan bittasi quyida keltirilmoqda.

```
type TArrayqarray[1. .N] of integer;
procedure Sim1 (Var X: TArray) ;
var i:integer;
begin
for i:=1 to N do X[i]:=N-X[i] +1;
end;
Procedure Sim2 (var X: TArray);
var i , r: integer;
begin
for i:=1 to N div 2 do
begin r:=X[i]; X[i] :=X[N-i+1]; X[N-i+1]:=r; end;
end;
procedure Sim3 (var X: TArray);
var Y:TArray;
i: integer;
begin
for i:=1 to N do Y[X[i]] :=i; X:=Y;
end;
function Cmp (X, Y: TArray):boolean;
var i: integer;
begin
i : q1 ;
```

```
while (i<=N) and (Y[i]=X[i]) do
Inc(i); if  i>N then Cmp:=false
                    else if Y[i]<X[i] then Cmp:=true
                                  else Cmp:=false;
end;
Procedure Solve(i:Integer);
var j: integer; f: boolean;
Y: TArray;
begin
if i<=N then begin for j :=1 to N do
if D_hod(i, j) then begin yurish(i, j) ; Solve (i+1) ; bekor (i, j);
end;
         end else begin
f:=true;
for j : =0 To 7 do begin Y:=X;
if j and 1 =0 then Sim1 (Y) ;
if j and 2 =0 then Sim2 (Y) ;
if j and 4 =0 then Sim3 (Y) ;
if Cmp(Y, X) then f:=false;
              end;
if f then begin Inc(S);{*Yechimlar soni, global o'zgaruvchi*}
Print; {*Yechimlarni chop qilish *}
end;
            end;
end;
```

Eslatma: Farzin haqidagi masalani hal qilish dasturini turli usullar bilan yozish mumkin. Quyida mumkin bo'lgan variantlardan biri keltirilmoqda. Dastur juda ham ihcham hamda o'ziga hos. Uni tahlil qilishga urinib ko'ring.

```
program Ferz;
uses crt;
const N=8;
var V:Array[1..N] of integer;
 procedure Rf(i : integer)
   var j, k, p, t: integer;
begin
    for j:=l To N do begin
        B[i]:=j; k:=l; p:=0;
        while (k<l) and (p=0) do begin
          if (B[k]=B[i]) or (abs (k-i)=abs(B[k]-B[i] )) then p:=1;
Inc(k);
        end;
 if p=0 then if  i<N then Rf(i+1)
 else begin for t:=l To N do write (B [t] : 3) ; writeLn; end; end;
end;
 begin
 clrscr;
 Rf(1);
 end.
```

Otning yurishi haqidagi masala. Har bir katakka faqat bir marta yurgan shaxmat taxtasi ot bilan to'la aylanib chiqish mumkin. Ana shunday aylanib chiqishlar soni sanab chiqing.

8*8 o'lchamli shaxmat taxtasida yurishlar soni 64! ga teng. Har bir aylanib chiqishni ot bilan aylanib chiqishmi yoki yo'qmi deb baholash lozim. So'ngra xamma imkoniyatlarni qarab chiqishlar sonini qisqartirishga (otning navbatdagi yurishini baholashga) xarakat qilamiz. Aytaylik, navbatdagi yurishni ot soat strelkasi bo'ylab amalga oshirsin. Izox uchun quyidagi tasvirlar keltirilmoqda.

		8		1	
	7				2
			...		
	6				3
		5		4	

1		5
4		2
	6	
	3	

1	10	
4	7	2
9		5
6	3	8

1	6	
4	9	2
7		5
10	3	8

...

8	1	10
11	4	7
6	9	2
3	12	5

Masala uchun ma`lumotlar strukturasi quyidagicha tashkil qilinadi:

```
Const Nq ; Mq ;
Dx:Array[1..8] of integer=(-2, -1, 1, 2, 2, 1, -1, -2);
Dy:Array[1. .8] of integer=(1, 2, 2, 1, -1, -2, -2, -1);
Var A:Array [-1. .N+2,-1. .M+2] of integer;
t : integer;
```

Dasturning asosiy vazifani bajaruvchi parchasi hisoblangan *Solve* protsedurasining matni quyidagicha.

```
Procedure Solve(k, u, q: integer);
var z , i, j: integer;
begin
A[x, y]:=q;
if q=N*M then Inc(t)
        else for z:=1 To 8 do begin
    i:=x+Dx[z]; j:=y+Dy[z];
    if A[i, j]=0 then Solve (i, j, q+1) end;
    A[x, y]:=0;
```

end;

Quyida asosiy dasturning bir qismi keltirilmoqda:

```
for i:=-1 to N+2  do
for  j:=-1 to M+2 do A[i, j] :=-1; {* ortiqcha if buyruqlarini yozishdan qutulish uchun to'siq elementlar yozilmoqda*}
for i:=1 to N do
for j:=1 to M do A[i, j] :=0 ;
t:=0;
for  i:=1 to N do
for  j:=1 to M  do Sol ve (i, j, 1) ;
writeln('ot bilan shaxmat taxtasini aylanib chiqishlar soni ', N,'*', M '-', t);
```

Agar ihtiyoriy N va M sonlari uchun ot yordamida shaxmat taxtasini aylanib chiqishlar soni jadvalini qurish uchun eng zamonaviy kompyuterlarning tezligi ham yetarli bo'lmasligi mumkin.

Shaxmat taxtasini ot bilan aylanib chiqishning bitta variantini aniqlashga urinib ko'raylik. Bunda 150 yil ldin Varnsdorf tomonidan taklif qilingan navbatdagi yurishni tanlash usulidan foydalanish mumkin. Bu usulning g'oyasi ot o'zi turgan katakdan bo'sh bo'lgan kataklarga yurishlar soni minimal bo'ladigan katakka yurishi bilan bog'liq. Agar bunday kataklar bir nechta bo'lsa, ulardan ihtiyoriy biriga yurish mumkin. Bu holda birinchi bo'lib burchak kataklari band qilinadi va "orqaga qaytishlar" soni yetarlicha darajada kamayadi. Qaralgan holat uchun *Solve* prosedurasi quyidagicha yoziladi.

```
procedure Solve (x, y, q: integer) ;
var W: Array [1..8] of integer;
```

```
xn, yn, i, j, m, mln: integer ;
begin
A[x, y]:=q;
if (q<N*M) then begin
for i:=l to 8 do begin {*W massivni shakllantirish*}
 W[i]:=0; xn:=x+Dx[i ];
yn :=y+Dy[i ] ;
if (A[xn, yn]=0) then begin
for j:=l to 8 do
if (A[xn+Dx]/J, yn +Dy[j]]=0) then Inc(W[i]) ;
end else W[i]:=-1; end;
i:=l;
while (i<=8) do begin min:=Maxint;
t:=1;{*turgan joyidan eng kam yurish mumkin bo'lgan katak
izlanmoqda*}
for j:=2 to 8 do if W[j]<min then begin
m:=j; min:=W[j]; end;
if (W[m]>=0) and (W[m]<Maxint) then
begin Solve (x+Dx[m], y+Dy[m], l+1);
 W[m]:=Maxint; {*qarab chiqishda  foydalanilgan katak
belgilanmoqda*}
end; Inc(i) ;
end;
end else begin (yechimlarni chop qilish);
```

```
halt; end;
A[x,u]:=0;
end;
```

12-§. Orqaga qaytib dasturlash usuli

Ko'pincha masalalar uchun dasturlar ishlab chiqishda odatiy usullardan foydalaniladi. Agar masalalarning algoritmlari oldindan ma`lum bo'lsa, dasturchi ana shu algoritmlarga suyanishi mumkin. Bunday tashqari, quyidan yuqoriga qarab dasturlash, boshlang'ich masalani bir biriga o'xshash modullarga ajratish kabi usullar ham mavjud. Dasturlashda keng qo'llash mumkin bo'lgan usullar yana biri iteratsiya usul bo'lib, unda qo'yilgan masalaning biror boshlang'ich yechimi toki kutilgan natijaga erishguncha ketma-ket (qadamba-qadam) yaxshilab boriladi. Bu usul ayniqsa "Sonli usullar" fanining ko'plab masalalari uchun yaxshi natija beradi.

Dasturlash masalalari boshqa fan masalalaridan shunisi bilan farqlanadiki, amalda ularning xar biriga alohida yondoshuv hamda ijodiy fikrlash talab qilinadi. Ta`bir joiz bo'lsa aytish mumkinki, kichik kashfiyot qilishga. ya'ni har bir masala uchun dastur ishlab chiqish jarayonida yangicha nostandart usullar o'ylab topishga to'g'ri keladi. Biz ushbu maqolada ana shunday usullardan ayrimlari haqida o'z tavsiyalarimizni bayon etamiz.

Hususiy maqsadlar usuli. Bu masalalar yechishning shunday usuliki, boshlang'ich masala yechimlari birgalikda qo'yilgan masalaning yechimini kafolatlovchi bir nechta kichik masalalarga (hususiy maqsadlarni belgilash) ajratiladi.

Teskarisidan qayta ishlash usuli. Dastur ishlab chiqshning bu usulida masalaning yechimi topildi degan nuqtai – nazarga asoslangan xolda bu yechimga erishish uchun zarur bo'lgan shart-sharoitlar ketma-ket aniqlab boriladi.

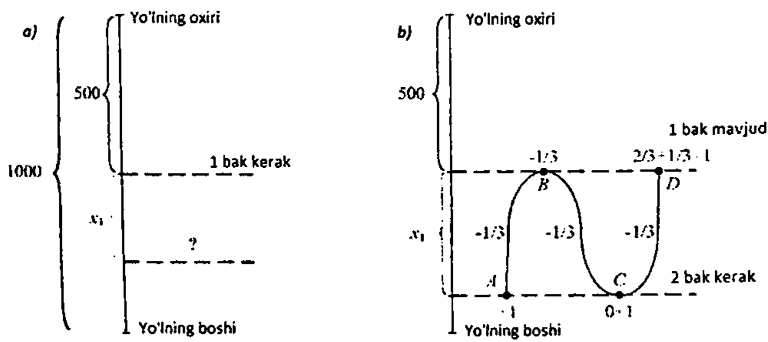
Shuni alohida ta`kidlash joizki, dastur ishlab chiqish birorta xam usulni sof xolda qo'llab bo'lmaydi, ya`ni alohida olingan bitta usul masalaning kutilgan yechimiga olib bormaydi. Odatda, har bir masalani yechish uchun bir qator usullar tanlab olinadi va ulardan birgalikda foydalaniladi. Boshqacha aytganda, usullar masalani yechish jarayoniga yondoshuvni belgilab beradi xolos. Nostandart yondoshuvga namuna sifatida quyidagi masalani keltiramiz.

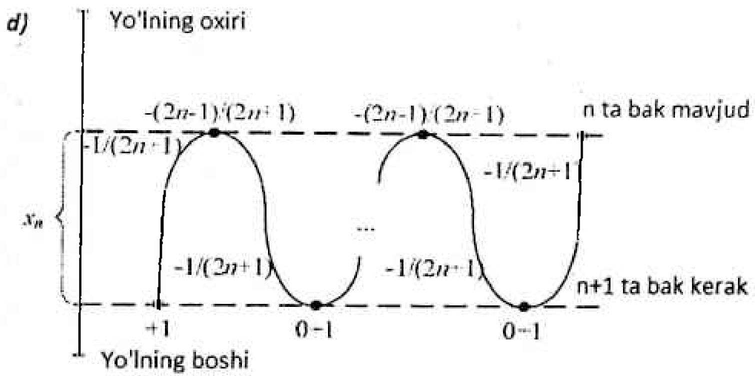
Mashina haqidagi masala. Sayohatchi o'z mashinasida kengligi 1000 km bo'lgan cho'lni kesib o'tishi lozim. Uning mashinasiga hajmi 500 litr bo'lgan bak o'rnatilgan xamda yo'lning xar 1 kilometriga 1 litrdan yoqilg'i sarf qiladi. Yo'l yoqalab yoqilg'i zahirasini tayyorlash uchun yetarli sharoit mavjud. Sayohatchi ma`lum bir masofaga yoqilg'i olib borishi va qoldirishi, shunigdek, yoqilg'i quyish uchun orqaga qaytishi mumkin. Sayohatchi cho'lni kesib o'tishi uchun zarur bo`lgan minimal yoqilg`i miqdorini aniqlang.

Masalaning yechish g'oyasi. Tabiiyki, bu masalaga nisbatan odatiy usullarni qo'llash kutilgan natijaga olib bormasligi ko'rinib turibdi. Shuning uchun, masalani yechish jarayoniga boshqacha usulda yondoshish talab qilinadi. Ma`lumki, cho`lni bir martada kesib o'tishning iloji yo'q. Shuning uchun, sayohatchidan cho'lning ma`lum bir masofasigacha borishi, u yerda yoqilg`ining ma`lum bir miqdorini qoldirib zahira tayyorlashi xamda orqaga qaytib kelishi uchun yetarli bo'lgan miqdordagi yoqilg'ini ham nazarda tutishi talab qilinadi. Shu tariqa, sayohatchi yo`lni qismlarga ajratadi va oldinga-orqaga bir necha marta borib kelib, yetarli zahira tayyorlagan xolda cho`lga ichkarilab boradi. U har gal yo'lning boshiga emas, balki yo`lning zahira tayyorlangan avvalgi qismiga qaytadi va shu yerdan yoqilg`i quyib, o'z xarakatini boshlaydi. Shunday qilib, sayohatchi 1000 km lik masofani xar birida yetarli yoqilg'i zahirasi mavjud bo'lgan $x_1, x_2, ..., x_k$ qismlarga ajratishi lozim.

Faraz qilaylik, cho'l kesib o'tilgan bo'lsin (teskarisidan qayta ishlash usuli). Masaladagi yoqilg'i sarfiini minimallashtirish shartiga ko'ra cho'lning oxirigi yetish uchun bakni to'la bo'shatish talab qilinadi. Buning uchun oxirgi zahira cho'lning 500 kilometrida tashkil qilinishi lozim. Yoqilg'ining minimal bo'lishi talabiga ko'ra, bak yo'lning oxirgi qismini kesib o'tishdan oldin ham bo'sh bo'lishi kerak.

Mashina haqidagi masalaning qadamlar ketma-ketligi 1-rasmda tasvirlangan. Shunday qilib, yo'lning oxirgi qismi uchun yoqilg'i miqdori va o'rni (cho'lning 500-chi kilometrida 500 litr) ma`lum bo'ldi. Undan avvalgi zahira va masofa qanday bo'ladi? Bu masofa oxirgi 500-chi kilometrda 500 litr zahirani tayyorlash uchun minimal yoqilg'ini ta`minlashga yetarli bo'lishi lozim. Bu masofani x_1 bilan (1-a rasm) belgilaymiz. Shu bilan biz birinchi hususiy maqsadni aniqladik.





1-rasm. Mashina haqidagi masalaning yechimi: ***a*** - hususiy maqsadning qo'yilishi; ***b*** -hususiy masalaning yechimi: x_1- ni hisoblash; ***s*** – iteratsiya: x2- ni hisoblash; ***d*** – iteratsiya: x_n ni hisoblash.

Endi qo'yilgan hususiy masalani yechishga urinib ko'ramiz. Faraz qilaylik, cho'l oxirigacha 500+x_1 km masofa qolguncha yetarlicha katta yoqilg'i zahirasi mavjud bo'lsin va u yerga bo'sh bak bilan yetib kelish talab qilinadi. Bu vaziyat 1-b rasmdagi A nuqtaga mos keladi. Bu nuqtada sayohatchi (1-b rasmdagi +1 nuqta) bakni to'ldirib, B nuqtaga (oxirgi zahira tayyorlash nuqtasi) yetib keladi. Bu nuqtada orqaga (avvalgi nuqtaga) qaytib kelish uchun bakda yetarli yoqilg'ini saqlagan xolda, ma'lum bir miqdordagi yoqilg'ini qoldirish lozim. S nuqtada bak bo'shaydi va sayohatchi uni to'ldirib, yana oxirgi zahiraga (D nuqta) qarab yo'lga chiqadi. U yerda oxirgi qadamda qoldirilgan yoqilg'ini oladi. Bu miqdor roppa-rosa 500 litr bo'lishi lozim va u cho'lning qolgan qismini kesib o'tish uchun yetarli bo'ladi.

Shunday qilib, sayohatchi b x_1 masofani uch marta bosib o'tadi va buning uchun 500 litr yoqilg'i sarflaydi. Bundan tashqari, cho'lning qolgan qismini bosib o'tish uchun 500 litr zahira tayyorlaydi. Demak, 500+x_1 km yo'lni bosib o'tish va yetarli zahira tayyorlash uchun 1000 litr yoqilg'i sarflandi. Bundan $3x_1+500 = 1000$ xamda $x_1=500/3$ ekanligi kelib chiqadi.

Bu miqdor bakning uchdan bir qismini tashki qiladi. 1.b-rasmda yoqilg'i xarajatlari tasvirlangan: uzunligi x_1 bo'lgan yo'lga bir bakning uchdan bir qismi sarf qilinadi, uchdan bir qismi zahiraga qoldiriladi, qolgan yoqilg'i bilan avvalgi nuqtaga qaytib keladi. Bakni yoqilg'i bilan to'ldirib, uchdan bir qism yoqilg'i sarflab, yana x_1 masofa bosib o'tiladi. Bunda bakda uchdan ikki qism yoqilg'i qoladi. Sayohatchi uning ustiga zahirada turgan uchdan bir qism yoqilg'ini quyib bakni to'ldiradi va cho'lning qolgan 500 km masofasini bemalol bosib o'tadi.

Shunday qilib, sayohatchi birinchi hususiy masalani hal qildi. Endi u ikkinchi hususiy masalaga o'tishi mumkin. Bu masala cho'l oxiridan $500+x_1=500+500/3$ km masofaga ikki bak yoqilg'i olib borishdan iborat bo'ladi. Bu masalani hal qilishda xuddi avvalgi masala kabi fikr yuritish mumkin. Xarakatlar sxemasi 1.s-rasmda keltirilgan. Bu xolda sayohatchi x_1 va x_2 punktlar orasida 5 marta borib kelishi zarur bo'ladi. Har bir tomonga borish uchun 1/5 qism bak yoqilg'i sarf qilinadi. Zahiraga esa 3/5 qism yoqilg'ini qoldirish zarur. Beshinchi marta yurishdan so'ng, cho'lning oxiridan $500+x_1$ masofada ikkita bakni to'ldirish uchun yetarli zahira tayyorlandi. Navbatdagi hususiy masala sifatida cho'l oxiridan $500+x_1+x_2$ masofaga 3 bak yoqilg'i tayyorlash olinadi. Bu yerda $x_2 = 500/5$.

Keyingi xarakatlar 1.d-rasmda bayon qilingan. Osongina ko'rish mumkinki, x_n nuqtada $n+1$ bakdan iborat zahira tashkil qilish zarur bo'lib, buning uchun avvalgi va joriy nuqtalar o'rtasida $2n+1$ marta borib kelish talab qilinadi. U xolda

$$x_n = 500/(2n + 1)$$

yoki sayohatchi bir tomonga borish uchun bir bakning $1/(2n + 1)$ qismini sarf qilib, zahirada

$$1-\frac{2}{2n+1}=\frac{2n-1}{2n+1}$$

miqdordagi yoqilg'ini qoldiradi. $2n+1$ marta borib kelishdan so'ng, zahira qilingan yoqilg'i miqdori $n(2n-1)/(2n+1)$ ga teng bo'ladi. Oxirgi borishda bakda $2n/(2n+1)$ qism yoqilg'i qoladi va xammasi bo'lib, x_n nuqtada

$$\frac{2n}{2n+1}+\frac{n(2n-1)}{2n+1}=\frac{n(2n+1)}{2n+1}=n$$

bak yoqilg'i zahirasi tayyorlanadi.

Masalaning yakuniy yechimini topish uchun sayohatchi tayyorlashi zarur bo'lgan k zahiralar sonini aniqlashi lozim. Unga mos ravishda zahira nuqtalari cho'lning oxiridan boshlab $500+x_1+x_2+x_3+...$ tarzida tashkil qilinadi. Demak, sayohatchi cho'lni bosib o'tishi uchun, uning ichkarilagan masofasi 1000 km dan ziyod bo'lishi shart, ya`ni

$$\sum_{i=1}^{k}\frac{500}{2i+1}\geq 1000.$$

Qo'yilgan masalani hal qilish uchun biz asosan uchta usullarni, ya`ni hususiy masalalar, teskarisidan qayta ishlash xamda iteratsiya metodlarini qo'lladik va bir qaraganda o'ta murakkab bo'lgan masalani oddiy sonli qatorning dastlabki k-xadlari yig'indisini xisoblash masalasiga keltirdik. Bu masala uchun hattoki yosh dasturlar ham osongina dastur ishlab chiqishlari mumkin. Shunday bo'lsa-da, biz ushbu masala uchun algoritm psevdokodi quyidagicha yoziladi:

Algoritm sayohatchi

// **kiruvchi ma'lumot'lar:** S – masofa (1000 km)

// **chiquvchi ma'lumotlar:** shahobchasi nomeri va benzin sarfi

S←0; i←0;

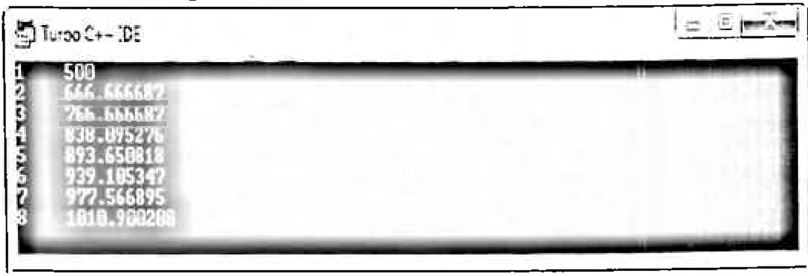
while (s<=1000)

s←s+500/(2*i+1);

i←i+1;

chiqarish i , s;

Ushbu algoritm asosida qurilgan dastur quyidagi natijani beradi:



Demak, sayohatchi hammasi bo'lib 8 ta zahira nuqtalarini tashkil qilishi lozim va bu nuqtalar cho'l oxiridan qanday masofalarda joylashishi dastur natijalaridan ko'rinib turibdiki.

Yuqoridagi mulohalalar masalalar uchun dastur ishlab chiqish jarayoniga nostandart usullar bilan yondoshish dasturchilarga katta qulayliklar taqdim etishi mumkin ekanligidan dalolat beradi.

MUNDARIJA

Otaxanov Nurillo Abdumalokivich

ALGORITM QURISH METODLARI

Terishga berildi 01.09.2019.
Bosishga ruxsat etildi 15.09.2019 y.
Bichimi 60x42 1/8, Hajmi 2 bosma taboq.
Adadi 100 nusxa. Bahosi kelishilgan narxda.

"NAMANGAN" nashriyoti
Namangan shahri, Navoiy ko`chasi, 36-uy
Nashriyot litsenziya raqami AI – 156
2009-yil 14-avgustda berilgan

"УАС" MCHJ bosmaxonasida chop etildi
Manzil: *Namangan shahri, A.Navoiy ko`chasi, 72-uy.*

ISBN 978-9943-5644-5-9