O`ZBEKISTON RESPUBLIKASI XALQ TA'LIMI VAZIRLIGI

NAVOIY DAVLAT PEDAGOGIKA INSTITUTI

A.A.Ibragimov

"Algoritmlar nazariyasi" fanidan amaliy mashg'ulotlar uchun uslubiy qo`llanma

Navoiy-2014

Ibragimov A.A. Algoritmlar nazariyasi fanidan amaliy mashg`ulotlar uchun uslubiy qo`llanma/ Navoiy: NDPI nashriyoti, 2014.- 90 b.

Ushbu uslubiy qo`llanma Navoiy DPI 5A110701 - "*Ta'limda axborot texnologiyalari*" mutaxassisligi magistrantlari uchun talaba tanlov fani sifatida o`qitilayotgan "*Algoritmlar nazariyasi*" fani dasturi asosida yozilgan bo`lib, mazkur fan bo`yicha amaliy mashgulotlarni olib borishga mo`ljallangan.

Unda algoritmlarning xossalari tahlili (Yevklid algoritmi va boshqa algoritmlar misolida), algoritmik hisoblash jarayoni, algorimtlarni formallashtirish usullari (Tyuring mashinasi va Markovning normal algoritmlariga doir masalalar yechish), muayyan masalani yechish uchun algoritmning mavjudligi haqida tushunchalar, algoritmlarni murakkablikka, o`sish tezliklariga nisbatan tahlil qilish, izlash va saralash algorimtlari hamda determinallanmagan algoritmlar tushunchalariga oid masalalar qaralgan.

Uslubiy qo`llanmadagi barcha mavzular so`nggi yillarda chop etilgan ilmiy-uslubiy adabiyotlar hamda fan yutuqlariga tayangan holda ishlab chiqilgan bo`lib, undan oliy ta'lim muassasalarining 5110700-*"Informatika o`qitish metodikasi"*, 5480100-*"Amaliy matematika va informatika"* yo`nalishi bakalavriat talabalari "Algoritmlash va dasturlash tillari", "Algotitmlar nazariyasi" fanlari bo`yicha, shuningdek, tadqiqotchi-izlanuvchilar foydalanishlari mumkin.

Taqrizchilar:

Bazarov M.B. – texnika fanlari doktori, professor,

Imomkulov S.A. – fizika-matematika fanlari doktori, professor.

Uslubiy qo`llanma Navoiy davlat pedagogika instituti Ilmiy-uslubiy kengashi tomonidan nashrga tavsiya qilingan (2014 yil _____ oyi __-sonli qarori)

So`z boshi

Talabalarni informatika fani yoki umumiy holda qaraydigan bo`lsak, axborot texnologiyalari sohasida yetuk mutaxassis qilib tayyorlashda algoritm, dasturlash texnologiyalari muhim ahamiyat kasb etadi.

Algoritmlar nazariyasi – Informatika va tadbiqiy matematikaning fundamental qismiga oid fan bo`lib, uning davomi bevosita samarali dasturlarni tuzish, sonli usullar, mukammallashtirish usullari va ob'ektga yo`naltirilgan dasturlash sohalarida o`z nazariy va amaliy tadbiqini topadi.

"*Algoritmlar nazariyasi*" fani jarayonlarning matematik modellarini tadqiq qilish usullari ustida ish olib boradi. Matematik va kompyuterli modellarni tadqiq qilish jarayonining natijaviyligini o`rgatishda asosiy uslub sifatida qaraladi. Jumladan, "*Algoritmlar nazariyasi*" uslublari asosida muayyan masalaga doir mavjud algoritmlarning ichida eng samaralisi ajratib olinadi.

Mazkur uslubiy qo`llanma "Algoritmlar nazariyasi" fanidan amaliy mashg`ulotlarni olib borishga mo`ljallangan bo`lib, unda algoritmning intuitiv va kibernetik ma'nodagi ta'riflari, algoritm xossalarini aniq masalalar algoritmlari asosida tahlil qilish usullari, algoritmik hisoblash jarayoni amaliyoti, algoritmlarni formallashtirish usullari bo`lgan Tyuring mashinasi va Markovning normal algoritmi, algoritm tahlili asoslari, izlash va saralash algoritmlari, sonli algoritmlar, determinallanmagan algoritmlar va shunga o`xshash boshqa muhim tushunchalar bayon etilgan. Fan dasturi asosida rejalashtirilgan har bir amaliy mashg`ulotning mavzusining maqsadi, asosiy nazariy tushunchalari (Nazariy qismda) va unga doir masalalar yechish (Amaliy qism) hamda mustaqil bajarish uchun topshiriqlar berilgan.

1 -AMALIY MASHG`ULOT: Algoritm xossalarining tahlili. Yevklid algoritmi.

Ishning maqsadi: Algoritmlarning xossalarini aniq misollar yordamida tahlil qilish.

I. Nazariy qism

Algoritm *– bu qoyidalarning qat'iy va chekli tizimi bo`lib, ba'zi ob'ektlar ustida bajariladigan amallarni aniqlaydi va chekli qadamdan keyin natijaga olib kelishini ta'minlaydi.*

Ixtiyoriy **algoritm** 5 ta muhim xossalarga ega:

- *Tushunarliligi* – algoritm ijrochi imkoniyatlariga moslangan holda, ya'ni ijrochi uchun tushunarli tarzda bo`lishi kerak.
- Algoritmning *aniqligi* – har bir qadam bajarilishining bir qiymatliligi.
- *Diskretliligi* – masalani yechish jarayonini bajarilish vaqtida kompyuter yoki insonga qiyinchilik tug`dirmasligi uchun bir necha sodda bosqichlar (bajarilish qadamlari)ga bo`lish.
- *Ommaviylik* – algoritmning bitta emas, balki shunday masalalar sinfini yechish uchun foydaliligi.
- *Natijaviylik* – chekli qadamlardan keyin dastlabki ma'lumotlar asosida natijani olishga imkon beruvchi algoritmning harakatlar yakuni.

Amaliyotda quyidagi ***algoritm*** turlari mavjud***:***

Chiziqli – amallar ketma-ket, biror-bir shart tekshirilmasdan bajariluvchi algoritm.

Tarmoqlanuvchi – belgilangan shartlarning o`zgarishiga bog`liq holda ko`rsatmalarning variantlari oldindan mo`ljallanadigan algoritm.

Takrorlanuvchi (Siklik) – alohida jarayonlar yoki jarayonlar guruhi bir necha marta bajariladigan algoritm.

Algoritmni yozish usullari: *so`zli, formulali, jadvalli, grafik.*

II. Amaliy qism.

Algoritmga oddiy matematik masalalarda arifmetik amallarni bajarish qoidalari: eng katta umumiy bo`luvchini topish; kvadrat tenglamaning ildizlarini topish; funksiyaning hosilasini topish qoidalari va hokazolar misol bo`ladi.

Yuqorida keltirilgan misollardan ko`rinadi-ki, algoritm tushunchasi bir xil tipli masalalar to`plamiga qo`llaniladi. Bunday bir xil masalalar ommaviy muammo deyiladi. Masalan, $ax^2 + bx + c=0$ ko`rinishdagi kvadrat tenglamalarni yechish masalasi ommaviy muammodir. Chunki biz a, b, c larni o`zgartirib bir xil tipli masalalar sinfini hosil qilamiz. Algoritm tushunchasiga aniq matematik ta'rif berish ancha mushkul masala bo`lganligi sababli, uning xarakterli xususiyatlarini Yevklid algoritmi va kvadrat tenglamani yechish algoritmi misolida tahlil qilamiz.

Yevklid algoritmida a va b sonlarning eng katta umumiy bo`luvchisi ushbu sonlar ayirmasining eng katta bo`luvchisi hamda ikkala a va b sonlarning ham eng katta umumiy bo`luvchisi bo`lish faktidan foydalanilgan.

Yevklid algoritmining bu ifodasiga aniqlik yetishmaydi, shuning uchun uni aniqlashtirish zarur bo`ladi.

Haqiqiy Yevklid algoritmi quyidagicha:

1. *a sonni birinchi son deb, b sonni ikkinchi son deb qaralsin.*
2. *Birinchi va ikkinchi sonlarni taqqoslang. Agar ular teng bo`lsa, 5-qadamga, aks holda 3-qadamga o`tilsin.*
3. *Agar birinchi son ikkinchi sondan kichik bo`lsa, ularning o`rni almashtirilsin.*
4. *Birinchi sondan ikkinchi son ayirilsin va ayirma birinchi son deb hisoblansin. 2-qadamga o`tilsin.*
5. *Birinchi sonni natija sifatida qabul qilinsin. Tamom.*

Bu qoidalar ketma-ketligi algoritmni tashkil etadi, chunki ularni bajargan ixtiyoriy, ya'ni ayirishni biladigan kishi ixtiyoriy sonlar jufti uchun eng katta umumiy bo`luvchini topa oladi.

Algoritmning diskretligi. Har bir algoritm qandaydir miqdorlarning boshlang`ich qiymatlarida ish boshlab, diskret rejimda ishlaydi. Ma'lum bir vaqt momentida miqdorlarning boshqa qiymatlariga o`tadi.

Masalan, Yevklid algoritmidan foydalanib, a va b sonlarning eng katta umumiy bo`luvchisini topaylik,

$$a = b \cdot q_0 + r_1;$$
$$b = r_1 \cdot q_1 + r_2;$$

$r_1 = r_2 \cdot q_2 + r_3$;
. . . . . .
$r_{n-3} = r_{n-2} \cdot q_{n-2} + r_{n-1}$;
$r_{n-2} = r_{n-1} \cdot q_{n-1} + r_n$;
$r_{n-1} = r_n \cdot q_n$.

Bundan $(a, b) = r_n$. Ko`rinib turibdi-ki, $a$ va $b$ sonlarning eng katta umumiy bo`luvchisini topishda (a, b) miqdorlarning boshlang`ich qiymati, keyingi qiymati $(b, r_1)$ va h.k. $(r_n, 0)$ miqdorlarning oxirgi qiymati bo`ladi.

***Algoritmning to`liq aniqlanganligi.*** Algoritmdagi kattaliklar sistemasining qiymatlari, o`zidan oldingi qiymatlari orqali to`liq aniqlanadi. Yuqoridagi misolda ko`rganimizdek:

(b, r_1) qiymatlar (a, b) orqali to`liq aniqlangan va h.k.

(r_{n-2}, r_{n-1}) esa (r_{n-3}, r_{n-2}) orqali ;

(r_{n-1}, r_n) esa (r_{n-2}, r_{n-1}) orqali ;

$(r_n, 0)$ esa (r_{n-1}, r_n) orqali to`liq aniqlangan.

Algoritmning tushunarliligi. Algoritm o`z tabiatiga ko`ra ishlash jarayoni sodda qadamlardan iborat. Buni ham yuqoridagi va boshqa misollardan ko`rish mumkin.

Algoritmning ommaviyligi. Bu haqda yuqorida aytganimizdek, har bir algoritm qandaydir masalalar sinfini yechishga mo`ljallangandir.

Algoritmning natijaviyligi. Miqdorlar qiymatlarini qurish jarayoni chekli qadamdan so`ng natija berishi lozim. Masalan, $ax^2+bx+c=0$ kvadrat tenglamani haqiqiy sonlar to`plamida yechish algoritmini misol sifatida oladigan bo`lsak, $D=b^2-4ac\geq 0$, bo`lganda ikkita yechim hosil qilamiz. Bu yechimlar algoritmning natijasiga aylanadi. Agar $D<0$, tenglamaning haqiqiy ildizlari yo`q bo`lib, algoritmning natijasi sifatida «tenglama haqiqiy ildizlarga ega emas», degan jumla olinadi.

MAVZU BO`YICHA TOPSHIRIQLAR

1. Algoritmning ta’rifini keltiring.
2. Algoritmning berilish usullarini ayting va misollar yozing.

3. Algoritmning turlarini ayting va misollar keltiring.

4. Ikki sonning eng kichik umumiy karralisini topish algoritmini tuzing.

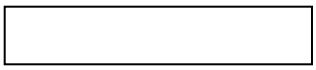
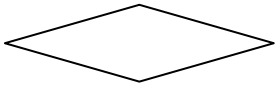
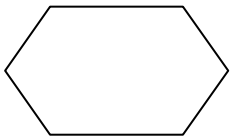
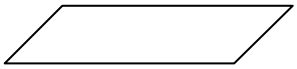
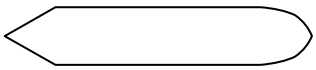
5. Algoritm xossalarini misollar yordamida tavsiflang.

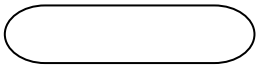
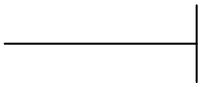
2 -AMALIY MASHG`ULOT: Algoritm blok-sxemasi.

Ishning maqsadi: Algoritm blok-sxemalarini tuzish amaliyotini o`tkazish. Algoritm blok-sxemasida ishlatiladigan figuralarning vazifalari, hisoblash yo`nalishlarini belgilash va chiziqli, tarmoqlanuvchi, takrorlanuvchi hamda rekurrent algoritmlar uchun blok-sxemalar yaratish malakasini hosil qilish.

I. Nazariy qism

Blok-sxemalarni tuzishda foydalaniladigan asosiy sodda geometrik figuralar quyidagilardan iborat:

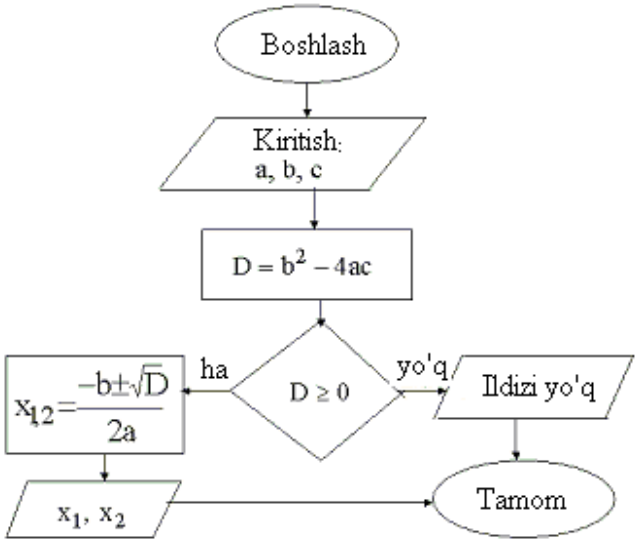
Nomi	Figura	Bajaradigan vazifasi
Jarayon		Bir yoki bir nechta amallarning bajarilishi natijasida ma’lumotlarning o`zgarishi
Qaror		Biror shartga bog`liq ravishda algoritmning bajarilish yo`nalishini tanlash
Shakl o`zgartirish		Dasturni o`zgartiruvchi buyruq yoki buyruqlar turkumini o`zgartirish amalini bajarish
Avval aniqlangan jarayon		Oldindan ishlab chiqilgan dastur (qism dastur) yoki algoritmdan foydalanish
Kiritish yoki chiqarish		Axborotlarni qayta ishlash mumkin bo`lgan shaklga o`tkazish yoki olingan natijani tasvirlash
Displey		EHMga ulangan displeydan axborotlarni kiritish yoki chikarish
Hujjat		Axborotlarni qog`ozga chiqarish yoki qog`ozdan kiritish
Axborotlar oqimi chizig`i		Bloklar orasidagi bog`lanishlarni tasvirlash

Bog`lagich		Uzilib qolgan axborot oqimlarini ulash belgisi
Boshlash yoki tugatish		Axborotni qayta ishlashni boshlash, vaqtincha yoki butunlay to`xtatish
Izoh		Bloklarga tegishli turli xildagi tushuntirishlar

Blok-sxemalar bilan ishlashni yaxshilab o`zlashtirib olish zarur, chunki bu usul algoritmlarni ifodalashning qulay vositalaridan biri bo`lib dastur tuzishni osonlashtiradi, dasturlash qobiliyatini mustahkamlaydi. Algoritmik tillarda blok-sxemaning asosiy strukturalariga maxsus operatorlar mos keladi.

II. Amaliy qism

Shuni aytish kerakki, blok-sxemalardagi yozuvlar odatdagi yozuvlardan katta farq qilmaydi. Misol sifatida $ax^2+bx+c=0$ kvadrat tenglamani yechish algoritmining blok-sxemasi quyida keltirilgan.



1-rasm. Kvadrat tenglamani yechish algoritmi.

Chiziqli algoritmlar. Har qanday murakkab algoritmni ham uchta asosiy struktura yordamida tasvirlash mumkin. Bular ketma-ketlik, ayri va takrorlash strukturalaridir. Bu strukturalar asosida chiziqli, tarmoqlanuvchi va takrorlanuvchi hisoblash jarayonlarining algoritmlarini tuzish mumkin. Umuman olganda, algoritmlarni shartli ravishda quyidagi turlarga ajratish mumkin:

chiziqli algoritmlar;

tarmoqlanuvchi algoritmlar;

takrorlanuvchi yoki siklik algoritmlar;

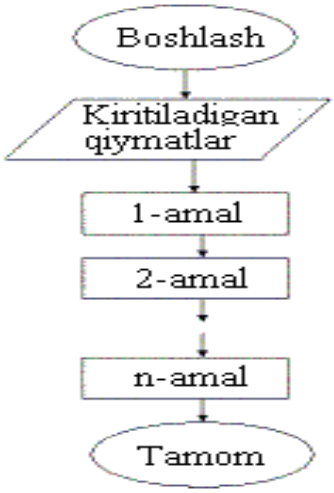
ichma-ich joylashgan siklik algoritmlar;

rekurrent algoritmlar;

takrorlanishlar soni oldindan no`malum algoritmlar;

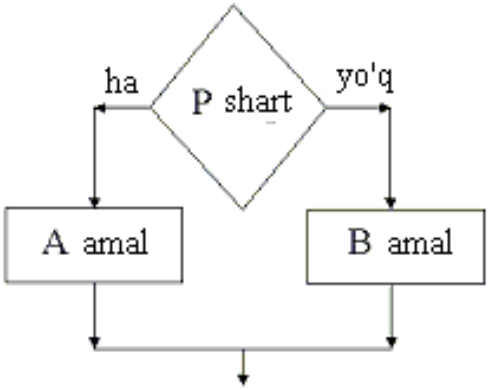
ketma-ket yaqinlashuvchi algoritmlar.

Faqat ketma-ket bajariladigan amallardan tashkil topgan algoritmlarga-*chiziqli algoritmlar* deyiladi. Bunday algoritmni ifodalash uchun ketma-ketlik strukturasi ishlatiladi. Strukturada bajariladigan amal mos keluvchi shakl bilan ko`rsatiladi. Chiziqli algoritmlar blok-sxemasining umumiy strukturasini quyidagi ko`rinishda ifodalash mumkin:



2-rasm. Chiziqli algoritm blok-sxemasi.

Tarmoqlanuvchi algoritmlar. Agar hisoblash jarayoni biror bir berilgan shartning bajarilishiga qarab turli tarmoqlar bo`yicha davom ettirilsa va hisoblash jarayonida har bir tarmoq faqat bir marta bajarilsa, bunday hisoblash jarayonlariga tarmoqlanuvchi algoritmlar deyiladi. Tarmoqlanuvchi algoritmlar uchun ayri strukturasi ishlatiladi. Tarmoqlanuvchi strukturasi berilgan shartning bajarilishiga qarab ko`rsatilgan tarmoqdan faqat bittasining bajarilishini ta’minlaydi.



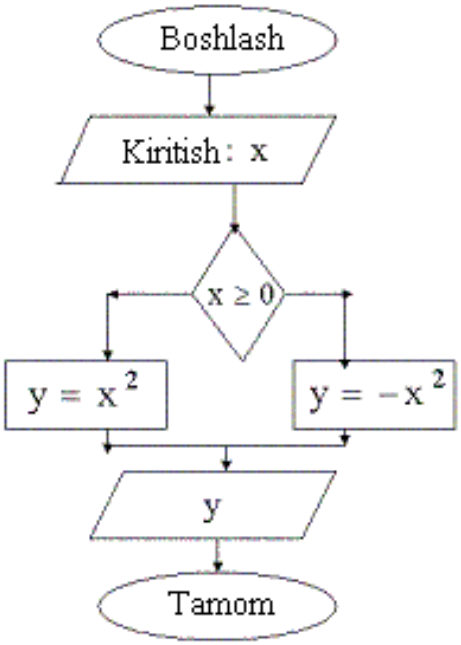
3-rasm. Tarmoqlanishning umumiy ko`rinishi.

Berilgan shart romb orqali ifodalanadi, P - berilgan shart. Agar shart bajarilsa, "ha" tarmoq bo`yicha A amal, shart bajarilmasa "yo`q" tarmoq bo`yicha B amal bajariladi.

Tarmoqlanuvchi algoritmga tipik *misol* sifatida quyidagi sodda *misol*ni qaraylik.

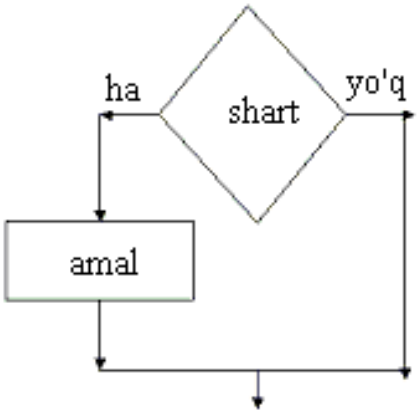
1- Misol: $Y = \begin{cases} x^2 & \text{agar } x \geq 0 \\ -x^2 & \text{agar } x < 0 \end{cases}$

Berilgan x ning qiytmatiga bog`lik holda, agar u musbat bo`lsa «ha» tarmoq bo`yicha $y=x^2$ funksiyaning qiymati, aks holda $y=-x^2$ funksiyaning qiymati hisoblanadi.



4-rasm. Tarmoqlangan funksiya qiymatini hisoblash algoritmi

Ko`pgina masalalarni yechishda, shart asosida tarmoqlanuvchi algoritmlarning ikkita tarmog`idan bittasining, ya'ni yoki «ha» yoki «yo`q» ning bajarilishi yetarli bo`ladi. Bu holat tarmoqlanuvchi algoritmning xususiy holi sifatida *aylanish strukturasi* deb atash mumkin. Aylanish strukturasi quyidagi ko`rinishga ega:



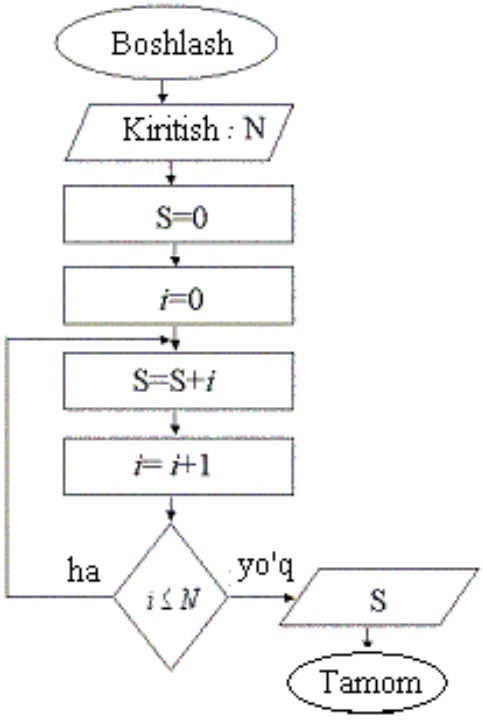
5-rasm. Aylanish strukturasining umumiy ko`rinishi.

Takrorlanuvchi algoritmlar. Agar biror masalani yechish uchun tuzilgan, zarur bo`lgan amallar ketma-ketligining ma'lum bir qismi biror parametrga bog`liq ko`p marta qayta bajarilsa, bunday algoritm *takrorlanuvchi algoritm* yoki *siklik algoritmlar* deyiladi. Takrorlanuvchi algoritmlarga tipik *misol* sifatida odatda qatorlarning yig`indisi yoki ko`patmasini hisoblash jarayonlarini qarash mumkin. Quyidagi yig`indini hisoblash algoritmini tuzaylik.

$$S = 1 + 2 + 3 + \ldots\ldots\ldots + N = \sum_{i=1}^{N} i$$

Bu yig`indini hisoblash uchun $i$=0 da S=0 deb olamiz va $i=i+1$ da $S=S+i^2$ ni hisoblaymiz. Bu yerda birinchi va ikkinchi qadamlar uchun yig`indi hisoblandi va keyingi qadamda i parametr yana bittaga orttiriladi va navbatdagi raqam avvalgi yig`indi $S$ ning ustiga qo`shiladi va bu jarayon shu tartibda to $i<N$ sharti bajarilmaguncha davom ettiriladi va natijada izlangan yig`indiga ega bo`lamiz. Bu fikrlarni quyidagi algoritm sifatida ifodalash mumkin:

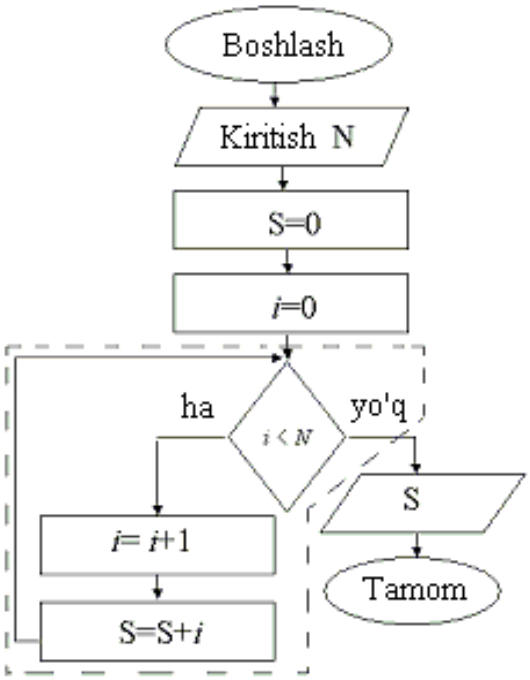
N –berilgan bo`lsin,
$i=0$ berilsin,
$S=0$ berilsin,
$i=i+1$ hisoblansin,
$S=S+i$ hisoblansin,
$i<N$ tekshirilsin va bu shart bajarilsa, 4-satrga qaytilsin, aks holda keyingi qatorga o`tilsin,
S ning qiymati chop etilsin.



6-rasm. 1 dan n gacha bo`lgan sonlar yig`indisini hisoblash algoritmi blok-sxemasi.

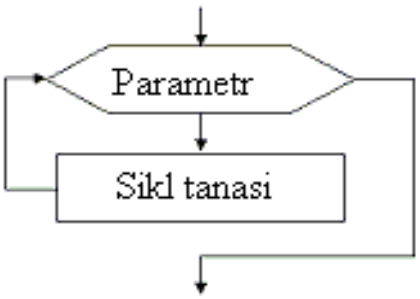
Yuqorida keltirilgan algoritm va blok sxemadan ko`rinib turibdiki amallar ketma-ketligining ma’lum qismi parametr *i* ga nisbatan *N* marta takrorlanayapti. ya’ni shartni oldin tekshiriladigan holatda chizish mumkin edi.

Masalan, yig`indining algoritmini qaraylik. Bu blok sxemaning takrorlanuvchi qismiga quyidagi, sharti oldin berilgan siklik strukturaning mos kelishini ko`rish mumkin.



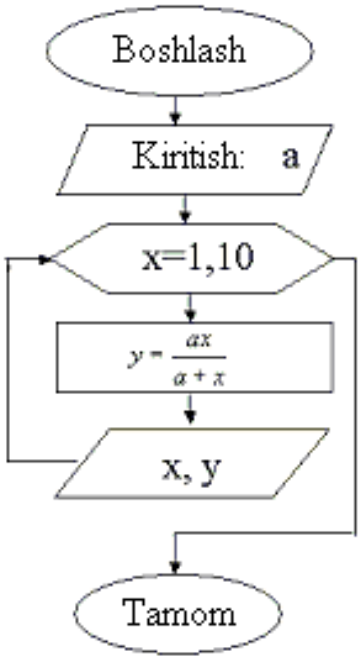
7-rasm. 1 dan *n* gacha bo`lgan sonlar yig`indisini hisoblash algoritmi.

Blok sxemalarining takrorlanuvchi qismlarini, quyidagi parametrli takrorlash strukturasi ko`rinishida ham ifodalash mumkin.



8-rasm. Parametrli takrorlash operatorining umumiy ko`rinishi.

Parametrli takrorlash operatoriga *misol* sifatida berilgan *x=1,2,3,.....10* larda $y = \frac{ax}{a+x}$ funksiyasining qiymatlarini hisoblash blok sxemasini qarash mumkin.

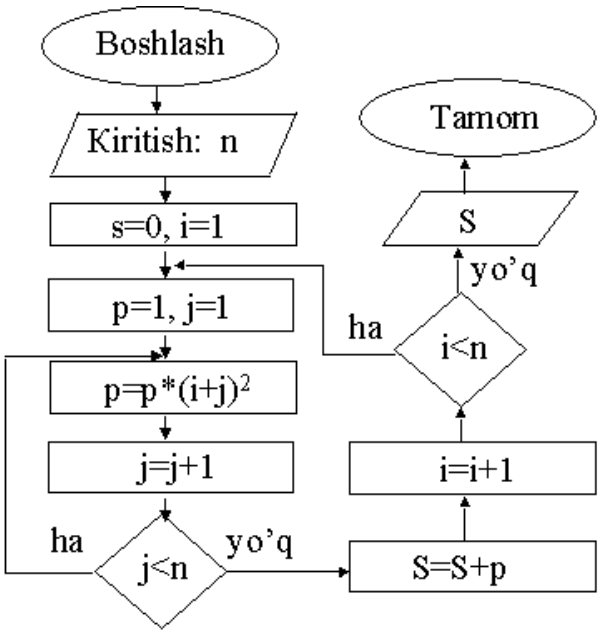


9-rasm. Parametrli takrorlash operatoriga doir algoritm.

Ichma-ich joylashgan siklik algoritmlar. Ba'zan, takrorlanuvchi algoritmlar bir nechta parametrlarga bog`liq bo`ladi. Odatda bunday algoritmlarni ichma-ich joylashgan algortmlar deb ataladi.

Misol sifati berilgan $n \times m$ o`lchovli $a_{ij}$ –matritsa elementlarining yig`indisini hisoblash masalasini qaraylik.

$S = \sum_{i=1}^{n} \prod_{j=1}^{n} (i+j)^2$ Bu yig`indini hisoblash uchun, *i* ning har bir qiymatida *j* bo`yicha ko`paytmani hisoblab, avval yig`indi ustiga ketma-ket qo`shib borish kerak bo`ladi. Bu jarayon quyidagi blok–sxemada aks ettirilgan. Bu yerda *i*-tashqi sikl - yig`indi uchun, *j*-esa ichki sikl-ko`paytmani hosil qilish uchun foydalanilgan.

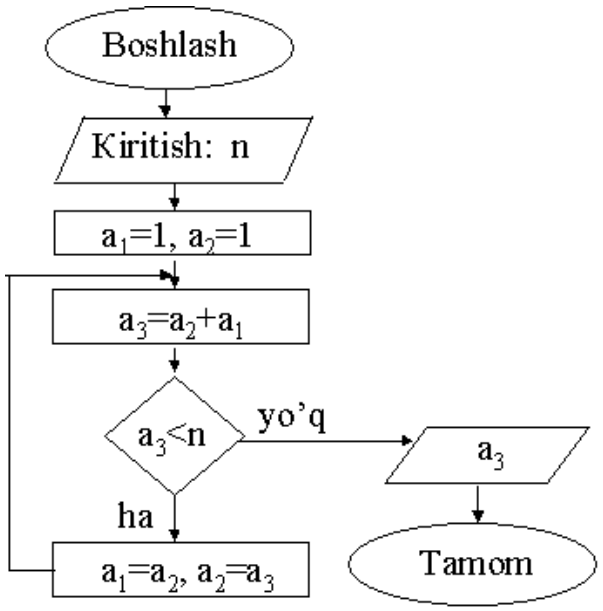


10-rasm. Ichma-ich joylashgan siklik algoritmga doir blok-sxema

Rekurrent algoritmlar. Hisoblash jarayonida ba'zi bir algoritmlarning o`ziga qayta murojaat qilishga to`g`ri keladi. O`ziga–o`zi murojaat qiladigan algoritmlarga *rekurrent algoritmlar* yoki *rekursiya* deb ataladi. Bunday algoritmga *misol* sifatida Fibonachchi sonlarini keltirish mumkin.

Ma'lumki, Fibonachchi sonlari quyidagicha aniqlangan:

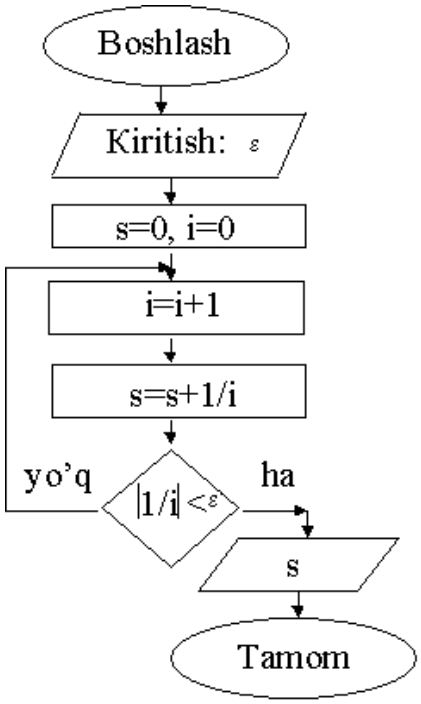
$a_0=a_1=1$, $a_i=a_{i-1}+a_{i-2}$ $i=2,3,4,\ldots$. Bu rekurrent ifoda algoritmiga mos keluvchi blok-sxema 11-rasmda keltirilgan. Eslatib o`tamiz, formuladagi *i*-indeksga hojat yo`q, agar Fibonachchi sonining nomerini ham aniqlash zarur bo`lsa, birorta parametr-kalit kiritish kerak bo`ladi.



11-rasm. Fibonachchi sonlarining *n*- hadini hisoblash algoritmi.

Amalda shunday masalalar uchraydiki, ularda takrorlanishlar soni oldindan berilmagan, ya'ni noma'lum bo`ladi. Ammo, bu jarayonni tugatish uchun biror bir shart berilgan bo`ladi.

Masalan, quyidagi $S = 1 + \frac{1}{2} + \frac{1}{3} + \ldots = \sum_{i=1}^{\infty} \frac{1}{i}$ qatorda nechta had bilan chegaralanish berilmagan. Lekin qatorni ε aniqlikda hisoblash zarur bo`ladi. Buning uchun $\left| \frac{1}{i} \right| < \varepsilon$ shartni olish mumkin.



12-rasm. Takrorlanishlar soni oldindan no`malum bo`lgan algoritmlarga doir blok-sxema.

Ketma-ket yaqinlashuvchi yoki iteratsion algoritmlar. Yuqori tartibli algebraik va transsendent tenglamalarni yechish usullari yoki algoritmlari ketma-ket yaqinlashuvchi – interatsion algoritmlarga *misol* bo`la oladi. Ma'lumki, transsendent tenglamalarni yechishning quyidagi asosiy usullari mavjud:

- *Urinmalar usuli (Nyuton usuli),*
- *Ketma-ket yaqinlashishi usuli,*
- *Vatarlar usuli,*
- *Kesmani teng ikkiga bo`lish usuli.*

Bizga

$$f(x)=0 \qquad (1)$$

transsendent tenglama berilgan bo`lsin. Faraz qilaylik bu tenglama [a,b] oraliqda uzluksiz va f(a)*f(b)<0 shartni qanoatlantirsin. Ma'lumki, bu holda berilgan tenglama [a,b] orilaqda kamida bitta ildizga ega bo`ladi va u quyidagi formula orqali topiladi.

$$X_{n+1} = X_n - \frac{f(X_n)}{f'(X_n)}, \qquad n = 0,1,2,... \qquad (2)$$

Boshlang`ich x_0 qiymat $f(x_0)f''(x_0)<0$ shart asosida tanlab olinsa, (2) iteratsiya albatta yaqinlashadi. Ketma-ketlik

$$|X_{n+1}-X_n|<\varepsilon$$

shart bajarilgunga qadar davom ettiriladi.

Berilgan musbat *a* haqiqiy sondan kvadrat ildiz chiqarish algoritmi tuzilsin.

Bu masalani yechish uchun kvadrat ildizni *x* deb belgilab olib,

$$\sqrt{a}=x \tag{3}$$

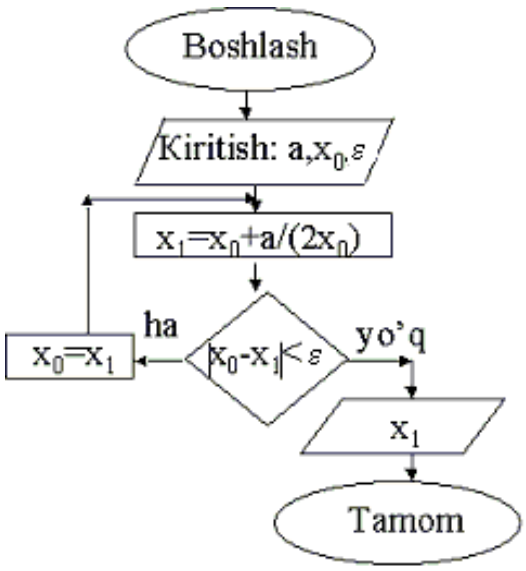
ifodalash yozib olamiz. U holda (1) tenglamaga asosan

$$f(x)=x^2-a \tag{4}$$

ekanligini topish mumkin. (4) ifodani (2) ga qo`yib, quyidagi rekurrent formulani topish mumkin:

$$X_{n+1}=\frac{1}{2}(X_n+\frac{a}{2X_n}) \tag{5}$$

Bu formulaga mos blok-sxema 13-rasmda keltirilgan. ε - kvadrat ildizni topishning berilgan aniqligi. Eslatib o`tamiz, algoritmda indeksli o`zgaruvchilarga zarurat yo`q.



13-rasm. Berilgan musbat *a* haqiqiy sondan kvadrat ildiz chiqarish algoritmi (iteratsion algoritmga doir blok-sxema).

MAVZU BO`YICHA TOPSHIRIQLAR

1. Algoritm blok-sxemasida ishlatiladigan figuralar va ularning ishlatilishini tavsiflang.
2. Chiziqli algoritm uchun blok-sxema tuzing:

a) A=$\frac{1+\sin^2(x-2y)}{x+y^2+\cos x}$+tg^{2}x B=cos(1+$\frac{Ax-y}{e^x+10^2}$-$\sqrt[3]{A}$);

b) A=ln(y-$\sqrt{x}+e^{x+y}$)+$\sqrt[3]{x-y}$ B=(x+tg$\frac{2\pi}{A}$)(5*10^{-6}+$\left|\frac{x-7}{A}\right|$);

c) To`g`ri burchakli uchburchakning katetlari berilgan. Uchburchak gipotenuzasi va yuzasini hisoblang.

3. Tarmoqlanuvchi algoritm uchun blok-sxema tuzing:

a) $\begin{cases} m^2n+1-c, & agar \quad n+1>0 \\ (m+n)^2+cm^2, & agar \quad n+1\le 0 \end{cases}$

b) $\begin{cases} \frac{a^2+b^2}{c}+\sqrt{a+x}, & agar \quad x\ge 0 \\ \frac{\sin x+a}{a-b}, & agar \quad x<0 \end{cases}$

4. Takrorlanuvchi algoritm uchun blok-sxema tuzing:

a) Quyida berilgan $f_1(x)$ va $f_2(x)$ funksiyalarning qiymatlari x ning berilgan $x\in[0,1]$ oralig`ida $h=0.1$ qadam bilan hisoblansin.

T/r	**Funksiya $f_1(x)$**	**Funksiya $f_2(x)$**
1	Sin (x+4.5)	20 / (1+x^2)
2	1+cos(x-2)	$\sqrt{x^2+1}$
3	e^x+Sin(x)	1+2^x
4	| Sin(x)+1 |	(1+2x)e^x
5	2-cos(x +1)	1-x^2
6	$\sqrt{2x}$ +x^2+1	e^x(1+cos$\frac{x}{2}$)

b) Quyida sodda takrorlanuvchi jarayonlar hisoblansin.

Masalalar

1. Natural son n berilgan. Hisoblansin:
   a) 2^n; b) n!;
2. Natural son n berilgan. Hisoblansin:
   (1+1/1^2) (1+1/2^2)…(1+1/n^2).
3. Natural son n berilgan. Hisoblansin:
   1/sin1+1/(sin1+sin2)+…+1/(sin1+…sin n).
4. Natural son n va haqiqiy son a berilgan. Hisoblansin:
   a (a+1)…(a+n-1).
5. Natural son n va haqiqiy son a berilgan. Hisoblansin:
   a (a-n) (a-2n)…(a-n^2).
6. Haqiqiy son x berilgan. Hisoblansin:
   x – x^3/3! + x^5/5! - x^7/7! + x^9/9! - x^{11}/11! + x^{13}/13!

3 -AMALIY MASHG`ULOT: Algoritmik hisoblash jarayoni. Holatlar, kiruvchi, joriy va chiquvchi berilganlar.

Ishning maqsadi: Algoritmik hisoblash jarayoni va undagi holatlar, berilganlar tiplari haqida ma'lumot berish. Aniq misollar asosida algoritmlar murakkabligi tushunchasini kiritish hamda turli masalalar sinfi uchun turli algoritmlar mavjudligini tushuntirish.

I. Nazariy qism

Informatsiyaga ishlov berish quyidagi sxema bo`yicha amalga oshiriladi:

BOSHLANG`ICH INFORMATSIYA ➔ IZLANAYOTGAN INFORMATSIYA

ya'ni, boshlang`ich informatsiyadan izlanayotgan informatsiya topiladi. Informatsiyaga ishlov berishni uni bir formadan boshqasiga tarjima qilish sifatida qarash mumkin. Masalan, $x + y = 5 \cap x - y = 7$ tenglamalar sistemasi aniqmas holda ildizlar haqida informatsiya beradi. Sistemani yechish bu informatsiyani aniq shaklga o`tkazishni anglatadi: $x = 6 \cap y = -1$.

Informatsiyaga ishlov berish jarayonida odatda qadamlarga bo`lib olishadi: $I_1$ ➔ $I_2$ ➔ … ➔ $I_{n-1}$ ➔ $I_n$ , bu yerda $I_1$ – boshlang`ich informatsiya, I_n – izlanilayotgan informatsiya, I_j, j = 2, …, n-1 – turli qadamlarda hosil qilinadigan oraliqlar. Qoida bo`yicha $I_1$ ➔ $I_n$ o`tishni bir-biridan nafaqat miqdori, balki tarkibi bilan farq qiluvchi turli qadamlar ketma-ketligi bilan hosil qilish mumkin. Informatsiyaga ishlov berish jarayonida qadamlar ketma-ketligini ifodalash uchun algoritmlardan foydalaniladi.

Shuning uchun algoritmda qo`llash mumkin bo`lgan so`zlar to`plamini *qo`llash sohasi* atamasi bilan ifodalaymiz. Qo`llanish sohasiga kirmaydigan kiruvchi so`zlarda algoritm yo to`xtaydi, yo uning harakatlari aniqlanmaydi (misoldagi kabi).

Berilganlar algoritmda o`zgaruvchilar sifatida (bizning misolda *a* va *b*), yoki algoritm bajarilish jarayonida o`z qiymatini o`zgartirmaydigan doimiy qiymatlar – konstantalar (bizning misolda *n*, *m*) tasvirlanishi mumkin.

*O`zgaruvchi* – mumkin bo`lgan qiymatlar to`plami bilan bog`liq bo`lgan atama. Algoritmning har bir aniq bajarilish vaqti oralig`ida har bir o`zgaruvchi to`plamga bog`liq bo`lgan aniq bir qiymatni qabul qiladi.

Ta'rif 1. *O`zgaruvchi turi* uning mumkin bo`lgan qiymatlar to`plamiga aytiladi. Kompyuter resurslari chegaralanganligi uchun bu oxirgi to`plam , ya'ni $\{v_i|i=1,k\}$.

Ta'rif 2. $\{p_i|i=1,k\}$ algoritmda foydalaniladigan o`zgaruvchilar to`plamini ko`rib chiqamiz. $\{p_i|i=1,k\}$ o`zgaruvchilar to`plamining *holati* deb bu o`zgaruvchilarning joriy to`plamiga aytiladi.

A – algoritm, $\{p_i|i=1,k\}$ – algoritmda foydalaniladigan barcha o`zgaruvchilar to`plami bo`lsin. Bu to`plamga π o`zgaruvchisini qo`shamiz, uning turi algoritmdagi amallar nomerlarining to`plamidir. Algoritmning har bir bajarilish qadamini $\{p_i|i=1,k\}$ U $\{\pi\}$ o`zgaruvchilar to`plamining holati sifatida ifodalash mumkin.

Ta'rif 3. Hisoblash jarayonida boshlang`ich berilganlar to`plami asosida vujudga kelgan algoritm qadamlar ketma-ketligi deb ataladi.

Ta'rif 4. Vujudga kelgan *A* algoritm *hisoblash jarayoning holati* deb, $\{p_i|i=1,k\}$ U $\{\pi\}$ o`zgaruvchilar to`plamining holatiga aytiladi, bu yerda $\{p_i|i=1,k\}$ – algoritmda ishlatilgan o`zgaruvchilar to`plami.

Hisoblash jarayonini uning holati ketma-ketligi ko`rinishida tasvirlash mumkinligin ko`rish qiyin emas. Bir holatdan boshqasiga o`tish algoritmnig biror amalini bajarishni ifodalaydi.

Ta'rif 5. Hisoblash jarayonining *terminal holati* deb, π *to`xtatish* amalining qiymatiga ega bo`lishiga aytiladi.

Ta'rif 6. *Natija* – bu algoritmni hisoblash jarayoni terminal holatidan aniqlangan qiymatlar to`plami.

Foydalanilgan berilganlar quyidagilarga bo`linadi:

1. *Kiruvchi* – kompyuterga kiradi va masalani yechish uchun shart sifatida foydalaniladi.

2. *Joriy yoki ichki* – dastur ichida informatsiyani saqlash va ishlov berish uchun ishlatiladi.

3. *Chiquvchi* – informatsiyaga ishlov berish natijasida dasturda hosil bo`lgan berilganlar. Matn, grafik, videotasvir va h. k. ko`rinishda bo`lishi mumkin.

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

Hodisa, jarayon, masala ➔ model ➔ algoritm ➔ dastur ➔ kompyuter ➔ natija.

II. Amaliy qism.

*Boshlang`ich berilganlar* – bu qiymatlar bilan algoritmning bajarilishi boshlanadi. Boshlang`ich berilganlar to`plami har doim aniqlangan. Boshlang`ich berilganlarni yozish usulini ko`rib chiqamiz. *Harf* – ixtiyoriy belgi. *Alfavit* – harflar to`plami $\{a_i \mid i=1, 2, \ldots, n\}$. *So`z* – biror alfavitdan ixtiyoriy harflar ketma-ketligi. Misol: A = {*a*, *b*} – alfavit; *a*, *ab*, *ba*, *abba* – shu alfavitdan so`zlar. So`z *uzunligi* deb so`zdagi harflar miqdoriga aytiladi. Nol uzunlikdagi so`z – bo`sh (ε bilan belgilanadi). Shunday qilib, algoritmga kirishda biror alfavitdan so`z (kirish so`zi) va natija ham so`z bo`ladi (chiquvchi so`z).

Ba’zan algoritmni ba’zi bir kiruvchi so`zlarga qo`llab bo`lmaydi.

Misol 1.

Algoritm:

1. *n ni 2 ga ko`paytiring;*
2. *ifodaga 1 ni qo`shing;*
3. *yig`indini 3 ga bo`ling;*
4. *n ni qoldiqqa bo`ling;*
5. *to`xtating.*

Hisoblash jarayonini boshlang`ich berilganlar uchun ko`rib chiqamiz.

Bosqich №	$n = 6$	$n = 7$
1.	12	14
2.	13	15
3.	4 (1 qoldiqda)	5 (0 qoldiqda)
4.	6	7:0?

Misol 2. Quyidagi masalalar sinfini hisoblash uchun algoritm:

$x=b$ nuqtada $a_n x^n + \ldots + a_1 x + a_0$ ifodaning qiymatini hisoblang, bu yerda $a_i \in R$, $b \in R$, R –moddiy sonlar to`plami. Bu masalalar sinfining xususiy holi sifatida, masalan,

x=1,5 (n=1, a_1=2, a_0=1, b=1,5) da 2x+1 ikki hadning qiymatini topish va x= –0,5 (n=3, a_3=1, a_2= –2, a_1=3, a_0=0, b= –0,5) da x^3-2x^2+3x ko`phadning qiymatini topishni keltirish mumkin.

Algoritm:

1. *i ni p ga, x ni b ga tenglashtiring;*
2. *r ni a_i ga tenglashtiring;*
3. *r ni r va x ifodaga tenglashtiring*
4. *i ni i–1 ga tenglashtiring*
5. *r ni r+a_i ga tenglashtiring*
6. *Agar i = 0, u holda to`xtating, aks holda 3-qadamga o`ting.*

Bu algoritmdagi hisoblashlarni quyidagicha ifodalash mumkin:

$$\left(...\left(\left(\left(a_n x + a_{n-1}\right)x + a_{n-2}\right)x + a_{n-3}\right)x+...+a_1\right)x + a_0$$

Nuqtadagi polinom qiymatini topishning bu usuli *Gorner sxemasi* deb ataladi. Biroq, bu masalaning yechimi uchun boshqa algoritmlar ham mavjud.

Boshlang`ich berilganlar: oldingi misoldagi kabi.

Natija: o`sha.

O`zgaruvchilar: $i \in Z$; $x, r \in R$.

Konstantalar: {a_i|i=0, n}, p, b.

Algoritm:

1. i ni p ga, r ni 0 ga, x ni b ga tenglashtiring*;*
2. x ni i darajaga oshiring*;*
3. a_i ni darajaga oshiring;
4. r ni r yig`indisiga teng ifodaga qo`ying;
5. Agar i = 0, u holda to`xtating, aks holda 6-qadamga o`ting;
6. i=i–1 ni qo`ying;
7. 2 qadamga o`ting.

Bu algoritm bo`yicha hisoblash quyidagi ifodani beradi: $a_n x^n+...+a_1 x + a_0$.

Talabalarga yuqorida boshlang`ich berilganlar misolida ko`rsatilgan algoritmlar asosida hisoblash jarayonini qurish taklif etiladi va ularning har xil ekanligiga ishonch hosil qilish mumkin.

Ko`rinib turibdiki, bir xil turdagi masalalar sinfini yechish uchun bir nechta turli algoritmlar mavjud. Ular asosida vujudga kelgan hisoblash jarayonlari amallar

to`plami va miqdori bilan farq qiladi. Hisoblash jarayonidagi amallar miqdori algoritmning muhim tomonlaridan biri hisoblanadi, chunki u algoritmni bajarish uchun kerak bo`lgan bajaruvchining vaqti va resurslarini aniqlaydi.

Ta'rif: *Algoritmning murakkabligi* deb, hisoblash jarayonida boshlang`ich berilganlar uchun berilganlar to`plami asosida vujudga kelgan algoritmdagi amallar miqdoriga aytiladi.

Ahamiyat bering, aynan hisoblash jarayonida, algoritmning o`zida emas. Turli algoritmlarning murakkabligini taqqoslash uchun u bir xil turdagi amallar atamasida hisoblanishi kerak. Masalan, ko`paytiring, qo`shing, tenglashtiring. (*i* – 1) ta ko`paytirish jarayoni kabi *i* ni darajaga oshirish amallari murakkabligini ifodalaymiz. U holda, 1-algoritmning murakkabligi (Gorner sxemasi bo`yicha) qo`shish amalining soniga va ko`paytirish *2p* ga teng bo`ladi. To`g`ri algoritm uchun u quyidagiga teng bo`ladi:

$$\sum_{i=0}^{n}(i+1)+n=\frac{(n+1)}{2}(n+2)+n=\frac{n^2+3n+2}{2}+n>\frac{n^2}{2}+2n>2n$$

Shunday qilib, bir xil boshlang`ich berilganlar uchun 1-algoritmning murakkabligi 2-algoritm murakkabligidan kichik, ya'ni birinchisi ikkinchiga ko`ra samaraliroq.

Xulosa: *Bir turdagi masalalar sinfini yechish uchun turli murakkablikdagi turli algoritmlar mavjud.*

MAVZU BO`YICHA TOPSHIRIQLAR

1. Algoritm - unga mos hisoblash jarayonidan nimasi bilan farqlanadi?
2. Evklid algoritmida n>m bo`lsin. Bu algoritm eng ko`p va eng kam qadamlar sonini amalga oshirishiga misollar keltiring.
3. Ko`rsatilgan so`zlarga qo`llab bo`lmaydigan algoritmlarga misol keltiring.
4. *Boshlang`ich berilganlar, kiruvchi so`z, chiquvchi so`z* atamalari nimani anglatadi?
5. Kiruvchi so`z algoritmning *qo`llanish sohasi*ga tegishli bo`lmasa nima ro`y beradi?
6. Boshlang`ich berilganlar to`plami asosida biror algoritmdan yaralgan *hisoblash jarayoniga* tushuncha bering va misol keltiring.
7. Biror algoritmdan yaralgan *hisoblash jarayoni holati* nimani anglatadi?

8. Hisoblash jarayoning *terminal holati* deganda nima tushuniladi?
9. Algoritmni hisoblash jarayoni *natijasi* deb nimaga aytiladi?
10. *Algoritm murakkabligi* deb nimaga aytiladi?
11. Misollarda quyidagi tasdiqni tushuntiring: “*Yagona masalalar sinfi yechimi uchun turli murakkablikdagi turli algoritmlar mavjud* ”.

4 -AMALIY MASHG`ULOT: Tyuring mashinasi ustida amallar bajarish. Tyuring mashinasi va EHM.

Ishning maqsadi: Algoritm tushunchasini formallashtirish modellaridan biri bo`lgan Tyuring mashinasi (TM) elementlari, uning ishlash prinsipini tushuntirish hamda u uchun dasturlar tuzish ko`nikma va malakasini shakllantirish.

I. Nazariy qism

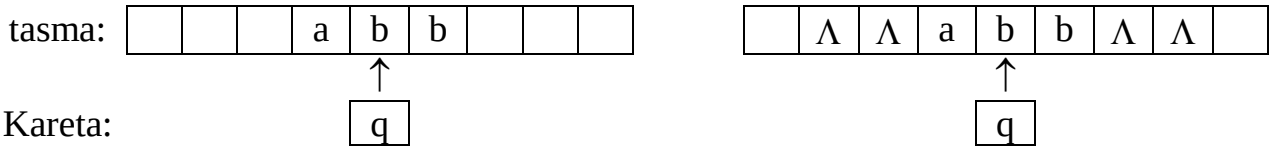
Tyuring mashinasining real hisoblash mashinalaridan prinsipial farqi shundaki, uning xotira qurilmasi qanchalik katta bo`lmasin, baribir u cheklidir. TMni uning cheksiz xotirasi tufayli fizik reallashtirishning iloji yo`q. Bu ma’noda TM har qanday hisoblash mashinasidan qudratliroqdir.

1.1. Tyuring mashinasining tuzilishi.

Tyuring mashinasi ikki qismdan iborat:

● ikki tomonlama cheksiz tasma (lenta) va u katakchalar (yacheykalar)ga bo`lingan;

● avtomat, katakchadan katakchaga qo`zg`aluvchi kareta, tasmaga yozilgan belgi (simvol) larni o`qiydi, o`chiradi va yozadi:



Tasma axborotni saqlash uchun xizmat qiladi. Tasmadagi katakchalar nomerlanmaydi va nomlanmaydi. Bo`sh katakchalar Λ belgi yordamida tasvirlanadi. Shuning uchun yuqorida keltirilgan rasmda tasvirlanganlar teng kuchlidir.

Kareta – bu Tyuring mashinasining faol qismi hisoblanadi. Har bir onda u biror katakcha ostida joylashgan bo`ladi va u katakchadagi belgini o`qiydi. Kareta

turgan katakcha *joriy katakcha*, undagi belgi esa *joriy belgi* deyiladi. Qo`shni va boshqa katakchalardagi belgilarni kareta ko`rmaydi. Shuningdek, har bir momentda kareta biror q holatda bo`ladi va uni $q_1, q_2, ...$ nomerlash mumkin. Qandaydir holatda bo`lgan kareta ma'lum bir amalni bajaradi. Masalan, tasma bo`yicha o`ngga suriladi va b belgini a ga almashtiradi.

Joriy belgi (S) va holat (q) juftligini konfiguratsiya deb ataymiz hamda $\langle S, q \rangle$ ko`rinishida belgilaymiz.

Kareta quyidagi 3 ta elementar amalni bajaradi:

1) Joriy katakchaga yangi belgini yozishi mumkin (boshqa katakchadagi belgini o`zgartira olmaydi);
2) Bir katakcha chapga yoki o`ngga surilishi, yoki o`z joyida qolishi mumkin (bir necha katakchaga "sakrash" ni bajara olmaydi);
3) Yangi holatga o`tishi mumkin.

Kareta boshqa amallarni bajara olmaydi, shuning uchun boshqa murakkab jarayonlarda mana shu uchta amallarga keltiriladi.

1.2. Tyuring mashinasi takti.

TM diskret rejimda «qadam-baqadam» ishlaydi: u vaqt momenti oralig`ida faqat bitta buyruqni bajaradi. TMning har bir qadamda bajargan ishi ***takt*** deyiladi. Har bir taktda quyidagi ketma-ketlikdagi amallarni bajaradi:

1) Joriy katakchaga biror S' belgini yozadi (xususiy holda shu belgining o`zini yozishi ham mumkin, u holda bu katakchadagi belgi o`zgarmaydi);
2) Bir katakcha chapga (L - bilan belgilaymiz, inglizcha *left* so`zidan olingan) yoki o`ngga (R - *right*) suriladi, yoki o`z joyida (N - bilan belgilaymiz) qoladi;
3) Qandaydir q' holatga o`tadi (ba'zida oldingi holatda qoladi).

Formal tarzda Tyuring mashinasining bir taktini quyidagi uchlik shaklida yozamiz:

$$S', [L, R, N], q'$$

bunda kvadrat qavs ichida yozuv, L, R, N harflaridan biri bo`lishini bildiradi. Masalan, $*, L, q_8$ yozuvi – joriy katakchaga * belgisini yozish, bir katakcha chapga siljish va $q_8$ holatga o`tishni bildiradi.

1.3. Tyuring mashinasi uchun dastur.

Tyuring mashinasini ishlatish uchun unga ***dastur*** tuzish lozim. Bu dastur quyidagi jadval shaklida yoziladi:

	S_1	S_2	…	S_i	…	S_n	Λ
q_1							
…							
q_i				$S', [L, R, N], q'$			
…							
q_m							

Jadval katakchalarida kareta bajarishi lozim bo`lgan amallar takti yoziladi va shu buyruqlar asosida TM ishlaydi. Demak, Tyuring mashinasini tasma, kareta va dastur to`liq aniqlaydi.

Tuzilgan dasturning bajarilishigacha quyidagi ishlarni amalga oshirish talab qilinadi:

- Birinchidan, tasmaga ***kirish so`zi*** ni yozish kerak, chunki dastur unga nisbatan bajariladi. Kirish so`zi – bu belgilarning chekli ketma-ketligi bo`lib, tasma katakchalariga ketma-ket joylashtiriladi. Kirish so`zining ichida bo`sh katakchalar bo`lmasligi kerak, uning chap va o`ng tomonlarida bo`sh katakchalar bo`ladi, chunki tasma ikki tomonlama cheksiz davom etgan.
- Ikkinchidan, karetani q_1 holatda va kirish so`zining birinchi belgisi ostiga joylashtirish lozim.

Shundan keyin dastur bajarilishi boshlanadi. Dastur jadvalining kirish so`zi birinchi belgisi va $q_1$ holat kesishmasidagi taktdan dastur o`z ishini boshlaydi. Shundan keyin, kareta yangi konfiguratsiyaga o`tadi va navbatdagi takt bajariladi va h.k.

Savol tug`iladi: “Dastur qachon ishini yakunlaydi?”. Buning uchun biz ***to`xtash takti*** tushunchasini kiritamiz. Bu taktda joriy katakchadagi belgi o`zgarmaydi va kareta siljimaydi, ya’ni $\langle S, q \rangle$ konfiguratsiya uchun $S, N, q$ takt bajariladi. Shu tarzda TM o`z ishini tugatadi.

Tyuring mashinasi ishida ikki holat yuzaga kelishi mumkin:

1) *“Yaxshi”* holat, ya’ni TM chekli taktlarni bajargandan keyin o`z ishini to`xtatadi. Bunday holda TM berilgan kirish so`zi uchun ***qo`llaniluvchi*** deyiladi va tasmada hosil bo`lgan yangi so`z chiqish so`zi, ya’ni natija yoki masalaning javobi bo`ladi.

   To`xtash vaqtida quyidagi muhim shartlar bajarilishi kerak:

   - chiqish so`zining ichida bo`sh katakcha bo`lmasligi lozim (eslatib o`tamiz, dastur bajarilishi jarayonida qayta ishlanayotgan so`z ichida bo`sh kataklar bo`lishi mumkin, lekin, oxirgi natija so`zda bo`lmasligi kerak);
   - kareta chiqish so`zining ixtiyoriy belgisi ostida to`xtashi mumkin (aynan qaysisi ekanligi muhim emas), agar so`z bo`sh bo`lsa, ixtiyoriy katakchada.

2) *“Yomon”* holat, ya’ni TM to`xtash taktiga tushmasdan, siklga tushib qoladi. Masalan, kareta har qadamda o`ngga siljiydi va to`xtamaydi, chunki tasma cheksiz. Bunday holda TM kirish so`zi uchun ***qo`llaniluvchi emas*** deyiladi va natija bermaydi.

Ta’kidlash mumkinki, TM uchun tuzilgan dastur bir vaqtning o`zida biror kirish so`zi uchun qo`llaniluvchi bo`lsa, boshqasi uchun qo`llaniluvchi bo`lmasligi mumkin (siklga tushib qoladi). Demak, *qo`llaniluvchi* yoki *qo`llaniluvchi emaslik* sharti faqat algoritmga emas, balki kirish so`ziga ham bog`liq ekan.

TM uchun dastur yozish qulay bo`lishi uchun quyidagi belgilashlarni kiritamiz:

a) Agar taktda joriy belgi o`zgarmasa, yoki kareta siljimasa, yoki holat o`zgarmasa, u holda, ularning o`rni ochiq qoldirib yoziladi. Masalan, $\langle a, q_1 \rangle$ konfiguratsiya uchun quyidagi yozuvlar ekvivalantdir:

$a, R, q_3 \Leftrightarrow \ , R, q_3$
$b, N, q_2 \Leftrightarrow b, \ , q_2$
$a, L, q_1 \Leftrightarrow \ , L,$
$a, N, q_1 \Leftrightarrow \ , \ ,$ (bu to`xtash takti)

b) TM o`z ishini biror taktdan keyin to`xtatishini ko`rsatmoqchi bo`lsak, u holda taktning uchinchi pozitsiyasiga "!" belgisini yozamiz. Masalan, $b, L,!$ takti – joriy katakchaga b ni yozish, chapga siljish va ishni to`xtatish kerakligini bildiradi.

II. Amaliy qism.

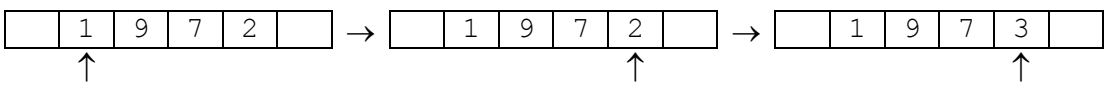
Tyuring mashinasi uchun dastur tuzish maqsadida bir necha tipik masalalar ko`rib chiqamiz. Masala qo`yilishini soddalashtirish uchun quyidagi ikki belgilashni kiritamiz:

- kirish so`zini *P* orqali belgilaymiz;
- *A* bilan kirish so`zining alfavitini belgilaymiz, undagi belgilar faqat va faqat *P* so`zida ishlatiladigan belgilardir (shuni ta'kidlash kerakki, ichki va chiquvchi so`zlarda boshqa belgilar ham paydo bo`lishi mumkin).

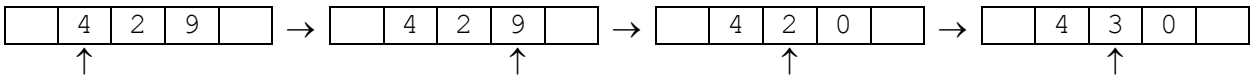
1-masala. A={0,1,2,3,4,5,6,7,8,9}. *P* – bo`sh bo`lmagan so`z bo`lsin, ya'ni u o`nlik raqamlar ketma-ketligidan tashkil topgan so`z bo`lib, o`nlik sanoq sistemasidagi nomanfiy va butun sonni ifodalaydi. TM uchun *P* sonidan 1 birlikka katta bo`lgan sonni topish dasturi tuzilsin.

Yechish. Bu masalani yechish uchun quyidagi amallarni bajarish lozim:

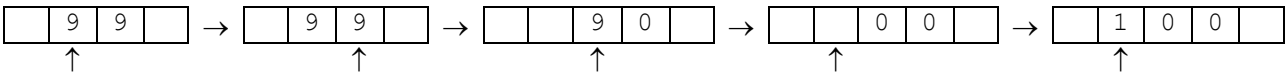
1. Karetani oxirgi raqamga olib boriladi.
2. Agar bu raqam 0 dan 8 gacha bo`lsa, u holda uni 1 birlikka oshiriladi va to`xtaydi, masalan:



3. Agar bu raqam 9 bo`lsa, u holda uni 0 ga almashtirib, karetani bir katakcha oldinga surib, keyin o`sha raqam yana 1 ga orttiriladi, masalan:



4. Maxsus holat: agar *P* soni faqat 9 raqamidan tashkil topgan bo`lsa (masalan, 99), u holda kareta 9 ni 0 ga almashtirgan holda chapga surilaveradi va oqibatda bo`sh katakka kelib qoladi. Ushbu bo`sh katakka 1 ni yozadi va to`xtaydi (javob 100 bo`ladi):



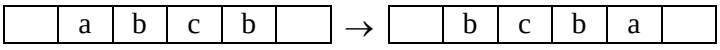
Yuqorida keltirilgan amallarga tayangan holda TM uchun dastur quyidagi ko`rinishda ifodalanadi:

	0	1	2	3	4	5	6	7	8	9	Λ
q1	0,R,q1	1,R,q1	2,R,q1	3,R,q1	4,R,q1	5,R,q1	6,R,q1	7,R,q1	8,R,q1	9,R,q1	Λ,L,q2
q2	1,N,!	2,N,!	3,N,!	4,N,!	5,N,!	6,N,!	7,N,!	8,N,!	9,N,!	0,L,q2	1,N,!

q1 – bu karetani oxirgi belgiga *"quvib"* boruvchi holat. Buning uchun kareta joriy belgilarni o`z o`rnida qoldirib, doimo o`ngga harakatlanadi va shu *q1* holatda qoladi. *q2* – holat esa, kareta ko`rib turgan raqamni bir birlikka orttiradi.

2-masala. *A*={*a,b,c*} alfavit berilgan bo`lsin. TM uchun bo`sh bo`lmagan *P* so`zning birinchi belgisini oxiriga o`tkazish dasturi tuzilsin.

Masalan:



Yechish. Bu masalani yechish uchun quyidagi amallarni bajarish lozim:

1. *P* kirish so`zining birinchi belgisi eslab qolinadi, so`ngra bu belgi o`chiriladi.
2. Karetani *P* so`zning oxiridagi bo`sh katakka olib boriladi va eslab qolingan belgi o`sha joyga yoziladi.

Karetani o`ng tomonga quvib borishni oldingi misoldan bilamiz, lekin, birinchi belgini qanday eslab qolamiz? Axir TMda xotira qurilmasi bo`lmasa, shuningdek, kareta chapga yoki o`ngga surilganda oldingi katakdagi belgini shu ondayoq esdan chiqaradi. Nima qilish kerak?

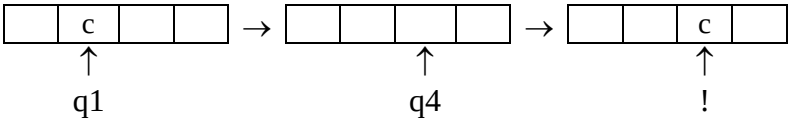
Bu vaziyatdan chiqish uchun karetaning turli *qi* holatlaridan foydalanish kerak. Agar birinchi belgi *a* bo`lsa, u holda *q2* holatga o`tish lozim, shunda kareta o`ngga surilib boradi va oxiriga *a* belgini yozadi. Agar birinchi belgi *b* bo`lsa, *q3* holatga; birinchi belgi *c* bo`lsa, *q4* holatga o`tish talab qilinadi. Shunday qilib, birinchi belgi qanday bo`lishidan qat'iy nazar, unga bitta holat mos qo`yiladi. Bu esa TM uchun dastur tuzishning bir usuli hisoblanadi.

Yuqoridagilarga asosan TM uchun masalaning dasturini tuzamiz:

	a	b	c	Λ	
q1	Λ,R,q2	Λ,R,q3	Λ,R,q4	,R,	1-belgini o`chirish, tarmoqlanish
q2	,R,	,R,	,R,	a, ,!	o`ng tomonga *a* belgini yozish
q3	,R,	,R,	,R,	b, ,!	o`ng tomonga *b* belgini yozish
q4	,R,	,R,	,R,	c, ,!	o`ng tomonga *c* belgini yozish

Endi bu dasturning kirish so`zi bir belgidan oshmagan hollar uchun qanday ishlashini tekshiramiz:

- Agar kirish so`zi "*bo`sh*" bo`lsa, TM siklga tushib qoladi va masala uchun "*yomon*" holat qayd etiladi, ya'ni *q1* holat saqlanib qolib, kareta o`ngga cheksiz harakatni davom ettiradi. Albatta, bu yerda TM ishini to`xtatish mumkin, lekin biz, bunday jarayonni namoyish qilish uchun shu yo`lni tanladik.
- Agar kirish so`zida bittagina belgi bo`lsa, u holda kareta bu belgini o`chiradi, o`ngga siljiydi va berilgan belgini shu katakka yozadi:



Shunday qilib, kirish so`zi bitta belgidan iborat bo`lsa, bir katak o`ngga siljiydi, xalos. Bu mumkin, chunki tasma nomerlanmagan va ikki tomonlama cheksiz bo`lganligi uchun so`zning o`ngga yoki chapga siljishi ahamiyatsiz.

MAVZU BO`YICHA TOPSHIRIQLAR

1. Tyuring mashinasi tuzilishi, vazifasi, amaliy ahamiyati va ishlash prinsipini tushuntiring.

2. Quyidagi masalalarda TM uchun dastur tuzilsin:

a) $A=\{a,b,c\}$ alfavit berilgan. Agar bo`sh bo`lmagan P so`zning birinchi va oxirgi belgilari bir xil bo`lsa, u holda so`z almashtirilmasin, aks holda u bo`sh so`zga almashtirilsin.

b) $A=\{a,b\}$ alfavitdan tuzilgan P so`zning ikkinchi belgisi o`chirilsin, agar u mavjud bo`lsa.

c) $A=\{a,b,c\}$. P so`zning tarkibidagi birinchi uchragan $a$ belgi o`chirilsin, agar shunday mavjud bo`lsa.

d) $A=\{a,b,c\}$. Agar P bo`sh bo`lmagan so`z bo`lsa, u holda uning birinchi belgisidan keyin a belgi joylashtirilsin. (Masalan, bcba $\rightarrow$ bacba).

5 - AMALIY MASHG`ULOT: Markovning normal algoritmlari.

Ishning maqsadi: Algoritm tushunchasini formallashtirish modellaridan biri bo`lgan Markovning normal algoritmlari haqida tushuncha berish, u yordamida masalalarni yechish ko`nikma va malakasini shakllantirish.

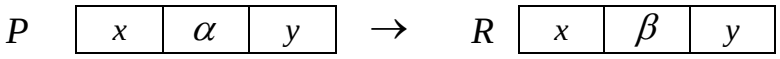
I. Nazariy qism

Markov normal algoritmlarining (MNA) maxsusligi shundaki, ularda faqat bitta elementar amal – o`rin almashtirish bajariladi.

O`rin almashtirish formulasi deb $\alpha \rightarrow \beta$ (“$\alpha$ ni $\beta$ ga almashtirish” deb o`qiladi) ko`rinishidagi yozuvga aytiladi. Bu yerda $\alpha$ va $\beta$ - ixtiyoriy so`zlar (bo`sh bo`lishi ham mumkin). Bunda α formulaning chap, β esa o`ng qismi deyiladi.

O`rin almashtirishning o`zi (amal sifatida) o`rin almashtirish formulasi bilan beriladi va qandaydir $P$ so`zga qo`llaniladi. Bu amalning ma’nosi shundaki, $P$ so`zda formulaning chap tomoni bilan mos keladigan qismi (ya’ni α) topiladi va u formulaning o`ng qismi (ya’ni $\beta$) bilan almashtiriladi. Bunda $P$ so`zning qolgan qismlari (α dan chap va o`ngdagi) o`zgarmaydi. Hosil bo`lgan $R$ so`zni **o`rin almashtirish natijasi** deyiladi. Shartli ravishda uni quyidagicha tasvirlash

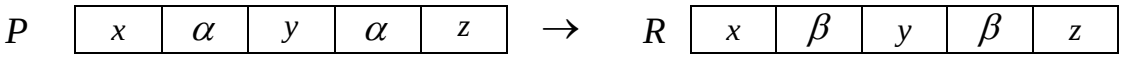
mumkin:



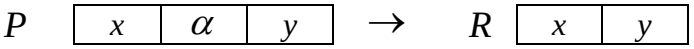
Kerakli aniqlashtirishlar:

1. Agar o`rin almashtirish formulasining chap qismi $P$ so`zning tarkibiga kirsa, u holda bu formulani P ga qo`llash mumkin deyiladi. Agar $\alpha$ $P$ so`zning tarkibiga kirmasa, u holda formulani P ga qo`llash mumkin emas deyiladi va o`rin almashtirish bajarilmaydi.

2. Agar α P so`zning tarkibiga bir necha bor kirsa, u holda ta'rifga ko`ra β ga faqat α ni P so`zga birinchi kirgani almashtiriladi:

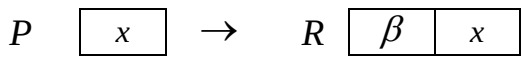


3. Agar o`rin almashtirish formulasining o`ng tarafi bo`sh so`z bo`lsa, u holda $\alpha\rightarrow$ o`rin almashtirish, P dan α ni o`chirishga olib kelinadi (shuni aytib o`tish kerakki, o`rin almashtirish formulalarida bo`sh so`zni alohida belgilash yo`q):



4. Agar o`rin almashtirish formulasining chap tarafida bo`sh so`z bo`lsa, u holda $\rightarrow\beta$ o`rin almashtirish, ta'rifga ko`ra, P so`zdan chap tarafga β ni yozishga keladi:

Bu qoidadan juda muhim natija kelib chiqadi: chap tarafi bo`sh bo`lgan formula ixtiyoriy so`z uchun o`rinli. Yana shuni ta'kidlab o`tamizki, chap va o`ng taraflari bo`sh bo`lgan formula so`zni o`zgartirmaydi.



Ta'rif: Markovning normal algoritmi (NAM) deb, bo`sh bo`lmagan tartiblangan o`rin almashtirish formulalari to`plamiga aytiladi:

$$\begin{cases} \alpha_1 \to \beta_1 \\ \alpha_2 \to \beta_2 \\ \dots \\ \alpha_k \to \beta_k \end{cases} \qquad (k \geq 1)$$

Bu formulalarda ikki turdagi strelkalar ishlatilishi mumkin: oddiy strelka ($\to$) va “dumli” strelka ($\mapsto$). Oddiy strelkali formula ***oddiy formula***, “dumli” strelkali formula esa – ***yakunlovchi formula*** deyiladi. Ular o`rtasidagi farq keyinroq tushuntiriladi.

MNA ni bajarish qoidasi:

Avvalo kiruvchi P so`z beriladi. U qayerda yozilgani muhim emas, MNA da bu savol oldindan belgilanmaydi.

MNA ning ishi qadamlar ketma-ketligini bajarishga keltiriladi. Har bir qadamda MNA ga kiruvchi o`rin almashtirish formulalari yuqoridan pastga qarab ko`rib chiqiladi va kiruvchi P so`zga qo`llaniladigan formulalardan birinchisi, ya’ni chap tarafi P so`zga kiruvchi eng yuqorida turgani tanlanadi. So`ng tanlangan formulaga ko`ra o`rin almashtiriladi. Yangi P' so`z hosil bo`ladi.

Keyingi qadamda bu P' so`z dastlabki deb olinadi va unga xuddi shu protsedura qo`llanadi, ya’ni formulalar yana yuqoridan pastga qarab ko`rib chiqiladi va $P'$ so`zga qo`llaniladigan birinchi formula qidiriladi, undan keyin mos o`rin almashtirish bajariladi va yangi P'' so`z olinadi. Va hokazo:

$P \to P' \to P'' \to \dots$

Shunga alohida e’tibor berish kerakki, MNA da har bir qadamda formulalar eng birinchisidan boshlab ko`rib chiqiladi.

Kerakli aniqlashtirishlar:

1. Agar navbatdagi qadamda oddiy formula ($\alpha \to \beta$) qo`llangan bo`lsa, u holda MNA ishni davom ettiradi.

2. Agar navbatdagi qadamda yakunlovchi formula ($\alpha \mapsto \beta$) qo`llangan bo`lsa, undan keyin MNA ishni tugatadi. Shu onda hosil bo`lgan so`z ***chiquvchi so`z*** bo`ladi, ya’ni MNA ni kiruvchi so`zga qo`llash natijasi.

Oddiy va yakunlovchi formulalar orasidagi farq shundaki, oddiy formulani

qo`llaganda MNA ishni davom ettiradi, yakunlovchi formuladan keyin esa – tugatadi.

3. Agar navbatdagi qadamda so`zga hech bir formulani qo`llab bo`lmasa, u holda MNA ishni tugatadi va chiquvchi so`z hozirgi so`z bo`ladi.

Shunday qilib, MNA ikki sababga ko`ra to`xtaydi: yoki yakunlovchi formula qo`llanilgan yoki hech bir formula to`g`ri kelmaydi. U ham, bu ham MNA ishining "yaxshi" nihoyasi hisoblanadi. Ikkala holda ham MNA ni kiruvchi so`zga qo`llash mumkin deyiladi.

Ammo MNA hech qachon to`xtamasligi ham mumkin; bu har bir qadamda qo`llash mumkin bo`lgan formula mavjud va bu formula oddiy bo`lganda yuz beradi. Unda MNA ni kiruvchi so`zga qo`llab bo`lmaydi deyiladi. Bu holda hech qanday natija haqida gapirib bo`lmaydi.

II. Amaliy qism.

MNA qo`llaniladigan oddiy masalalarni ko`rib chiqamiz.

Tyuring mashinasi holidagi kabi masalaning qo`yilishini qisqartirish uchun quyidagi belgilashlardan foydalanamiz:

- P harfi bilan kiruvchi so`zni;

- A harfi bilan kiruvchi so`z alifbosini belgilaymiz, ya'ni $P$ ga va faqat $P$ ga kiruvchi belgilar to`plami (lekin MNA ni bajarilish jarayonida boshqa belgilar hosil bo`lishi mumkin).

Bundan tashqari, masalalarda o`rin almashtirish formulalaridan o`ng tarafda ularning tartib raqamlarini yozamiz. Bu raqamlar formulalar tarkibiga kirmaydi, ular faqat MNA qadamlarini bajarishda kerak.

1-masala (belgini qo`yish va o`chirish). $A=\{a,b,c,d\}$. P so`zda uni tarkibiga kiruvchi birinchi $bb$ so`zni ddd ga almashtirish va P ning tarkibidagi barcha c belgilarni o`chirish kerak.

Masalan: $abbcabbca \to adddabba$

Yechish. MNA da Tyuring mashinasidan farqli ravishda belgilarni qo`yish va o`chirish oson amalga oshiriladi. So`zga yangi belgilarni qo`yish – bu qandaydir so`z qismini ko`proq belgidan iborat so`z qismiga almashtishdir; masalan, $bb \to ddd$ formula yordamida ikkita belgi uchta belgiga almashtiriladi. Bunda avvaldan qo`shimcha belgilar uchun joy ajratishni o`ylash kerak emas, MNA da so`z avtomatik tarzda suriladi. Belgilarni o`chirish esa – qandaydir so`z qismini kamroq belgidan iborat so`z qismiga almashtirishdir; masalan, $c$ belgini o`chirish $c \to$ formulasi bilan amalga oshiriladi. Bunda so`z tarkibida hech qanday bo`sh joylar hosil bo`lmaydi, MNA da so`zni siqish avtomatik bajariladi.

Aytilganlarni hisobga olsak, bizning masalani ushbu MNA yechishi kerak:

$$\begin{cases} bb \to ddd & (1) \\ c \to & (2) \end{cases}$$

Ammo bu unday emas. Bu MNA ni kiruvchi *abbcabbca* so`zda tekshirib ko`raylik (strelkalardan yuqorida qo`llaniladigan formulalar raqamlari yozilgan, so`zlarda strelkalardan chap tarafda esa, ko`rinarli bo`lishi uchun formulalar qo`llanilayotgan qismlar ostiga chizib qo`yilgan):

$$a\underline{bb}cabbca \overset{1}{\to} adddca\underline{bb}ca \overset{1}{\to} addd\underline{c}adddca \overset{2}{\to} adddabb\underline{c}a \to \ldots$$

Birinchi *bb* ni *ddd* ga almashtirib, bu MNA *c* belgini o`chirishga o`tmadi, balki boshqa *bb* larni ham almashtirdi. Nega? Eslatib o`tamizki, MNA ning har bir qadamida o`rin almashtirish formulalari doimo birinchi formuladan boshlab yuqoridan pastga qarab ko`rib chiqiladi. Shuning uchun birinchi formula qo`llanilar ekan u boshqa formulalarning bajarilishiga yo`l qo`ymaydi. Bu MNA da o`rin almashtirish formulalar tartibi muhimligini ko`rsatadi.

Shuni hisobga olib, formulalarning o`rnini almashtiramiz:

$$\begin{cases} c \to & (1) \\ bb \to ddd & (2) \end{cases}$$

Bu yangi algoritmni yana o`sha so`zda tekshirib ko`ramiz:

$$abb\underline{c}abbca \overset{1}{\to} abbabb\underline{c}a \overset{1}{\to} a\underline{bb}abba \overset{2}{\to} adddab\underline{ba} \overset{2}{\to} adddaddda$$

Shunday qilib, MNA avval barcha *c* belgilarni o`chirdi va undan so`ng

birinchi bb ni ddd ga almashtirdi. Ammo MNA shu bilan to`xtamadi va qolgan $bb$ larni ham almashtirdi. Nega? Gap shundaki, MNA formulalar ishlatib bo`lmagunga qadar o`z ishini davom ettiradi. Lekin bizga bu kerak emas, shuning uchun biz MNA ni birinchi $bb$ ni almashtirgandan keyin to`xtatishimiz kerak. Shuning uchun ham yakunlovchi o`rin almashtirish formulalari kerak bo`ladi. Shunday ekan, bizning algoritmda $bb \to ddd$ oddiy formulani $bb \mapsto ddd$ yakunlovchi formulaga almashtirish kerak:

$$\begin{cases} c \to & (1) \\ bb \mapsto ddd & (2) \end{cases}$$

Ana endi bizning algoritm to`g`ri ishlaydi:

$$abb\underline{c}abbca \overset{1}{\to} abbabb\underline{c}a \overset{1}{\to} a\underline{bb}abba \overset{2}{\mapsto} adddabba$$

(2) yakunlovchi formulani qo`llagandan so`ng hosil bo`lgan so`z, chiquvchi so`z bo`ladi, ya'ni berilgan so`zga MNA ni qo`llash natijasi.

Biz MNA ni, tarkibiga bb kirmaydigan so`zda ham tekshirib ko`ramiz:

$$d\underline{c}acb \overset{1}{\to} da\underline{c}b \overset{1}{\to} dab$$

Oxirgi so`zga ($dab$) hech qaysi formulani qo`llab bo`lmaydi, shuning uchun MNA ning ta'rifiga ko`ra algoritm to`xtaydi va bu so`z chiquvchi deb e'lon qilinadi.

2-masala (belgilarni o`rnini almashtirish). $A=\{a,b\}$. $P$ so`zni shunday o`zgartiringki, natijada uning boshida barcha a belgilar, oxirida esa - b belgilar yozilsin.

Masalan: $babba \to aabbb$

Yechish. Bu masalani yechish uchun murakkab MNA kerakdek tuyuladi. Ammo bu unday emas, masala faqat bitta formuladan iborat MNA yordamida yechiladi:
$\{ba \to ab$

P so`zda $b$ belgidan o`ng tarafda hech bo`lmaganda bitta $a$ belgi bo`lsa ham bu formula a ni b ning chap tarafiga o`tkazadi. Formula $b$ dan o`ng tarafda a

qolmaguncha ishlaydi va bu barcha a belgilar b dan o`ngda joylashganini anglatadi. Masalan:

$$\underline{ba}abba \to abb\underline{ba} \to ab\underline{ba}b \to a\underline{ba}bb \to aabbb$$

Algoritm oxirgi so`zda to`xtaydi, negaki, unga bizning formulani qo`llab bo`lmaydi.

Bu va oldingi masalalar, MNA Tyuring mashinasidan farqli ravishda belgilarni o`rin amlamtirish, o`rniga qo`yish va o`chirishni oson bajarishi bilan farqlanishini ko`rsatadi. Ammo MNA da boshqa muammo paydo bo`ladi: qanday qilib ishlanishi kerak bo`lgan belgi (so`z qismini) ni fiksirlash mumkin? Bu muammoni keyingi masalada ko`rib chiqamiz.

3-masala (maxsus belgining ishlatilishi). $A=\{a,b\}$. Bo`sh bo`lmagan P so`zdan uning birinchi belgisini o`chiring. Bo`sh so`zni o`zgartirmang.

Yechish. So`zning birinchi belgisini o`chirib shu zahoti to`xtash kerak. Shuning uchun masalani quyidagi MNA yechadiganga o`xshaydi:

$$\begin{cases} a\mapsto & (1) \\ b\mapsto & (2) \end{cases}$$

Ammo bu noto`g`ri algoritm, bunga uni $bbaba$ so`zga qo`llab ishonch hosil qilish mumkin:

$$bb\underline{a}ba \overset{1}{\mapsto} bbba$$

Ko`rinib turibdiki, bu MNA so`zning birinchi belgisini emas, balki uning tarkibiga kiruchvi birinchi a ni o`chirdi, bu esa turli narsa. Agar kiruvchi so`z a belgi bilan boshlansa, bu algoritm to`g`ri ishlaydi. Bu MNA da o`rin almashtirish formulalari yordam bermaydi, negaki, u bu holda $a$ bilan boshlanuvchi so`zlarda noto`g`ri ishlay boshlaydi.

Nima qilish kerak? So`zning birinchi belgisini qandaydir belgilash, fiksirlash kerak, masalan, uning oldiga so`z alifbosiga kirmaydigan qandaydir belgi, misol uchun *, qo`yish kerak. Shundan so`ng $*\zeta\mapsto$ ko`rinishidagi formulalar yordamida

bu belgi va birinchi ζ belgini bo`sh o`ringa almashtirish va to`xtash mumkin:

$$bbaba \to *bbaba \mapsto baba$$

* belgisini birinchi belgidan oldinga qanday qo`yish mumkin? Bu chap tarafi bo`sh bo`lgan $\to *$ formulasi bilan amalga oshiriladi, u ta'rifga ko`ra o`zining o`ng qismini so`zdan chap tarafga qo`yadi.

Natijada quyidagi MNA ni hosil qilamiz:

$$\begin{cases} \to * & (1) \\ *a \mapsto & (2) \\ *b \mapsto & (3) \end{cases}$$

Uni avvalgi kiruvchi so`zda tekshirib ko`ramiz:

$$bbaba \overset{1}{\to} *bbaba \overset{1}{\to} **bbaba \overset{1}{\to} ***bbaba \overset{1}{\to} \dots$$

Ko`rinib turibdiki, bu algoritm chap tarafga yulduzchalar qo`shib yozmoqda. Nega? Chap tarafi bo`sh bo`lgan formula har doim o`rinli, shuning uchun (1) formula boshqa formulalarni ishlatmay cheksiz ishlaydi. Bundan juda muhim qoida kelib chiqadi: agar MNA da chap tarafi bo`sh formula ($\to \beta$) mavjud bo`lsa, u holda uning o`rni MNA ning oxirida bo`ladi. Bu qoidani hisobga olib MNA ni qaytadan yozamiz:

$$\begin{cases} *a \mapsto & (1) \\ *b \mapsto & (2) \\ \to * & (3) \end{cases}$$

Bu algoritmni tekshiramiz:

$$bbaba \overset{3}{\to} *\underline{b}baba \overset{2}{\mapsto} baba$$

Xuddi hammasi joyidaga o`xshaydi. Ammo bu unday emas: bizning algoritm bo`sh kiruvchi so`zda aylanib qoladi, negaki, har doim (3) formula qo`llanadi, shartga ko`ra esa bunday so`zda MNA to`xtashi kerak. Bu xatoning sababi nimada? Gap shundaki, biz * belgisini so`zning birinchi belgisini belgilash, so`ng * va shu belgini o`chirish uchun kiritdik. Lekin bo`sh so`zda bitta ham belgi yo`q, shuning uchun (1) va (2) formulalar ishlamaydi va har doim (3) formula bajariladi. Shunday ekan, bo`sh so`z holini hisobga olish uchun, (1) va (2) formulalardan

so`ng yana bir formulani yozish kerak, bu formula “yakka” yulduzcha belgisini o`chirib tashlaydi va algoritmni to`xtatadi:

$$\begin{cases} *a \mapsto & (1) \\ *b \mapsto & (2) \\ * \mapsto & (3) \\ \to * & (4) \end{cases}$$

Nihoyat, biz to`g`ri algoritm tuzdik.

Keyingi masalalarga o`tishdan oldin, 3-masalada ishlatgan yulduzcha usulini umumlashtiramiz.

Faraz qilaylik, qaralayotgan P so`zning tarkibiga $\alpha$ so`zning qismi kirsin:

P	...	α	...	α	...	α	...

va α lardan birini β ga almashtirish kerak bo`lsin. Bunday almashtirish $\alpha \to \beta$ formula yordamida amalga oshiriladi. Ammo, agar biz bu formulani $P$ so`zga qo`llasak, u holda birinchi $\alpha$ almashtiriladi. Agar ikkinchi yoki oxirgi $\alpha$ ni almashtirish kerak bo`lsa nima qilish kerak? β ga birinchi α emas, balki boshqasi almashtirilishi uchun, bu boshqani qandaydir belgilash, ajratish kerak. Buning uchun uning yoniga (chapga yoki o`ngga) $P$ ning tarkibiga kirmaydigan qandaydir belgi, masalan *, qo`yish kerak:

P	...	α	...	$*\alpha$	...	α	...

Bunday belgini keyinchalik *maxsus belgi* deb ataymiz. Uning vazifasi – kerakli α ni boshqalaridan ajratishdir. Modomiki, faqat shu tashkil etuvchining oldida maxsus belgi turganligi tufayli, aynan shu α ni β ga almashtirish uchun $*\alpha \to \beta$ formuladan foydalanish kerak.

Bu maxsus belgili usulni yodlab qolish kerak, negaki, MNA da u ko`p ishlatiladi.

Shuningdek, yana bir muammo qoladi: qanday qilib α ning kerakli yoniga maxsus belgini qo`yish mumkin? Quyidagi misolda buni ko`rib chiqamiz.

4-masala (almashadigan belgini maxsus belgi bilan fiksatsiyalash). A={0,1,2,3}. P bo`sh bo`lmagan so`z bo`lsin. Uni to`rtlik sanoq sistemasidagi manfiy bo`lmagan butun son sifatida qarab, ikkilik sanoq sistemasidagi yozuvi topilsin.

Masalan: 0123→00011011

Yechish. Ma'lumki, to`rtlik sanoq sistemasidagi sonni ikkilik sanoq sistemasidagi songa o`tkazish uchun, to`rtlikdagi har bir raqamni, mos keluvchi ikkilik raqamlari juftiga almashtiriladi: 0→00, 1→01, 2→10, 3→11.

Bunday almashtirish quyidagi MNA yordamida amalga oshiriladi:

$$\begin{cases} 0 \to 00 & (1) \\ 1 \to 01 & (2) \\ 2 \to 10 & (3) \\ 3 \to 11 & (4) \end{cases}$$

Lekin bu algoritm to`g`ri emas, bunga 0123 kirish so`zi misolida ishonch hosil qilish mumkin:

$$\underline{0}123 \xrightarrow{1} \underline{0}0123 \xrightarrow{1} \underline{0}00123 \xrightarrow{1} \dots$$

Bu yerdagi xatolik shundaki, 4 likdagi sonlarni 2 lik sanoq sistemasida ifodalagach, bu sanoq sistemasidagi raqamlar o`xshash bo`lganligi uchun ularni farqlab bo`lmay qoladi va MNA ikkilik raqamlarni ham almashtirishni boshlaydi. Shuning uchun bu yerda sonning almashtirilgan va almashtirilmagan qismlarini bir-biridan ajratish kerak. Buning uchun * maxsus belgidan foydalanamiz va har bir almashtirilgan raqamdan so`ng uni almashmagan raqam tomon surib boramiz:

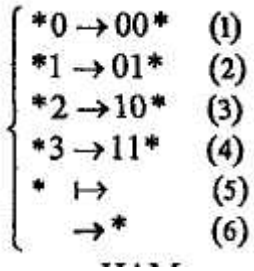
$$0123 \to *0123 \to 00*123 \to 0001*23 \to 000110*3 \to 00011011*$$

Ko`rinib turibdiki, * belgisidan chap tomonda sonning ikkilikka o`tkazilgan qismi, o`ng tomonida esa o`zgartirilmagan qismi joylashgan. Shuning uchun to`rtlik va ikkilik raqamlarini farqlashda hech qanday chalkashlik bo`lmaydi.

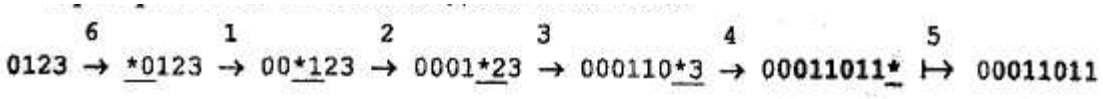
Shunday qilib, * maxsus belgi dastlab sonning chap tomoniga qo`yiladi va har bir qadamda to`rtlik raqamni ikkilik raqamga almashtirgan holda o`ngga siljib

boradi. Oxir oqibatda * belgisidan o`ng tomonda hech qanday raqam qolmaydi, keyin * ni yo`qotish va to`xtash kerak. Qanday qilib bu belgini boshida chap tomonga kiritib, keyin to`xtash oldidan o`chirib tashlash kerak? Bu bizga oldingi misollardan ma'lum, maxsus belgi * ni sonlardan "sakratib" o`tkazish *α→βy* formula bilan amalga oshiriladi, bunda α – tortlik raqam, βy – esa ikkilikdagi unga mos keluvchi raqamlar jufti.

Natijada quyidagi algoritmga ega bo`lamiz:



Bu MNA ni 0123 kirish so`zida tekshiramiz:



MAVZU BO`YICHA TOPSHIRIQLAR

1. A={f,h,p}. P so`zdagi *ph* juftlikni *f* ga almashtiring.
2. A={f,h,p}. P so`zdagi faqat birinchi *ph* juftlikni *f* ga almashtiring, agar u bo`lsa.
3. A={a,b,c}. P so`zining chap tomonidan *bac* so`zini yozing.
4. A={a,b,c}. P so`zini bo`sh so`zga almashtiring, ya'ni P ning barcha belgilarni o`chiring.
5. A={a,b,c}. Ixtiyoriy kirish so`zini *a* so`ziga almashtiring.

6 - AMALIY MASHG`ULOT: Algoritm tahlili asoslari. Algoritmlar tahlili uchun matematik tushunchalar.

Ishning maqsadi: Algoritm tahlili asoslarini aniq misollar asosida tushuntirish. Dastur psevdokodlarini tahlil qilish. Amallar sonining minimal va maksimal

qiymatlarini topish usullarini o`rganish. Algoritm tahlilida matematik tushunchalardan foydalanish.

I. Nazariy qism

Algoritm tahlili kompyuterga yoki dasturlash tiliga bog`liq emas, shuning uchun ham quyida algoritmlarni psevdokodlar yordamida keltiramiz. Algoritmni tahlil qilib, qo`yilgan masalani ushbu algoritm bilan yechish uchun qancha vaqt talab qilinishi haqida tasavvur hosil qilish mumkin.

Har bir qaralayotgan algorimtni N o`lchovli boshlang`ich ma'lumotlar massivida masalaning qanchalik tez yechilishi bilan baholaymiz.

Masalan, saralash algoritmi N ta qiymatdan iborat ro`yxatni o`sish tartibida joylashtirish uchun qancha taqqoslash talab qiladi yoki N*N o`lchamli ikkita matritsani ko`paytirishda qancha arifmetik amallar zarurligini hisoblash.

Bitta masalani turli algoritmlar bilan yechish mumkin. Algoritmlar tahlili bizga algoritmni tanlash uchun qurol bo`ladi. To`rtta qiymatdan eng kattasini tanlaydigan ikkita algoritmni qaraymiz:

```
largest = a
if b > largest then
largest = b
end if
return a
if s > largest then
largest = s end if if d > largest then
largest = d end if return largest
```

```
if a > b then if a > s then I f a > d then
return a else
return d end if else
if  s > d then
return s else
return d end if end if else
if b > s then if b > d then
return b else
return d end if else
if s > d then
return s else
return d end if end if end if
```

Ko`rinib turibdiki, yuqorida qaralayotgan algoritmlarning har birida uchta taqqoslash bajariladi. Birinchi algoritmni o`qish va tushunish oson, ammo kompyuterda bajarilish nuqtai nazaridan ularning murakkablik darajalari teng. Bu ikki algoritm vaqt nuqtai nazaridan teng, lekin birinchi algoritm largest nomli qo`shimcha o`zgaruvchi hisobiga ko`proq xotira talab qiladi.  Agarda son yoki belgilar taqqoslansa, ushbu qo`shimcha o`zgaruvchi katta ahamiyatga ega

bo`lmaydi, lekin boshqa turdagi ma’lumotlar bilan ishlaganda bu muhim ahamiyatga ega. Ko`plab zamonaviy dasturlash tillari katta va murakkab ob’ektlarni yoki yozuvlarni taqqoslash operatorlarini aniqlash imkonini beradi. Bunday hollarda qo`shimcha o`zgaruvchilarni joylashtirish katta joy talab qiladi. Algoritmlarning effektivligini tahlil qilishda, bizni birinchi navbatda vaqt masalasi qiziqtiradi, ammo xotira muhim rol o`ynaydigan vaziyatda uni ham muhokama qilamiz.

Algoritmlarning turli xossalari bitta masalani yechuvchi ikki turdagi algoritmlarning effektivligini taqqoslash uchun xizmat qiladi. Shuning uchun biz, hech qachon matritsalarni ko`paytirish algoritmi bilan saralash algoritmini emas, balki ikkita turli saralash algoritmlarini bir-biri bilan taqqoslaymiz.

Algoritm tahlilining natijasi – belgilangan algoritmning kompyuterda qancha vaqt yoki takrorlash talab qilishini aniq hisoblovchi formula emas. Bunday ma’lumot muhim emas, bu holatda kompyuter turi, u bitta yoki undan ortiq foydalanuvchi tomonidan ishlatilayaptimi, uning protsessori va chastotasi qanaqa, protsessor chipida komandalar to`liqmi va kompilyator bajarilayotgan kodni qay darajada amalga oshirmoqda kabi tomonlarni nazarda tutish kerak.

Bu shartlar algoritm bajarilish natijasida dasturning ishlash tezligiga ta’sir qiladi. Yuqoridagi shartlar hisobiga dasturni boshqa tez ishlaydigan kompyuterga o`tkazilganda algoritm yaxshi ishlaganday bajarilishi tezroq amalga oshadi. Aslida esa unday emas, biz shuning uchun tahlilimizda kompyuterning imkoniyatlarini inobatga olmaymiz.

II. Amaliy qism.

Oddiy va katta bo`lmagan dasturlarda bajariladigan amallar sonini N ning funksiyasi ko`rinishida aniq hisoblash mumkin. Aksariyat holatlarda bunga zaruriyat qolmaydi. Masalan, $N + 5$ ta va $N + 250$ ta amal bajariladigan ikki algoritm orasida N ning yetarlicha katta qiymatlarida deyarli farq bo`lmaydi. Shunga qaramay, biz algoritmlarni bajariladigan amallar soniga qarab tahlil qilamiz.

Algoritm tomonidan bajariladigan jarayonlar borki, biz ularning hammasini hisoblab o`tirmaymiz, buning sababi shundaki, hatto uning eng kichik sozlashi ham samaradorlikning sezilmas yaxshilanishiga olib keladi. Fayldagi turli belgilar sonini hisoblovchi algoritmni qaraymiz. Bu masala yechimi uchun algoritmning taxminiy ko`rinishi quyidagicha bo`ladi:

for all 256 belgilarni do
hisoblagichni nolga tenglash end for
while agar faylda belgi qolsa do
navbatdagi belgini ko`rsat va hisoblagichni bittaga oshir end while
do
hisoblagichni nolga tenglashtirish end for
while faylda belgi mavjud bo`lsa do
navbatdagi belgini ko`rsat va hisoblagichni bittaga oshir
end while

Ushbu algoritmni ko`rib chiqamiz. U takrorlanish bajarilishida 256 ta o`tish qiladi. Agar berilgan faylda N ta belgi bo`lsa unda ikkinchi takrorlanishda *N* ta o`tish qilinadi. «Bu qanday hisoblash?» degan savol tug`iladi. **For** siklida avval sikl o`zgaruvchisi bajariladi, keyin har bir o`tishda uning sikl chegarasidan chiqmayotganligi tekshiriladi va o`zgaruvchi qiymatini oshiradi. Bu esa sikl bajarilishida 257 yuklash bajariladi (biri sikl o`zgaruvchisi, 256 tasi hisoblagich uchun), ya’ni 256 ta oshirish va 257 ta sikl chegarasidan chiqmaganligini tekshirish (bitta amal siklni to`xtatish uchun qo`shilgan). Ikkinchi siklda *N*+1 marta shart tekshiriladi (+1 fayl bo`sh bo`lgandagi oxirgi tekshiruv), va *N h*isoblagichni oshirish.

Jami amallar:

Oshirish	*N + 256*
Yuklash	*257*
Shartlarni tekshirish	*N + 258*

Shunday qilib, 500 belgidan iborat fayl berilsa algoritmda 1771 ta amal bajariladi, ulardan 770 tasi natija beradi (43%). Endi N ning qiymati oshganda

nima bo`lishini ko`ramiz. Agar fayl 50 000 belgidan iborat bo`lsa, unda algoritm 100 771 amal bajaradi, ularning 770 tasi natija uchun (jami amallar sonining 1% ini tashkil etadi). yechimga qaratilgan amallar soni oshmayapti, lekin $N$ katta bo`lganda ularning foizi juda kam.

Endi boshqa tomoniga e'tibor qaratamiz. Kompyuterda ma'lumotlar bilan shunday ishlashga mo`ljallanganki, katta xajmdagi ma'lumotlar blokini ko`chirish va yuklash bir xil tezlikda amalga oshiriladi. SHuning uchun biz avval 16 ta hisoblagichga boshlang`ich qiymat 0 ni yuklaymiz, keyin qolgan hisoblagichlarni to`ldirish uchun shu blokdan 15 ta nusxa olamiz.Bu esa sikl bajarilish davomida tekshirishlar sonini 33 ga, yuklashlar sonini 33 va oshirishlar sonini 31 ga kamayishiga olib keladi. Demak amal bajarilishlar soni 770 dan 97 gacha kamaydi, ya'ni 87%. Agar erishilgan natijani 50000 belgidan iborat fayl ustida bajarsak, tejamkorlik 0.7% ni tashkil qiladi (100771 ta amal o`rniga 100098 amal bajaramiz).

Agarda barcha amallarni sikldan foydalanmay 31 ta yuklashlar orqali bajarganimizda, vaqtni yanada tejagan bo`lardik, ammo bu usul 0.07 foyda keltiradi. Ishimiz unumli bo`lmaydi.

Ko`rib turganimizdek, algoritmning bajarilish vaqti bilan bog`liq barcha amallar befoyda. Tahlil tili bilan aytganda, boshlang`ich ma'lumotlar hajmining ortishiga aloxida e'tibor qaratish kerak.

Avvalgi ishlarda algoritmlarni tahlil qilishda algoritmlarni Tyuring mashinasida hisoblash aniqlangan. Tahlilda masalani yechish uchun zarur bo`lgan o`tishlar soni hisoblangan. Bu turdagi tahlil to`g`ri bo`lib, ikki algoritmning nisbiy tezliklarini aniqlash imkonini beradi, ammo uning amaliyotda qo`llanilishi kam, chunki ko`p vaqt talab qiladi.

Avval bajariladigan algoritmning Tyuring mashinasidagi o`tish funksiyalarini yozish, keyin esa bajarilish vaqti hisoblanadi.

Algoritmlarni tahlil qilishning boshqa yaxshiroq usuli - uni biror yuqori bosqichli til Pascal, C, C++, JAVA da yozish yoki oddiy psevdokodlarda yozishdir. Barcha algoritmlarning asosiy boshqaruv strukturasini ifodalaganda

psevdokodlarning xossalari ahamiyatga ega emas. Ixtiyoriy til bizning talabimizga javob beradi, chunki **for yoki while** shaklidagi sikllar, **if, case yoki switch** ko`rinishidagi tarmoqlanish mexanizmlari barcha dasturlash tillarida mavjud. Har gal biz bitta aniq algoritmni ko`rib chiqishimizga to`g`ri keladi – unda birdan ortiq funksiya yoki programma fragmenti kiritilgan bo`ladi, shuning uchun yuqorida keltirilgan tillarning tezligi umuman muhim emas. Psevdokodlardan foydalanishimizning sababi shunda.

Algoritmni tahlil qilishda kiruvchi ma’lumotlarni tanlash uning bajarilishiga ta’sir qilishi mumkin. Aytaylik, ba’zi saralash algoritmlari, agar kirish ro`yxati saralangan bo`lsa, juda tez ishlashi mumkin, boshqa algoritmlar shunday ro`yxatda uncha katta bo`lmagan natijani ko`rsatadi. Tasodifiy ro`yxatda esa natija buning teskarisi bo`lishi mumkin. Shuning uchun biz ma’lumotlarning bir kirish ro`yxatidagi algoritmlar harakatini tahlil qilish bilan chegaralanmaymiz. Biz algoritmni ham eng tez, ham eng sekin ishlashini ta’minlovchi ma’lumotlarni qidiramiz. Bundan tashqari, biz barcha mavjud ma’lumotlar to`plamidagi algoritmlarning o`rtacha samarasini ham baholaymiz.

Eng yaxshi holat

Bo`limning nomlanishidan ham ko`rinib turibdiki, algoritmlar uchun eng yaxshi holat bu qisqa vaqt ichida amalga oshiriladigan algoritmning ma’lumotlar jamlanmasi. Bunday jamlanma algoritm eng amal bajaradigan qiymatlar kombinatsiyasini ifodalaydi. Agar biz izlash algoritmini tekshirsak, izlangan qiymat birinchi algoritm tekshirayotgan katakka yozilgan bo`lsa (odatda maqsadli qiymat yoki kalit deb ataladi), ma’lumotlar to`plami eng yaxshi hisoblanadi. Bunday algoritmga uning murakkabligidan qatiy nazar, bitta taqqoslash kerak bo`ladi. Shuni eslatish kerakki, ro`yxatdan izlashda, uning qanchalik uzun bo`lishidan qatiy nazar, eng yaxshi holat doimiy vaqtni talab qiladi. Umuman, eng yaxshi holatda algoritmni bajarish vaqti kichik yoki doimiy bo`ladi, shuning uchun biz bunday tahlilni kam o`tkazamiz.

Eng yomon holat

Eng yomon holatni tahlil qilish juda muhim, chunki u algoritm ishining maksimal vaqtini tasavvur qilishga yordam beradi. Eng yomon holatni tahlil qilganda algoritm eng ko`p ish bajaradigan kirish ma’lumotlarini topish zarur. Izlovchi algoritm uchun bu kabi kiruvchi ma’lumotlar – bu shunday ro`yxatki, unda izlangan kalit oxirida keladi yoki umuman bo`lmaydi. Natijada N taqqoslash kerak bo`ladi. Eng yomon holatning tahlili tanlangan algoritmga qarab dasturning ishlash vaqti uchun yuqori bahoni beradi.

O`rtacha holat

O`rta holatning tahlili eng murakkab hisoblanadi, chunki u ko`pgina detallarni hisobga olishni talab qiladi. Tahlil asosini mavjud bo`lgan kiruvchi ma’lumotlar to`plamini bo`lib chiqish lozim bo`lgan turli guruhlarni aniqlash tashkil qiladi. Ikkinchi qadamda kiruvchi ma’lumotlar to`plami qaysi guruhga tegishli bo`lish ehtimoli aniqlanadi. Uchinchi qadamda har bir guruhdagi ma’lumotlarga algoritmning ish vaqti hisoblanadi. Algoritmning bir guruhdagi hamma kiruvchi ma’lumotlar uchun ishlash vaqti bir xil bo`lishi kerak, aks holda guruhni bo`lish lozim. O`rtacha ish vaqti

$$A(n)\sum_{i=1}^{m} p_i \cdot t_i, \qquad (1)$$

formula orqali hisoblanadi. Bu yerda n - kiruvchi ma’lumotlar o`lchami, $m$ - guruhlar soni, $p_i$ - kiruvchi ma’lumotlarning $i$ sonli guruhga tegishlilik ehtimoli, $t_i$ – $i$ sonli guruhdagi ma’lumotni qayta ishlash uchun algoritmga kerak bo`ladigan vaqt deb belgilangan.

Ba’zi hollarda biz kiruvchi ma’lumotlarning har bir guruhga tushish ehtimolini bir Xill deb taxmin qilamiz. Boshqacha aytganda, agar guruh 5 ta bo`lsa, birinchi guruhga tushish extimoli ikkinchi yoki boshqa guruhga tushish extimolidek, ya’ni har bir guruhga tushish extimoli 0,2 ga teng. Bu holda ishning o`rtacha vaqtini avvalgi formula Bilan yoki unga ekvivalent soddalashtirilgan barcha guruhlarning teng extimolligida haqiqiy bo`lgan

$$A(n)\sum_{i=1}^{m} p_i \cdot t_i, \qquad (2)$$

formuladan foydalanishimiz mumkin.

Algoritm tahlili uchun zarur matematik tushunchalar.

Algoritmlar tahlilini olib borishda biz ba'zi muhim matematik tushunchalardan foydalanamiz. Bunga sonni chapdan va o`ngdan yaxlitlash ham kiradi. $X$ sonni chapdan (yoki butun qismini) yaxlitlash deganda, qiymati $X$ sondan oshmaydigan eng katta butun sonni tushunamiz va u $\lfloor X \rfloor$ kabi belgilanadi. Masalan, $\lfloor 2.5 \rfloor = 2$, $\lfloor -7.3 \rfloor = -8$. O`ng tomondan yaxlitlash deganda esa, qiymati X sondan kichik bo`lmagan eng kichik butun sonni tushunamiz va u $\lceil X \rceil$ kabi belgilanadi. Masalan, $\lceil 2.5 \rceil = 3$, $\lceil -7.3 \rceil = -7$. Bu chapdan va o`ngdan yaxlitlashni ba'zi kattaliklar uchun u yoki bu amallarning necha marta bajarilganini hisoblashda qo`llash mumkin. Masalan, $N$ ta elementni jufti bilan taqqoslashda, yani birinchi ikkinchi bilan, uchinchi to`rtinchi bilan va h.k., taqqoslashlar soni $\lfloor N/2 \rfloor$ ga teng bo`ladi. Agar $N = 10$ bo`lsa, u holda jufti bilan taqqoslash 5 ga, ya'ni $\lfloor 10/2 \rfloor = \lfloor 5 \rfloor = 5$ bo`ladi. Agarda $N - 11$ bo`lsa, u holda taqqoslashlar soni o`zgarmaydi, ya'ni $\lfloor 11/2 \rfloor = \lfloor 5.5 \rfloor = 5$.

Natural N sonining faktoriali $N!$, 1 dan N gacha bo`lgan barcha natural sonlarning ko`paytmasini ifodalaydi. Masalan, $3! = 3 \cdot 2 \cdot 1 = 6$, $6! = 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 720$. Ko`rinib turibdiki, faktorialning qiymati $N$ oshishi bilan juda tez o`sadi.

Logarifmlar.

Bizning tahlilimizda logarifmlar sezilarli o`rin egallaydi, shuning uchun ularning xossalarini ko`rib chiqishimiz lozim. *x* sonining *u* asos bo`yicha logarifmi deb shunday darajaga aytiladiki, *x* ni olish uchun *u* ni ko`tarish kerak bo`ladi. Masalan, $\log_{10}45$ taxminan 1,653 ga teng, chunki $10^{1.653} \sim 45$. Logarifmning asosi

ixtiyoriy son bo`lishi mumkin, lekin bizning tahlilimizda ko`proq 10 va 2 asosli logarifmlar uchraydi.

Logarifm – o`sib boruvchi funksiya. Bu degani, agar $X>Y$ bo`lsa, har bir B asos uchun $\log_B X > \log_B Y$. Logarifm – o`zaro bir ifodali funksiya. Bu degani, agar $\log_B X = \log_B Y$ bo`lsa $X=Y$ bo`ladi. Shuningdek, logarifmning unga kiruvchi o`zgaruvchilarning musbat qiymatida to`g`ri bo`lgan quyidagi muhim xossalarini bilish lozim:

$$\log_B 1 = 0;$$

$$\log_B B = 1;$$

$$\log_B(XY) = \log_B X + \log_B Y;$$

$$\log_B X^Y = Y\log_B X;$$

$$\log_A X = \frac{\log_B X}{\log_B A}$$

shu xossalar yordamida funksiyani soddalashtirish mumkin. Bir asosdan boshqasiga o`tish xossasidan foydalanib, ko`plab kalkulyatorlar 10 asosli logarifmlar va natural logarifmlarni hisoblaydi. $\log_{42}75$ ni hisoblash uchun nima qilamiz?

Binar daraxtlar

Binar daraxti shunday tuzilishga egaki, undagi har bir tugun ikkita tugundan ortiq bo`lmagan bir ajdod nasldan iborat bo`ladi. Daraxtning eng yuqori tuguni yagona ajdodsiz tugun hisoblanadi; u ildizli tugun deb ataladi. N tugunli binar daraxti kam $[\log_2 N+1]$ tugunga ega (tugunlarning maksimal zichligida). Masalan, 15 tugunli to`la binar daraxtida bir ildiz, ikkinchi darajada 2 ta tugun, 3-darajada 4 ta tugun va 4-darajada 8 ta tugun bor; bizning tengligimiz ham $[\log_2 15]+1=[3.9]+1=4$ darajani beradi. Daraxtga yana bir tugunning qo`shilishi yangi darajaning hosil bo`lishiga olib keladi va ularning soni teng bo`ladi $[\log_2 16] + 1 = [4] + 1 = 5$. N tugunli eng katta binar daraxti N darajaga ega: bu daraxtning har bir tugunida bitta nasl bor (daraxtning o`zi ham oddiy ro`yxat ko`rinishiga ega).

Agar daraxtning darajalarini raqamlab chiqsak, ildiz 1 darajada deb hisoblab, *K* raqamli darajada 2^{K-1} tuguni yotadi. *J* darajali (1 dan *J* gacha raqamlangan) to`la binar daraxtida hamma barglar *J* raqamli darajada yotadi va har bir tugunda birdan *J-1* darajada ikkita to`g`ridan-to`g`ri nasl bor. *J* darajali to`la binar daraxtida 2^J - 1 tugun bor. Bu axborot keyinchalik ham sizga asqotishi mumkin. Bu formulalarni yaxshiroq tushunish uchun binar daraxtlarini chizishni va natijalaringizni yuqoridagi formulalar bilan solishtirishingizni maslahat beramiz.

Ehtimolliklar

Biz algoritmlarni kiruvchi ma'lumotlarga ko`ra tahlil qilmoqchimiz, buning uchun esa u yoki bu kiruvchi ma'lumotlar to`plami qanchalik ko`p uchrashini baholashimiz kerak. Shu bilan birga, biz kiruvchi ma'lumotlar u yoki bu sharoitlarga to`gri kelish extimolligi bilan ishlashimizga to`g`ri keladi. U yoki bu hodisaning extimolligi nol va bir oralig`idagi sondan iborat, 0 extimolligi hodisa hech qachon sodir bo`lmasligi,1 extimoli esa bo`lishi mumkinligini bildiradi. Agar bizga turli kiruvchi qiymatlarning soni aniq 10 ga tengligi ma'lum bo`lsa, ishonch Bilan aytishimiz mumkinki, har qanday bunday kirishning extimolligi 0 va 1 oralig`ida bo`ladi, barcha extimolliklarning yig`indisi 1 ga teng, chunki ulardan bittasi amalga oshishi mumkin. Agar har bir kirishning amalga oshish extimolligi bir xil bo`lsa, ulardan har birining extimolligi 0.1 ga teng bo`ladi (10 dan 1 yoki 1/10).

Bizning tahlilimiz, asosan barcha imkoniyatlarni ko`rib chiqishdan iborat bo`ladi, keyin esa biz ularning hammasi teng extimolli deb faraz qilamiz. Agar imkoniyatlarning umumiy soni N ga teng bo`lsa, ulardan har birining amalga oshishi extimolligi 1/N ga teng bo`ladi.

Qo`shish formulalari

Algoritmlarni tahlil qilganda biz ba'zi kattaliklar yig`indisini qo`shishimizga to`g`ri keladi. Aytaylik, bizda sikli algoritm bor. Agar sikl o`zgaruvchisi 5 qiymatini olsa, sikl 5 marta bajariladi, agar uning qiymati 20 ga teng bo`lsa 20 bo`ladi. Agar sikl o`zgaruvchisi M ga teng bo`lsa, sikl M marta bajariladi. Agar

sikl o`zgaruvchisi 1 dan N gacha hamma qiymatlarga o`tsa, sikl bajarilishining jami soni 1 dan N gacha bo`lgan hamma natural sonlar yig`indisiga teng bo`ladi. Bu yig`indini biz $\sum_{i=1}^{N} i$ ko`rinishida yozamiz. Yig`indi belgisining pastki qismida o`zgaruvchi yig`indining boshlang`ich qiymati, yuqori qismida esa – oxirgi qiymati turibdi. Bunday ifodalanish bizni qiziqtirgan yig`indi bilan qanday bog`liqligi tushunarli.

Agar biror qiymat shu kabi yig`indi ko`rinishida yozilsa, natijani boshqa shu kabi ifodalar bilan solishtirish mumkin bo`lishi uchun uni soddalashtirish kerak. Ikki sondan kattasi $\sum_{i=11}^{N}(i^2-i)u\sum_{i=0}^{N}(i^2-20i)$. SHuning uchun yig`indini soddalashtirish uchun biz quyidagi formulalardan foydalamiz, bunda S - i ga bog`liq bo`lmagan o`zgaruvchi.

$$\sum_{i=1}^{N} Ci = C\sum_{i=1}^{N} i$$

$$\sum_{i=L}^{N} i = \sum_{i=0}^{N-L}(i+L)$$

$$\sum_{i=L}^{N} i = \sum_{i=0}^{N} i - \sum_{i=0}^{L-1} i$$

$$\sum_{i=1}^{N}(A+B) = \sum_{i=1}^{N} A + \sum_{i=1}^{N} B$$

$$\sum_{i=0}^{N}(N-i) = \sum_{i=0}^{N} i$$

tengligi 0 dan N gacha sonlar yig`indisi N dan 0 gacha sonlar yig`indisiga tengligini bildiradi. Ba’zi hollarda bu tenglikni qo`llash hisoblashlarni yengillashtiradi.

$$\sum_{i=1}^{N} 1 = N$$

$$\sum_{i=1}^{N} C = CN$$

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$$

tengligini yodlab olish oson, agar 1 dan N gacha bo`lgan sonlarni juftlikka ajratsak. 1 ni N ga qo`shib, 2 ni N-1 ga va hokazo, biz har biri N+1 ga teng sonlar to`plamini olamiz. Bunday sonlar nechta bo`ladi? Albatta, ularning soni juft qilib ajratilgan elementlar sonining yarmiga teng, ya'ni N/2.

$$\frac{N}{2}(N+1) = \frac{N(N+1)}{2}$$

$$\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6} = \frac{2N^3 + 3N^2 + N}{6}$$

$$\sum_{i=0}^{N} 2^i = 2^{N+1} - 1$$

tengligini ikki bo`linadigan sonlar orqali eslab qolishi mumkin. 0 dan 10 gacha bo`lgan ikki darajasining yig`indisi 11111111111 ikkilangan soniga teng. Bu songa 1 ni qo`shib 100000000000, ya'ni 2^{11} ni hosil qilamiz. Lekin bu natija noldan o`nggacha bo`lgan ikki darajalarining yig`indisidan 1 ga ko`p, shuning uchun yig`indining o`zi 2^{11}-1 ga teng. Endi agar 10 o`rniga N qo`ysak, biz tenglikka kelamiz.

Har qanday son uchun

$$\sum_{i=1}^{N} A^i = \frac{A^{N+1} - 1}{A - 1} \text{ ixtiyoriy A soni uchun}$$

$$\sum_{i=1}^{N} i2^i = (N-1)2^{N+1} + 2$$

$$\sum_{i=1}^{N} \frac{1}{i} \approx \ln N$$

$$\sum_{i=1}^{N} \log_2 i \approx N \log_2 N - 1.5$$

Yig`indilarni soddalashtirishda avval ularni yuqoridagi tengliklar yordamida yanada oddiy sonlarga ajratish, so`ngra yig`indilarni boshqa ayniyatlar yordamida almashtirish mumkin.

MAVZU BO`YICHA TOPSHIRIQLAR

1. Matnli fayldagi bosh harflar sonini hisoblovchi algoritmning psevdokodini yozing. Bu algoritmda nechta taqqoslashlar soni

zarurligini aniqlang. Hisoblagichning amallar soni mumkin bo`lgan maksimal va minimal qiymatini aniqlang (Berilgan fayldagi belgilar sonini *N* deb oling).

2. Bizga ularning soni ma'lum bo`lmagan bir nechta sonlar nabori berilgan. Ularning o`rta arifmetik qiymatini hisoblovchi algoritm psevdokodini yozing. Bu algoritmda qanday tipdagi amal bajariladi va har bir tipda nechta amal bajariladi?
3. Oddiy kalkulyator natural (e asosli) va o`nli (10 asosli) logarifmlarni hisoblay oladi. Bu amallar yordamida $\log_{27} 59$ ni qanday hisoblash mumkin?
4. Bizga narda o`yinida ishlatiladigan, har bir yog`ida 1 dan 6 gacha sonlarga mos keluvchi nuqtalar tasvirlangan tosh berilgan bo`lsin. Toshni tasodifiy tashlaganda 1 dan 6 gacha sonlarning tushush ehtimolini hisoblang. Agar shunday toshdan ikkitasini tashlaganda sonlar yig`indisi qanday oraliqda o`zgarishi mumkin.
5. Quyidagi ifodalarda "yig`indi" belgisisiz ekvivalentlarini toping:

1) $\sum_{i=1}^{N}(2i+7)$;
2) $\sum_{i=1}^{N}(i^2-2i)$;
3) $\sum_{i=7}^{N} i$;
4) $\sum_{i=5}^{N}(2i^2+1)$;
5) $\sum_{i=1}^{N} 6^i$;
6) $\sum_{i=7}^{n} 4^i$.

7 - AMALIY MASHG`ULOT: Algoritmlarning o`sish tezliklarini tahlil qilish.

Ishning maqsadi: Algoritm o`sish tezliklarini tahlil qilish. Katta omega, katta o va katta teta funksiyalar sinflari asosida algoritmlarni tahlil qilish.

I. Nazariy qism

Algoritmdagi amallar soni uni tahlil qilishda muhim rol o`ynamaydi. Kiruvchi ma'lumotlarning hajmi ko`payganida bu sonning o`sish tezligi muhimroq hisoblanadi. U algoritmning ***o`sish tezligi*** deb ataladi.

O`sish tezliklarini tasniflash.

Algoritm murakkabligining o`sish tezligi muhim rol o`ynaydi va biz o`sish tezligi formulasi kata ustunlikka ega hadi bilan aniqlanishini ko`rdik. Shuning uchun biz sekin o`sadigan kichik hadlarga e'tibor qaratmaymiz. Barcha kichik hadlarni olib tashlab, murakkablikning o`sish tezligi hisoblanuvchi algoritm yoki funksiyaning tartibiga ega bo`lamiz. Algoritmlarni ular murakkabligining o`sish tezligiga qarab guruhlarga ajratish mumkin. Biz 3 toifani kiritamiz: murakkabligi mazkur funksiya kabi tez o`suvchi algoritmlar, murakkabliklari o`sha tezlikda o`suvchi algoritmlar va murakkabligi bu funksiyadan sekin o`suvchi algoritmlar.

Katta omega

f singari tez o`suvchi funksiyalar sinfini biz $\Omega(f)$ orqali belgilaymiz (katta omega deb o`qiladi). Agar hamma qiymatlarda erkin o`zgaruvchan kattalik $n$ va musbat $c$ son uchun $g(n) > c\,f(n)$ o`rinli bo`lsa, $g$ funksiya shu sinfga tegishli bo`ladi. $\Omega(f)$ sinfi o`zining quyi chegarasi bilan izohlansa, undagi hamma funksiyalar $f$ kabi tez o`sadi.

Biz algoritmlarning samaradorligi bilan qiziqamiz, shuning uchun $\Omega(f)$ sinfi bizni u darajada qiziqtirmaydi: masalan, $\Omega(n^2)$ ga n^2 dan tez o`suvchi hamma funksiyalar kiradi, aytaylik n^3 va 2^n.

Katta O

Spektrning boshqa tarafida $O(f)$ sinfi joylashgan (katta O deb o`qiladi). Bu sinf $f$ dan tez o`smaydigan funksiyalardan tashkil topgan. Funksiya f, $O(f)$ sinflari uchun yuqori chegarani belgilaydi. Formal nuqtai nazardan g funksiyasi $O(f)$ sinfiga tegishli, agar barcha n va qandaydir musbat c konctanta uchun $g(n) < c\,f(n)$ o`rinli bo`lsa.

Bu sinf biz uchun juda muhim. Ikkita algoritmdan birinchisining murakkabligi ikkinchisiga qaraganda katta O sinfiga tegishli bo`lsa, u holda ikkinchi algoritm birinchisiga qaraganda qo`yilgan masalani yaxshi yecha olmaydi.

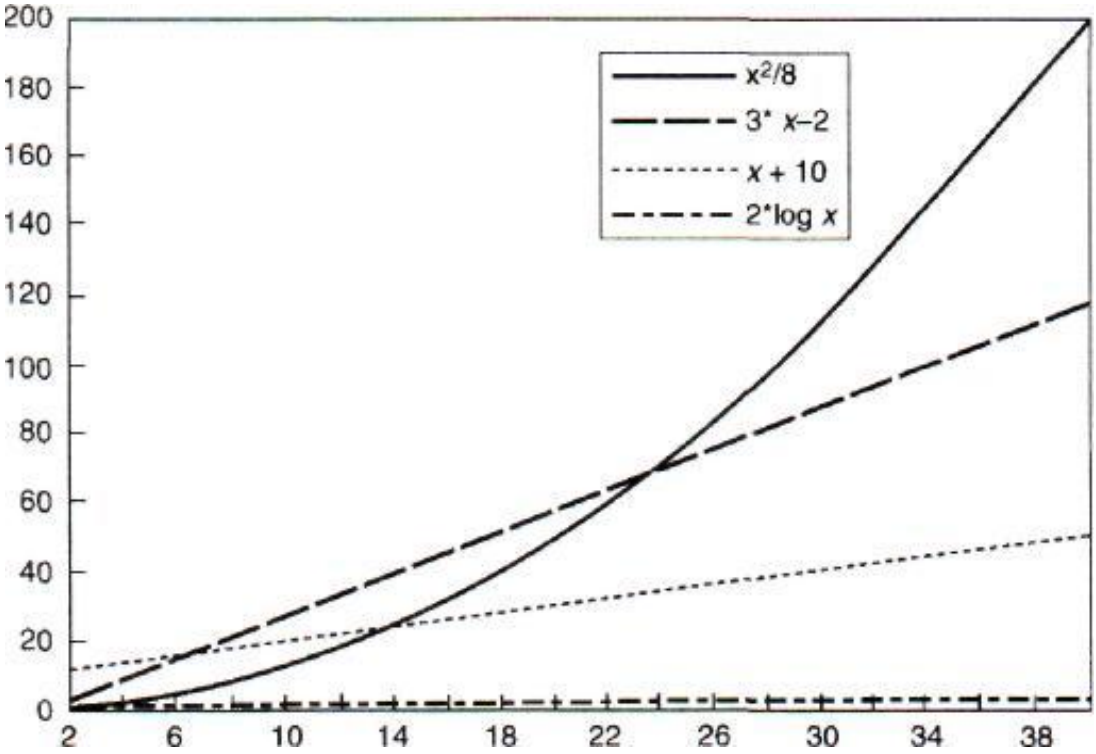
Katta teta

$\Theta(f)$ orqali biz f singari tezlikda o`suvchi funksiyalar sinfini belgilaymiz (katta teta deb o`qiladi). Formal nuqtai nazardan bu sinf ikki avvalgi sinflarning kesishmasidan iborat, $\Theta(f) = \Omega(f) \cap O(f)$.

Algoritmlarni taqqoslaganda bizni qaralayotgan masalani o`rganilganlaridan ko`ra tezroq yechadiganlari qiziqtiradi. Shuning uchun agar topilgan algoritm Θ sinfiga tegishli bo`lsa, u bizni unchalik qiziqtirmaydi.

II. Amaliy qism

Agar ***1-rasm***ga diqqat bilan qarasak, funksiya grafiklarining quyidagi xususiyatlarini ko`rsatish mumkin. $x^2$ funksiya avval sekin o`sadi, lekin x o`sganda uning o`sish tezligi ham oshadi. x funksiyasining o`sish tezligi o`zgaruvchining hamma qiymatlari oralig`ida doimiydir. 2 log $x$ funksiyasi umuman o`smaydi, lekin bu yolg`on tasavvur. Haqiqatda esa u o`sadi, faqat juda sekin.



1 – rasm. *To`rtta funksiyaning grafigi.*

Funksiya qiymatlarining nisbiy balandligi biz ko`rib chiqayotgan o`zgaruvchining qiymatlari katta yoki kichikligiga bog`liq. x=2 da funksiya

qiymatini taqqoslaymiz. Bu nuqtadagi eng kichik qiymatli funksiya $x^2/8$; eng kata qiymatli funksiya $x+10$ hisoblanadi. Biroq x ortishi bilan $x^2/8$ funksiya osha boshlaydi.

Algoritmlarni tahlil qilish paytida bizni algoritmning o`sish tezligi qiziqtiradi. Funksiyaning nisbiy «o`lchami» bizni x o`zgaruvchining katta qiymatlarida qiziqtiradi.

Ba'zi ko`p uchraydigan funksiya sinflari 2-rasmdagi jadvalda keltirilgan. Bu jadvalda biz berilgan sinfning funksiya qiymatini erkin o`zgaruvchan kattalik qiymatining keng diapazonida keltirdik. Ko`rinib turibdiki, kiruvchi ma'lumotlarning oz o`lchamlarida funksiya qiymatlari sezilarli farq qilmaydi, ammo bu o`lchamlari o`sganda farq sezilarli oshadi. Bu jadval 1-rasmga qaraganda taassurotni kuchaytiradi. Shuning uchun biz kiruvchi ma'lumotlarning katta hajmlarida nima sodir bo`lishini o`rganamiz, kichik hajmlardagi farq ko`rinmas bo`lganligi sababli.

1 va 2-rasmlardagi ma'lumotlar funksiyalarning yana bir xossasini namoyish etadi. Tez o`suvchi funksiyalar sekin o`suvchi funksiyalardan ustunlik qiladi. Shuning uchun biz agar algoritm murakkabligi ikki yoki bir necha funksiyalar yig`indisidan iborat bo`lsa, tez o`suvchi funksiyalardan tashqari barcha funksiyalarni olib tashlaymiz.

	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n
1	0.0	1.0	0.0	1.0	1.0	2.0
2	1.0	2.0	2.0	4.0	8.0	4.0
5	2.3	5.0	11.6	25.0	125.0	32.0
10	3.3	10.0	33.2	100.0	1000.0	1024.0
15	3.9	15.0	58.6	225.0	3375.0	32768.0
20	4.3	20.0	86.4	400.0	8000.0	1048576.0
30	4.9	30.0	147.2	900.0	27000.0	1073741824.0
40	5.3	40.0	212.9	1600.0	64000.0	1099511627776.0
50	5.6	50.0	282.2	2500.0	125000.0	1125899906842620.0
60	5.9	60.0	354.4	3600.0	216000.0	1152921504606850000.0
70	6.1	70.0	429.0	4900.0	343000.0	1180591620717410000000.0
80	6.3	80.0	505.8	6400.0	512000.0	1208925819614630000000000.0
90	6.5	90.0	584.3	8100.0	729000.0	1237940039285380000000000000.0

100	6.6	100.0	664.4	10000.0	1000000.0	126765060022823000000000000000.0

2 – rasm. *Funksiyalarning o`sish sinflari*

Masalan, agar biz algoritmni tahlil qilganda, uning x^3-30x taqqoslash qilinishini bilsak, algoritmning murakkabligi x^3 kabi o`sadi, deymiz. Buning sababi shundaki, 100 ta kiruvchi raqamlarda $x^3$ va  orasidagi farq atiga 0,3 % ni tashkil qiladi. Keyingi bo`limda biz bu fikrni formallashtiramiz.

Katta O sinfining tavsifi

Berilgan funksiya O(f) ga tegishli ekanligini ikki xil yo`l bilan tekshirish mumkin: yuqoridagi tavsif orqali yoki quyidagi tavsif yordamida:

$g \in O(f)$, àãàð $\lim_{n\to\infty} \frac{g(n)}{f(n)} = c$ ixtiyoriy c konstanta uchun.

Bu shuni anglatadiki, $g(n)/f(n)$ ning munosabatlar chegarasi mavjud bo`lsa va u cheksizlikdan kichik bo`lsa, $g \in O(f)$ bo`ladi. Ba’zi funksiyalar uchun buni tekshirish oson emas. Lopital qoidasiga ko`ra, bunday holda funksiyalar limitini ularning hosilasi limitiga almashtirish mumkin.

Belgilash

Θ(f), Ω(f) va O(f) sinflarining har biri to`plam hisoblanadi, shuning uchun «g – shu to`plam elementi» iborasi ahamiyatlidir. Biroq, tahlillarda $g = O(f)$ deb yozishadi, aslida esa $g \in O(f)$.

MAVZU BO`YICHA TOPSHIRIQLAR

1. Quyidagi funksiyalarni o`sish tartibida joylashtiring. Agar ularning ba’zilari bir xil tezlikda o`suvchi bo`lsa, ularni alohida ajratib ko`rsating:

2^n	$\log_2 \log_2 n$	$n^3 + \log_2 n$
$\log_2 n$	$n - n^2 + 5n^3$	2^{n-1}
n^2	n^3	$n \log_2 n$
$(\log_2 n)^2$	$\sqrt{n}$	6
$n!$	n	$(3/2)^n$

2. Quyida keltirilgan funksiyalar jufti uchun $f = O(g)$ yoki $g = O(f)$ tengliklardan biri o`rinli bo`lishini aniqlang:

a) $f(n) = (n^2 - n)/2, \quad g(n) = 6n$

b) $f(n) = n + 2\sqrt{n}, \quad g(n) = n^2$

c) $f(n) = n + n\log_2 n, \quad g(n) = n\sqrt{n}$

d) $f(n) = n^2 + 3n + 4, \quad g(n) = n^3$

e) $f(n) = n\log_2 n, \quad g(n) = n\sqrt{n}/2$

f) $f(n) = n + \log_2 n, \quad g(n) = \sqrt{n}$

g) $f(n) = 2\log_2 n^2, \quad g(n) = \log_2 n + 1$

h) $f(n) = 4n\log_2 n + n, \quad g(n) = (n^2 - n)/2$

8 - AMALIY MASHG`ULOT: Izlash va saralash algoritmlari.

Ishning maqsadi: Izlash va saralash algoritmlarini tahlil qilish, xususan, ketma-ket izlash, ikkilanma izlash, pufaksimon saralash, Shell saralashi algoritmlarini aniq misollarda tahlil qilish.

I. Nazariy qism

1. Izlash algoritmlari.

1.1. Ketma-ket izlash.

Ketma-ket izlash algoritmida bizni ro`yxatdan ma'lum bir maqsad elementi deb ataluvchi elementni izlash talab qilinadi. Ketma-ket izlashda biz har doim ro`yxat tartiblanmagan deb tasavvur qilamiz, bir qancha algoritmar tartiblangan ro`yxat bilan ishlashda juda yaxshi natijalar ko`rsatsa ham bunga qat'iy zarurat yo`q. Izlash asosan kerakli element ro`yxatda borligini tekshirishdan emas, balki elementga taalluqli ma'lumot, kalit so`zni olish uchun qo`llaniladi.

Masalan, kalit so`z ishchining nomeri, tartib raqami yoki istalgan mukammal identifikator bo`lishi mumkin. Kerakli kalit topilgandan keyin dastur tez-tez unga bog`liq ma'lumotlarni o`zgartiradi yoki barcha ma'lumotlarni ko`rsata oladi. Izlash algoritmidan oldin kalitning o`rnini aniqlash kabi muhim masala bor. Shu sababli izlash algoritmlari kerakli kalitni aniqlaydigan element o`rnini qaytaradi. Agar kalit so`z topilmasa, izlash algoritmi massivning yuqori chegarasidan oshadigan qiymatni qaytaradi. Bizning maqsad uchun ro`yxat elementlari 0 dan N gacha nomerlarga ega deb tasavvur qilamiz. Agar bu element

ro`yxatda bo`lmasa, 0 qaytarish imkonini beradi. Oddiylik uchun kalit so`z qaytarilmaydi deb tasavvur qilamiz.

Ketma-ket izlash algoritmalari ro`yxat bo`ylab birinchi elementdan boshlab qidirilayotgan element topilmagunga qadar bittadan elementni tekshirib chiqadi. Aniqki, kalit so`z qancha uzoqda joylashgan bo`lsa, uni izlashga shuncha ko`p vaqt ketadi. Ketma-ket izlash algoritmlarining analizi uchun buni eslab qolish talab qilinadi.

Ketma-ket izlashning to`liq algoritmi quyidagicha ko`rinishda:

```
SequentialSearch(list.target,N)
listspisok \\dlya prosmotra
Targe \\selevoye znacheniye
N  \\chislo elementov v spiske
for i=l to N do
if  (target=list[i]) return i
end if end for return
```

1.2. Ikkilanma izlash.

Qidirilayotgan qiymatni tartiblangan ro`yxatning o`rta elementi bilan solishtirganda uchta holat bo`lishi mumkin: qiymatlar teng, qidirilayotgan qiymat kichik yoyinki qidirilayotgan qiymat katta. Birinchisida juda yaxshi holatda qidiruv tugadi. Qolgan ikki holat uchun ro`yxat yarmini tashlab yuborishimiz mumkin.

Qidirilayotgan qiymat o`rta qiymatdan kichkina bo`lganda, agar u ro`yxatda mavjud bo`lsa, u ro`yxatning birinchi yarmida bo`ladi. Agar u o`rta elementdan katta va ro`yxatda mavjud bo`lsa, u ro`yxatning ikkinchi yarmida bo`ladi. YAgona tekshirishda ro`yxat yarmini tashlab yuborishimiz uchun shuning o`zi yetarli.Bu jarayonni qaytarish natijasida qolgan yarim ro`yxatning yarmini tashlab yuborishimiz mumkin. Natijada biz quyidagi algoritmga ega bo`lamiz:

```
Binary
Search(list.target,N)
list \\ ko`rib chiqish uchun ro`yxat
```

```
target      \\ maqsadli qiymat
N     \\ ro`yxatdagi elementlar soni
start=1
end=N
while start<=end do
middle=(start+end)/2
select(Compare(list[middle],target))  from
case -1:   start=middle+1
case  0:  return middle
case 1:   end=middle-l
end select
end while
return 0
```

Agar qidirilayotgan qiymat topilgan o`rta qiymatdan oshib ketsa, start o`zgaruvchisiga middle o`zgaruvchisidan 1 ta ko`p bo`lgan qiymat beriladi.Agar qidirilayotgan qiymat topilgan o`rta qiymatdan kichik bo`lsa, end o`zgaruvchisiga middle dan 1 ta kam qiymat beriladi. Bittaga surish shuni bildiradiki, solishtirish natijasida biz o`rta qiymat qidirilayotgan qiymatligini aniqlaymiz, shuning uchun uni tahlil qilishdan chiqarishimiz mumkin.

Takrorlanish (sikl) har doim ham to`xtaydimi? Agar qidirilayotgan qiymat topilsa, return operatori ishlab turganligi uchun ham javob albatta ma’qul bo`ladi. Kerakli qiymat topilmagan taqdirda, har bir takrorlanish iteratsiyasida yoki start qiymati oshadi yoki end qiymati kamayadi. Bu shuni ko`rsatadiki, qiymatlar doimiy yaqinlashadi. Ma’lum bir vaqtga yetganda bu ikki qiymat tenglashadi va takrorlanish yana bir marta start=end=middle shartni tekshirish bilan qaytariladi. Bu o`tishdan keyin (agar element bu index bilan qidirilayotgan element bo`lmasa) yoki start qiymati middle va end dan bittaga katta, yoki teskarisi, end o`zgaruvchisi qiymati middle va start dan bittaga kam bo`ladi. Ikkala holda ham while takrorlanishga qo`yilgan shart yolg`on va takrorlanish boshqa bajarilmaydi. Shuning uchun takrorlanish har doim tugaydi.

Bu algoritm to`g`ri qiymatni beradimi? Agar qidirilayotgan qiymat topilsa, javob shartsiz maqbuldir, chunki return operatori bajarildi. Agar ro`yhatning qolgan yarim qismining o`rta qiymati to`g`ri kelmaydigan bo`lsa, har bir

takrorlanishda qolgan yarim elementlar tashlab yuboriladi, chunki ular juda katta yoki juda kichik.

Yuqorida aytilganidek, natijada biz solishtirishimiz kerak bo`lgan yagona elementga kelamiz. Agar bu bizga kerak kalit bo`lsa, u holda middle o`zgaruvchisi uni qaytaradi.

Agar kalit qiymati qidirilayotgan qiymatdan farqli bo`lsa, start o`zgaruvchisi qiymati end qiymatidan oshadi yoki teskarisi - end qiymati start nikidan kamayadi.

Agar qidirilayotgan qiymat ro`yxatda bo`lganda, u middle qiymatidan kichik yoki katta bo`lardi. Ammo start va end ko`rsatadiki, oldingi solishtirish qolgan barcha imkoniyatlarni cheklaydi va shuning uchun qidirilayotgan qiymat ro`yhatda yo`q.

Demak takrorlanish yakunlanadi, qaytariladigan qiymat esa nolga teng bo`ladi, bundan ko`rinadiki bo`lishlarni ketma-ket tarzda bajarish lozim. Ro`yxat hajmidan qat'iy nazar ketma-ket bo`lish (va kerak vaqtda ko`rilmaydigan bo`lakni tashlab yuborish) da biz ro`yxatdan bitta elementga kelamiz. SHunday qilib, algoritm to`g`ri javob qaytaradi.

Algoritm bir necha marotaba ro`yxatni teng ikkiga bo`lishi uchun tasavvur qilamizki, ma'lum k lar uchun $N = 2^k - 1$. Agar bu to`g`ri bo`lsa, ikkinchi bo`linishda qancha element qoladi? Uchinchida-chi? Umuman aytganda, aniqki, bir qancha bo`linishdan keyin takrorlanish $2^j$-1 elementli ro`yxat bilan davom etadi, 2^{j-1}- 1 ta element ro`yhatning birinchi yarmida, $2^{j-1}$- 1 qismi esa ikkinchi yarmida. SHuning uchun keyingi o`tishda 2^{j-1}- 1 ta element $1 \leq j \leq k$ qoladi. Bu holat so`nggi tahlilni osonlashtiradi, lekin keyingi misollardan bunga hojat yo`qligini ko`rasiz.

2. Saralash algoritmlari.

2.1. Qo`yishlar bilan saralash.

Qo`yishlar bilan saralashning asosiy g`oyasi shundaki, yangi elementni saralangan ro`yxatga qo`shishda uni ixtiyoriy joyga emas, balki kerakli joyga joylashtirish lozim, so`ng butun ro`yxatni boshqatdan saralash kerak. Qo`yishlar bilan saralashda ixtiyoriy ro`yxatning birinchi elementi saralangan ro`yxatning 1

uzunligi deb hisoblanadi. 2 elementli saralangan ro`yxat boshlang`ich ro`yxatdagi 2-elementni 1-element joylashgan ro`yxatning kerakli o`rniga joylashtirish orqali yaratiladi. Endi boshlang`ich ro`yxatning 3-elementini saralangan 2 elementli ro`yxatga qo`yish mumkin. Bu jarayon boshlang`ich ro`yxatning barcha elementlari kengayuvchi saralangan ro`yxatga joylashgunga qadar davom etadi.

Bu jarayonni ifodalovchi algoritm quyida keltirilgan:

```
InsertionSort(list,N)
list
N
For i=2 to N do
        newElement=list[i]
        location=i-1
        while(location>=1) and (list[location]>newElement) do
                list[location=1]=list[location]
                location=location-1
        end while
list[location+1]=newElement
end for
```

Bu algoritm yangi qo`yiladigan elementni newElement o`zgaruvchisiga yuklaydi. So`ng barcha katta elementlarni 1 pozitsiyaga surib, bu elementga joy ajratadi. Siklning oxirgi iteratsiyasi elementni location+1 nomer bilan location+2 pozitsiyaga o`tkazadi.

2.2. Pufaksimon saralash.

Endi pufaksimon saralashga o`tamiz. Bu usulning asosiy prinsipi kichik qiymatlarni surish natijasida ro`yxatning yuqori qismiga o`tkazish va shu vaqtning o`zida katta elementlarni pastga tushurishdan iborat. Pufaksimon saralashning turli variantlari bor. Bu paragrafda variantlardan birini ko`rib chiqamiz, qolganlarini topshiriqlarda uchratishingiz mumkin.

Pufaksimon saralash algoritmi ro`yxat bo`yicha bir nechta o`tish yo`llarini amalga oshiradi. Har bir o`tishda qo`shni elementlar taqqoslanadi. Agar qo`shni elementlarning tartibi noto`g`ri bo`lsa, u holda ularning joyi almashadi. Har bir o`tish ro`yxat boshidan boshlanadi. Dastlab birinchi va ikkinchi element taqqoslanadi, so`ng 2- va 3-element, keyin 3- va 4-element taqqoslanadi va hokazo; noto`g`ri tartibda turgan elementlar joy almashadi. Birinchi o`tishda

ro`yxatning eng katta elementi topilsa, u barcha elementlar bilan ro`yxat oxirigacha o`rin almashadi. 2-o`tishda ro`yxatning kattaligi bo`yicha 2-elementi oxiridan ikkinchi o`ringa o`tadi. Bu jarayonning davom etishi natijasida har bir element o`z o`rnini egallaydi. Bunda kichik qiymatlar ham yuqoridan o`z o`rniga joylashadi. Agar biror o`tishda elementlarning o`rin almashishi bajarilmasa, u holda barcha elementlar kerakli tartibda joylashgan bo`ladi va algoritmning bajarilishi to`xtatiladi. Shuni ta'kidlash kerakki,har bir o`tishda bir vaqtning o`zida bir nechta element o`z o`rni tomon siljib boradi. Pufaksimon saralash algoritmi quyida keltirilgan:

```
BubbleSort(list,N)
list
N
number0fPairs=N
swappedElements=true
while swappedElements do
    number0fPairs=number0fPairs-1
    swappedElements=false
    for i=1 to number0fPairs do
        if list[i]>list[i+1]then
            Swap(list[i],list[i+1])
            swappedElements=true
        end if
    end for
end while
```

2.3. Shell saralashi.

Shell saralashini Donald L. Shell o`ylab topgan. Bu saralashning o`ziga xosligi shundaki, butun ro`yxat aralashtirilgan ro`yxatostilarning majmuasi sifatida qaraladi. 1-qadamda bu ro`yxat ostilar juft elementlarni ifodalaydi. 2-qadamda har bir guruhda to`rttadan element mavjud bo`ladi. Jarayonning takrorlanib borishi natijasida har bir ro`yxatostisida elementlar soni ortib boradi, ro`yxat ostilar soni esa, mos holda kamayadi. 3.1-rasmda 16 elementdan iborat ro`yxatni saralashda foydalanish mumkin bo`lgan ro`yxat ostilar tasvirlangan. 3.1(a) - rasmda 8 ta ro`yxatosti tasvirlangan, bunda har birida 2 tadan element, ya'ni 1-ro`yxatostisi 1- va 9-elementlarni o`z ichiga oladi, 2- ro`yxatostisi 2- va 10-elementlarni o`z ichiga oladi va hokazo. 3.1(b)rasmda har biri to`rttadan elementni o`z ichiga olgan 4 ta

ro`yxatostisini ko`rishimiz mumkin. Bunda 1- ro`yxatostisi 1-, 5-, 9- va 13-elementlarni o`z ichiga oladi. 2- ro`yxatostisi 2-, 6-, 10- va 14-elementlardan iborat. 3.1(v)-rasmda juft va toq elementlardan iborat 2 ta ro`yxatostisi tasvirlangan. 3.1(g)-rasmda yana bitta ro`yxatga qaytib kelamiz.

Ro`yxat ostilarni saralash 3.1 da o`rganilgan Qo`yishlar bilan saralashni qo`llash yordamida amalga oshiradi. Natijada biz quyidagi algoritmga ega bo`lamiz:

```
Shellsort(list,N)
...
End while
```

Increment o`zgaruvchisi ro`yxatosti element nomerlari orasidagi qadam kattaligini o`z ichiga oladi. Biz algoritmni ro`yxat elementining uzunligidan kichik bo`lgan, ikkining eng yuqori darajasidan bittaga kam bo`lgan qadamdan boshlaymiz. Shuning uchun 1000 elementdan iborat ro`yxat 1-qadamining qiymati 511 ga teng bo`ladi. Shuningdek, qadam qiymati ro`yxatostilar soniga teng bo`ladi. 1- ro`yxatostisining elementlari 1 va 1+increment nomerlariga ega; oxirgi ro`yxat ostining 1-elementi sifatida increment nomerli element xizmat qiladi.

While siklining oxirgi bajarilishida passes o`zgaruvchisining qiymati 1 ga teng bo`ladi, shuning uchun InsertionSort funksiyasini oxirgi chaqiruvda increment o`zgaruvchisining qiymati 1 ga teng bo`ladi.

Bu algoritmning tahlili ichki InsertionSort algoritmining tahliliga asoslanadi. 2.1 da hisoblaganimizdek, Qo`yishlar bilan saralashning yomon holatida N ta elementdan iborat ro`yxat $(N^2-N)/2$ ta amalni talab etadi, o`rta holda esa N^2 elementni talab qiladi.

II. Amaliy qism

Ketma-ket izlash algoritmlarining yomon holat tahlili.

Ketma-ket izlash algoritmlarida ikkita yomon shartlashgan holat mavjud. Birinchi holat, qidirilayotgan element ro`yxat oxirida. Ikkinchisi, u umuman ro`yxatda bo`lmagan holat hisoblanadi. Bu holatlarga qancha solishtirishlar

bajarilishini ko`ramiz. Biz ro`yxatdagi barcha kalit so`zlar unikal deb tasavvur qilgandek, shu sababli agar so`nggi elementda bir xillik uchrasa oldin qilingan barcha solishtirishlar natijasiz bo`lgan. Ammo algoritm oxirgi elementga bormaguncha bu solishtirishlarni davom ettiradi. Natijada N ta solishtirish bajarilgan bo`ladi, bunda N-ro`yxat elementlari soni.

Qidirilayotgan element ro`yxatda mavjud emasligini tekshirish uchun uni barcha element bilan solishtirib chiqishga to`g`ri keladi.Agar biz biror-bir elementni tashlab ketsak, qidirilayotgan elementni ro`yxatda umuman yo`qligini yoki uni tashlab ketganligimizni aniqlay olmaymiz. Bu shuni ko`rsatadiki, biror-bir element qidirilayotgan element emasligini aniqlash uchun N ta solishtirish talab qilinadi.

Shuning uchun qidirilayotgan element ro`yxat oxirida yoki ro`yxatda umuman yo`qligini bilish uchun N ta solishtirish bajarish kerak. Ma’lumki N istalgan izlash algoritmining yuqori murakkablik chegarasini beradi, agar solishtirishlar N tadan ortiq bo`lsa, u holda bu holat qaysidir element bilan solishtirishlar kamida ikki marotaba bo`lganini, demak ortiqcha ish qilingani va algoritmni yaxshilash mumkinligini bildiradi.

Yuqori murakkablik chegarasi bilan murakkablik tushunchalari o`rtasida yomon shartlashgan holatlarda farq bor.

Yuqori chegara masala uchun kerakli, yomon holat tushunchasi uchun esa aniq algoritmda hal qiluvchi ahamiyatga ega. Keyingi bo`limda biz yomon shartlashgan holat yuqori chegara N dan kam bo`lgan izlash algoritmlari bilan tanishamiz.

Ketma-ket izlash algoritmlarining o`rta holat tahlili.

Izlash algoritmlarida ikkita o`rta holat bo`lishi mumkin. Birinchisida qidiruv har doim muvaffaqiyatli yakunlanadi, ikkinchisida qidirilayotgan qiymat ro`yxatda mavjud bo`lmaydi. Agar qidirilayotgan qiymat ro`yxatda bo`lsa, u holda N solishtiruvli vaziyatni o`z ichiga oladi: u birinchi, ikkinchi, uchinchi, to`rtinchi va

hokozo bo`lishi mumkin. Biz bunday barcha holatlar mumkin deb tasavvur qilamiz, bunda har bir holat 1/N ga teng.

Quyidagi savollarga javob beramiz.

1) *Agar qidirilayotgan qiymat birinchi o`rinda bo`lsa, nechta solishtirish bajarish talab qilinadi?*
2) *Ikkinchi o`rinda bo`lsachi?*
3) *Agar uchinchida?*
4) *Agar N ta elementdan so`nggisi bo`lsachi?*

Agar siz algoritmga diqqat bilan qaragan bo`lsangiz, unda javoblar 1,2,3 va N ko`rinishda bo`lishini aniqlashingiz qiyin emas. Bundan ko`rinadiki, har bir N holat uchun solishtirishlar soni qidirilayotgan qiymat o`rniga teng. Natijada murakkabligi o`rtacha hollar uchun

$$A(N) = \frac{1}{N}\sum_{i=1}^{N} i$$

$= \frac{1}{N} \cdot \frac{N(N+1)}{2}$ tenglik hosil qilamiz

$$\frac{N+1}{2}.$$

Tasavvur qilamiz, qidirilayotgan qiymat ro`yxatda uchramasligi mumkin, bunda imkoniyatlar soni N+1 ga o`sadi. Ko`rgandikki, qiymat ro`yxatda bo`lmaganda uni izlash N ta solishtirishni talab qiladi. Agar barcha N+1 imkoniyatlar bo`lishi mumkin deb qaralsa, quyidagi kelib chiqadi:

$$A(N) = \frac{1}{N+1}\left(\sum_{i=1}^{N} i + N\right)$$

$$= \frac{1}{N+1}\sum_{i=1}^{N} i + \frac{1}{N+1} \cdot N$$

$$= \frac{1}{N+1} \cdot \frac{N(N+1)}{2} + \frac{N}{N+1}$$

$$\approx \frac{N+2}{2}$$

(Qachon N juda katta bo`lganda, 1/(N+1) 0 ga intiladi.)

Ko`rinib turibdiki, elementni ro`yxatda mavjud emasligini hisobga olganda o`rtacha murakkablik bor yo`g`i ½ ga oshadi. Ro`yxat uzunligi ulkan bo`lgan holatda ro`yxat uzunligi bilan bu qiymat solishtirilganda u sezilarli kichikdir.

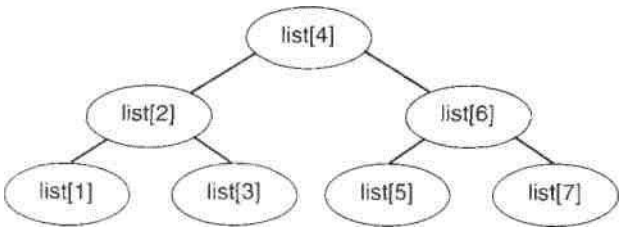
Ikkilanma izlash algoritmlarining yomon holat tahlili.

Oldingi abzatsda biz ro`yxatni ikkiga bo`lish ko`rsatgichi bir qancha takrorlanishlardan keyin birga kamayishini ko`rsatgan edik. Oxirgi takrorlanish iteratsiyasi bo`laklar kattaligi 1 ga teng bo`lganda bajarilishini ham ko`rgan edik, bu esa j=1 ($2^1$-1=1 bo`lganligi uchun) bo`lganda bajariladi. Bu shuni bildiradiki, N=$2^k$-1 dan ( o`tishlar soni k) oshmaydi. Oxirgi tenglik k ga bog`liqligini bilgan holda, yomon shartlashgan vaziyat o`tishlar soni $k= log_2(N+l)$ ga teng bo`lganda bajariladi.

Izlash jarayoni tahlilida daraxt yordam berishi mumkin. Daraxt shoxlarida ma’lum o`tishda tekshiriladigan elementlar turadi. Agar qidirilayotgan qiymat ayni qiymatdan kichkina bo`lsa chap shoxda, katta bo`lsa o`ng shoxda bo`ladi.

7 ta elementli ro`yxatning daraxt yechimi 1-rasmda ko`rsatilgan. Umumiy holatda biz ro`yxat har xil qismlarining o`rtasini tanlashga harakat qilamiz, daraxt bu holatda shartli muvozanatlidir.

Agar biz $N = 2^k$—1 deb olsak, bu holatdagi daraxt yechimi har doim to`liq bo`ladi. Unda k ta qism bo`ladi, bunda $k = \log_2(N+ 1)$. Biz har bir qismda bittadan solishtirish bajaramiz, shuning uchun barcha solishtirishlar soni $\log_2(N+1)$ dan oshmaydi.



1-rasm. *Ro`yxatdagi 7 ta elementdan izlashning daraxt yechimi*

Ikkilanma izlash algoritmining o`rta holat tahlili.

Ketma-ket qidiruv holati kabi o`rta holat tahlilida ham ikkita vaziyatga qaraymiz. Birinchi holatda qidirilayotgan qiymat ro`yxatda aniq mavjud, ikkinchi holatda esa qidirilayotgan qiymat ro`yxatda mavjud bo`lmasligi mumkin. Birinchi holatda qidirilayotgan qiymat N ta imkoniyatli vaziyatda. Biz bu barcha holatlarni bir xil 1/N imkoniyatli deb hisoblaymiz. Agar izlashning binar daraxt usulini

qaraydigan bo`lsak, 1 elementli qismida faqat 1 ta solishtirish, 2 elementli qismida 2 ta solishtirish, 3 elementli qismida 3 ta solishtirish bo`ladi. Umuman, I elementli qismda i ta solishtirish talab qilinadi. 7.3.2 da i qism ta 2^{i-1} shox va $N=2^k-1$ da daraxt k qismliligi ko`rsatilgan. Bu shuni ma'lum qiladiki, barcha solishtirishlar sonini quyidagi formula bilan topish mumkin.

$$A(N)=\frac{1}{N}\sum_{i=1}^{k} i2^{i-1} \qquad (N=2^k-1 \text{ учун})$$

$$=\frac{1}{N}\cdot\frac{1}{2}\sum_{i=1}^{k} i2^{i}$$

(7.19) dan foydalanib quyidagini hosil qilamiz:

$$A(N)=\frac{1}{N}\cdot\frac{1}{2}((k-1)2^{k+1}+2)$$

$$=\frac{1}{N}((k-1)2^{k}+1))$$

$$=\frac{1}{N}(k2^{k}-2^{k}+1)$$

$$=\frac{k2^{k}-(2^{k}-1)}{N}$$

$$=\frac{k2^{k}}{N}-1$$

$N = 2^k-1$ bo`lganda, $2^k = N+1$ o`rinli. Shuning uchun

$$A(N)=\frac{k(N+1)}{N}-1$$

$$=\frac{kN+k}{N}-1$$

N ning oshishi bilan k/N nolga yaqinlashadi va

$A(N)\sim$ k-1($N= 2^k$ -1 uchun);

$A(N)\sim\log_2(N+1)-1$.

Endi ikkinchi holatga qaraymiz, bunda qidirilayotgan qiymat ro`yxatda yo`q. Elementning vaziyatlari soni oldingidek N ga teng, bundan tashqari N+1 – vaziyat qidirilayotgan elementni ro`yhatdan tashqari solishtirish ham bor.

Vaziyatlar soni N+1 teng, chunki qidirilayotgan element ro`yhatning birinchi elementidan kichkina bo`lishi, irinchidan katta ikkinchidan kichkina va hokazo, oxiri N dan katta bo`lishi mumkin. Har bir bu hollar uchun qidirilayotgan element ro`yxatda mavjud bo`lmagan holatda k tadan solishtirish mavjud. Barcha imkoniyatlar 2 N+1 ta.

Bundan

$$A(N)=\frac{1}{2N+1}\left(\sum_{i=1}^{k} i2^{i-1}+(N+1)k\right)$$ Nq2^k-1 uchun

Yuqoridagi tenglikdan:

$$A(N)=\frac{((k-1)2^k+1)+(N+1)k}{2N+1}$$
$$=\frac{((k-1)2^k+1)+(2^k-1+1)k}{2(2^k-1)+1}$$
$$=\frac{(k2^k-2^k+1)+2^k k}{2^{k+1}-1}$$
$$\approx\frac{k2^{k+1}-2^k+1}{2^{k+1}}$$
$$\approx k-\frac{1}{2}=\log_2(N+1)-\frac{1}{2}$$ N=2^k-1 uchun

Oxirgi qiymat holat natijasini sezilarsiz darajada oshiradi. Misol uchun 2^{20}—1= 1 048 575 ta elementdan tashkil topgan ro`yxatdan birinchi holatda 19, ikkinchi holatda esa 19,5 natijani oldik.

Qo`yishlar bilan saralash algoritmlarining yomon holat tahlili

Agar ichki while siklini ko`rib chiqadigan bo`lsak, yana boshqatdan qo`shilayotgan element barcha elementlarda kichik va ro`yxatning saralangan qismida joylashgan bo`lsa, jarayonning katta qismi bajariladi. Bu holatda location o`zgaruvchisining qiymati 0 ga teng bo`lganda sikl to`xtaydi. Shuning uchun algoritm bajarilishining asosiy qismi har bir yangi element ro`yxat boshiga qo`shilganda amalga oshiriladi. Bu faqatgina boshlang`ich ro`yxat elementlari kamayish tartibda bo`lganda bajariladi. Bu eng yomon hodisalardan biri, lekin boshqalari ham bor.

Bunday ro`yxatni qayta ishlashni qanday amalga oshirish kerakligini ko`rib chiqamiz. Dastlab ro`yxatning 2-elementi qo`yiladi. U faqatgina bitta element bilan taqqoslanadi. Ikkinchi qo`yilgan element oldingi ikkita element bilan, uchinchi qo`yiladigan element uchta element bilan va hokazo, i-qo`yiladigan element i ta element bilan taqqoslanadi hamda bu jarayon N-1 marta takrorlanadi. Shunday qilib, eng yomon holatdagi saralash murakkabligi quyidagiga teng:

$$W(N) = \sum_{i=1}^{N-1} i = \frac{(N-1)N}{2} = \frac{N^2 - N}{2}$$

$$W(N) = O(N^2)$$

Qo`yishlar bilan saralash algoritmlarining o`rta holat tahlili

O`rta hol tahlilini 2 bosqichga ajratamiz. Dastlab navbatdagi element holatini aniqlash uchun kerak bo`ladigan taqqoslashlarning o`rta qiymatini hisoblaymiz. Keyin, 1-qadam natijasidan foydalanib, barcha kerakli amallarning o`rta qiymatini hisoblash mumkin. Dastlab, i-elementning joylashish o`rnini aniqlash uchun taqqoslashlarning o`rta qiymatini hisoblaymiz. YUqorida aytib o`tganimizdek, ro`yxatga i-elementni qo`shganda u kerakli joyda joylashgan bo`lsa ham, hech bo`lmaganda bitta taqqoslash bajariladi.

i-element uchun nechta mumkin bo`lgan holat mavjud? Qisqa ro`yxatlarni ko`rib chiqamiz va ixtiyoriy holatda natijalarni umumlashtirishga harakat qilamiz. 1-qo`shiladigan element uchun 2 ta imkoniyat bor: ikki elementdan birinchisi yoki ikkinchisi bo`lishi mumkin. 2-qo`shiladigan elementning 3 holatdan birida bo`lishi mumkin: 1,2,3 raqamlari bilan. Demak, i-qo`shiladigan element i+1 ta holatdan birini egallashi mumkin. Barcha imkoniyatlar ehtimollik darajasini teng deb olamiz.

Har bir i+1 pozitsiyaga yetib olish uchun nechta taqqoslash talab etiladi? i ning kichik qiymatlarini ko`rib chiqamiz va natijani birlashtirishga harakat qilamiz. Agar 4-qo`shilayotgan element 5-pozitsiyaga tushsa, u holda taqqoslash yolg`on bo`ladi. Agar uning 4-pozitsiyasi to`g`ri bo`lsa, u holda 1-taqqoslash rost hisoblanadi, ikkinchisi esa – yolg`on. 3-pozitssiyaga tushsa, 1- va 2-taqqoslash rost, 3-si esa yolg`on bo`ladi. 2-pozitsiyada 1-,2- va 3-taqqoslash rost va 4-si yolg`on bo`ladi. 1-pozitsiyada barcha taqqoslashlar rost bo`ladi, yolg`on taqqoslashlar bo`lmaydi. Bundan esa i+1, i, i-1,…, 2 pozitsiyalarga tushuvchi i-element uchun taqqoslashlar mos holda 1,2,3 …i ekanligi kelib chiqadi. 1-pozitsiyaga tushganda esa taqqoslashlar soni i ga teng bo`ladi. i- qo`yiladigan element uchun taqqoslashlarning o`rta qiymati quyidagi tenglik bilan beriladi:

$$A_i = \frac{1}{i+1}\left(\sum_{p=1}^{i} p + i\right) = \frac{1}{i+1}\left(\frac{i(i+1)}{2} + i\right) = \frac{i}{2} + \frac{i}{i+1} = \frac{i}{2} + 1 - \frac{1}{i+1}$$

Biz i-elementni qo`yishdagi taqqoslashlarning o`rta qiymatini hisobladik. Endi bu natijani ro`yxatdagi har bir N-1 element uchun yig`ish kerak:

$$A(N) = \sum_{i=1}^{N-1} A_i = \sum_{i=1}^{N-1} (\frac{i}{2} + 1 - \frac{1}{i+1})$$

$$A(N) = \sum_{i=1}^{N-1} \frac{i}{2} + \sum_{i=1}^{N-1} 1 - \sum_{i=1}^{N-1} \frac{1}{i+1}$$

$$\sum_{i=1}^{N-1} \frac{i}{2} = \sum_{i=1}^{N-1} 1 = \sum_{i=1}^{N-1} \frac{1}{i+1}$$

$$A(N) \approx \frac{1}{2} \frac{(N-1)N}{2} + (N-1) - (\ln N - 1) = \frac{N^2 - N}{4} + (N-1) - (\ln N - 1)$$

$$= \frac{N^2 + 3N - 4}{4} - (\ln N - 1) \approx \frac{N^2}{4};$$

$$A(N) = O(N^2).$$

Pufaksimon saralash algoritmlarining yaxshi holat tahlili.

SwappedElements bayroq izohi noto`g`ri ekanligini ta'kidlash uchun yaxshi holat tahlilini qisqacha ko`rib chiqamiz. Bajarilayotgan ish hajmi qaysi holatda minimal bo`lishini ko`rib chiqamiz. For sikli 1-o`tishda to`liq bajarilishi kerak va shuning uchun algoritmga kamida N-1 ta taqqoslash talab etiladi. 2 ta imkoniyatni ko`rib chiqish kerak: 1-o`tishda hech bo`lmaganda bitta o`rin almashtirish yoki o`rin almashtirish bajarilmaydi. 1-holatda o`rin almashtirish SwappedElements bayrog`ining qiymati true ga o`zgarishiga olib keladi, demak while sikli yana takrorlanadi, bunda N-2 ta taqqoslash talab etiladi. 2-holatda SwappedElements element bayrog`i false qiymatini saqlab qoladi va algoritm bajarilishi to`xtatiladi.

Shuning uchun yaxshi holda N-1 ta taqqoslash bajariladi, bunda 1-o`tishda o`rin almashtirishlar bo`lmaydi. Bundan esa ma'lumotlarning yaxshi to`plami talab qilingan tartibda elementlar ro`yxatini tashkil etishi kelib chiqadi.

Pufaksimon saralash algoritmlarining yomon holat tahlili.

Yaxshi holda ro`yxatdagi elementlar talab qilingan taribda joylashar ekan, elementlarning teskari tartibda joylashishi yomon holni bermasmikan, degan savol tug`iladi. Agar eng katta element 1-o`rinda tursa, u holda u ro`yxat oxirigacha qolgan barcha elementlar bilan yonma-yon qo`yib boriladi.1-o`tishdan oldin katta

qiymatdagi 2-element 2-holatni egallaydi, biroq 1-taqqoslash va 1-o`rin almashtirish natijasida u 1-o`ringa o`tkaziladi. 2-o`tishning boshida 1-holatda katta qiymatdagi 2-element joylashgan bo`ladi va u ro`yxatning oxiridan 2-elementgacha qolgan elementlar bilan yonma-yon o`rin almashib boradi. Bu jarayon barcha qolgan elementlar uchun bajariladi, shuning uchun for sikli N-1 marta takrorlanadi. SHuning uchun berilganlarning teskari tartibi yomon holga olib keladi.

Yomon holda nechta taqqoslash amalga oshiriladi? 1-o`tishda qo`shni elementlarning N-1 ta taqqoslashi, 2-o`tishda esa N-2 ta taqqoslash bajariladi. Tadqiqotlar shuni ko`rsatadiki, har bir navbatdagi o`tishda taqqoslashlar soni bittaga kamayadi. Shuning uchun yomon hol murakkabligi quyidagi formula bilan beriladi:

$$W(N) = \sum_{i=N-1}^{1} i = \sum_{i=1}^{N-1} i = \frac{(N-1)N}{2} = \frac{N^2 - N}{2} \approx \frac{1}{2} N^2 = O(N^2).$$

Pufaksimon saralash algoritmlarining o`rta holat tahlili.

Yuqorida ta’kidlaganimizdek, yomon holatda for sikli N-1 marta takrorlanadi. O`rta holda elementlarning o`rnini almashtirmasdan hosil bo`lgan o`tishlarning ixtiyoriysida ehtimolligi teng deb faraz qilamiz. Har bir holatda nechta taqqoslash bajarilishini ko`rib chiqamiz. 1-o`tishdan keyin taqqoslash soni N-1 g teng bo`ladi. 2 ta o`tishdan keyin N-1+N-2 ta bo`ladi. birinchi i ta o`tishda taqqoslashlar sonini S(i) deb belgilaymiz. Agar birorta ham o`rin almashtirish bajarilmasa, algoritm o`z ishini to`xtatadi. O`rta hol tahlilida bunday imkoniyatlarni ko`rib chiqish lozim. Natijada quyidagi tenglikka ega bo`lamiz:

$$A(N) = \frac{1}{N-1} \sum_{i=1}^{N-1} C(i)$$

bu yerda S(i) – for siklining dastlabki i ta o`tish oralig`idagi taqqoslashlar soni. Bunga nechta taqqoslash talab etiladi? S(i) ning qiymati quyidagi tenglikda ko`rsatilgan:

$$C(i) = \sum_{j=N-1}^{i} j = \sum_{j=i}^{N-1} j = \sum_{i=i}^{N-1} j - \sum_{j=i}^{i-1} j$$

yoki

$$C(i) = \frac{(N-1)N}{2} - \frac{(i-1)i}{2} = \frac{N^2 - N - i^2 + i}{2}$$

Oxirgi tenglikni A(N) ifodaga qo`yib quyidagini hosil qilamiz:

$$A(N) = \frac{1}{N-1}\sum_{i=1}^{N-1}\frac{N^2 - N - i^2 + i}{2}$$

N i ga bog`liq bo`lmaganligi uchun:

$$A(N) = \frac{1}{N-1}((N-1)\frac{N^2 - N}{2} + \sum_{i=1}^{N-1}\frac{-i^2 + i}{2})$$

$$A(N) = \frac{N^2 - N}{2} + \frac{1}{2(N-1)}(\sum_{i=1}^{N-1}(-i^2) + \sum_{i=1}^{N-1} i)$$

Natijada:

$$A(N) = \frac{N^2 - N}{2} + \frac{1}{2(N-1)}(-\frac{(N-1)N(2N-1)}{6} + \frac{(N-1)N}{2})$$

$$= \frac{N^2 - N}{2} - \frac{(2N-1)N}{12} + \frac{N}{4} = \frac{6N^2 - 6N - 2N^2 + N + 3N}{12}$$

$$= \frac{4N^2 - 2N}{12} \approx \frac{1}{3}N^2 = O(N^2)$$

MAVZU BO`YICHA TOPSHIRIQLAR

1. Ketma-ket izlashning o`rta holat bo`yicha murakkabligi maqsadli qiymat 0.25 ga teng bo`lganda qanday bo`ladi. Agar uning qiymati 0.75 ga teng bo`lsa, ro`yxatning birinchi yarmida bo`lish ehtimoli qanday bo`ladi?
2. Ikkilanma izlash algoritmi uchun 12 elementli ro`yxatning yechim daraxtini quring. Daraxtning ichki tarmoqlari tekshirilayotgan elementlar bilan belgilangan bo`lishi, daraxtning chap tarmog`i maqsadli qiymat tekshirilayotgan elementdan kichik bo`lgan holni, o`ng tomoni esa katta bo`lgan holni ko`rsatishi kerak.
3. InsertionSort algoritmining har bir o`tishidan keyin [7, 3, 9, 4, 2, 5, 6, 1, 8] ro`yxatning holatini yozing.
4. BubbleSort algoritmining har bir o`tishidan keyin [3, 5, 2, 9, 8, 1, 6, 4, 7] ro`yxat holatini tasvirlang.
5. ShellSort algoritmining 8, 4, 2 va 1 qadamlarida [16, 15,14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1] ro`yxatning barcha o`tishidan keyingi holatini yozing. Nechta taqqoslash amalga oshirildi?

9 - AMALIY MASHG`ULOT: Sonli algoritmlar: ko`phadlarning qiymatlarini hisoblash va matritsalar ustida amallar

Ishning maqsadi: Ba'zi sonli algoritmlarning murakkabligi tahlili. Ko`phadlarning qiymatlarini hisoblashda va matritsalar ustida amallar bajarish uchun mo`ljallangan algoritmlarning bajariladigan amallar soniga nisbatan mukammalligini tekshirish.

I. Nazariy qism

1. Ko`phadlarning qiymatini hisoblash.

Umumiy ko`rinishda yozilgan quyidagi ko`phadni qaraymiz:

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0 .$$

Faraz qilaylik $a_n, a_{n-1}, \ldots, a_0$ koeffitsientlar ma'lum, doimiy va massivga yozilgan bo`lsin. Bu shuni anglatadiki, ko`phadlarni hisoblash uchun yagona kiruvchi ma'lumot sifatida x hisoblanadi, dastur natijasi esa x nuqtasidagi ko`phadlarning qiymati hisoblanadi.

Hisoblashning standart algoritmi chiziqlidir, ya'ni:

```
Evaluate(x)
x // nuqta, ko`phadning qiymatini hisoblash uchun
result=a[0] + a[1]*x
xPower=x
for i=2 to n do
xPower=xPower*x
result=result+a[i]*xPower
end for
return result
```

Bu algoritm ancha sodda va uning tahlili aniq. For siklida ikkita ko`paytirish amali mavjud bo`lib, jami amallar soni $n-1$ ga teng. Bundan tashqari bir ko`paytirish sikldan oldin bajariladi. Shu sababli ko`paytirishlar umumiy soni $2n-1$. Siklda bitta qo`shish amali bajariladi. Yana bitta qo`shish amali sikldan oldin bajariladi. Shu sababli qo`shish amallarining umumiy soni n ga teng.

1.1. Gorner sxemasi.

Gorner sxemasi orqali hisoblash samaraliroq, chunki u murakkab emas. Bu sxema ko`phadning quyidagi ko`rinishiga asoslangan:

$$p(x) = ((\ldots((a_n x + a_{n-1})x + a_{n-2})x + \ldots + a_2)x + a_1)x + a_0 .$$

Ko`rinib turibdiki, ko`phadning bu ko`rinishi oldingi ko`phad bilan ekvivalent. Bu ko`rinishdagi ko`phadga mos algoritm quyidagi ko`rinishga ega :

HornersMethod(x)

x // nuqta, ko`phadning qiymatini hisoblash uchun

for i=n-1downto 0 do

*result=result*x*

result=result+a[i]

end for

return result

Sikl *n* marta bajariladi, bunda sikl ichida bitta ko`paytirish va qo`shish mavjud. Shu sababli Gorner sxemasi orqali hisoblashda *n* ta ko`paytirish va *n* ta qo`shish bajariladi – bu esa standart algoritmga nisbatan ikki marta kam.

1.2. Koeffitsiyentlarni dastlabki qayta ishlash.

Ko`phad koeffitsiyentlarini algoritm ishidan oldinroq qayta ishlash orqali natijani yaxshilash mumkin. Asosiy g`oya shundan iboratki, ko`phadni tasvirlash kichik darajali ko`phadlar orqali bo`ladi. Masalan, $x^{256}$ darajani hisoblash uchun paragraf boshidagi Evaluate funksiyasiga o`xshash siklli funksiyadan foydalanish mumkin. Natijada 255 ta ko`paytirish bajariladi. Muqobil yondashuv shundan iboratki, *result=x\*x* qo`yiladi, shundan keyin $result = result * result$ amali 7 marta bajariladi. Birinchi bajarishdan so`ng o`zgaruvchi x^4, ikkinchidan keyin x^8 ni, uchinchidan keyin x^{16}, to`rtinchidan keyin $x^{32}$ ni, beshinchidan keyin $x^{64}$, oltinchisidan keyin $x^{128}$ va yettinchisidan so`ng x^{256} ni tashkil etadi.

Koeffitsientlarni dastlabki qayta ishlash uslubi ishlashi uchun ko`phadlar unimodal (yani, koeffitsient $a_n = 1$ ) bo`lishi, ko`phad darajasi esa ikkining

qandaydir darajasidan 1 birlikka kam, yani ($n = 2^k - 1$, $k>1$) bo`lishi kerak. Bu holda ko`phadni quyidagi ko`rinishda tasvirlash mumkin:

$$p(x) = (x^j + b)\,q(x) + r(x), \text{ bunda } j = 2^{k-1}.$$

Har ikki q va r ko`phadlardagi hadlar $p$ ga qaraganda ikki marta kam. Kerakli natijani olish uchun $q(x)$ va $r(x)$ larni alohida hisoblab, keyin $p(x)$ ko`phadga qo`yib hisoblaymiz. Agar bunda *b* ning qiymatini to`g`ri tanlasak, har ikki *q* va *r* ko`phadlar unimodal bo`ladi. Har birining darajasi ularning har biriga ushbu protsedurani qo`llash imkoniyatini beradi. Bu protsedurani ketma-ket qo`llash orqali hisoblashni ancha tejash mumkin.

2. Matritsalarni ko`paytirish.

Matritsa – matematik ob'ekt bo`lib, ikkilik massivga ekvivalentdir. Matritsada sonlar qator va ustunlarda joylashadi. Ikkita bir xil matritsani elementar darajada qo`shish yoki ayirish mumkin. Agar birinchi matritsa ustuni sonlari ikkinchi matritsa qatori sonlari bilan to`g`ri kelsa, bu matritsalarni ko`paytirish mumkin. Natijada, matritsa qatori birinchi matritsa qatori va ustuni esa ikkinchi matritsa ustuniga teng bo`ladi. 3 x 4 hajmdagi matritsani 4 x 7 hajmdagi matritsaga ko`paytirsa, u holda biz 3 x 7 hajmdagi matritsaga ega bo`lamiz.

Matritsalarni ko`paytirish kommutativ emas: bir xil o`lchamli ikkita kvadrat matritsalarni *AB* va *BA* ko`rinishda ko`paytirish mumkin, ammo natija bir-biridan farq qiladi. Ta'kidlash lozimki, ikkita *a* va *b* sonlarni ko`paytirish kommutativdir, ya'ni $a \cdot b = b \cdot a$ o`rinli.

Ikki matritsani ko`paytirish yoki ayirish uchun har bir birinchi qator ko`phadlari va ikkinchi har bir matritsa ustuniga ko`paytiriladi. Keyin bunday ko`paytma yig`indisi hisoblanadi va natija to`g`ri kelgan katakka yoziladi. Rasmda ikki matritsani ko`paytirish keltirilgan (1- rasm).

Matritsani ko`paytirish 24 marta ko`paytirish 16 marta qo`shishni talab qiladi. Umuman standart hajmdagi *a x b* matritsani a x s matritsaga ko`paytirilsa, ab ko`paytirish va a(b-1) qo`shish bajariladi.

$$\begin{pmatrix} a\ b\ c \\ d\ e\ f \end{pmatrix}\begin{bmatrix} A\ B\ C\ D \\ E\ F\ G\ H \\ I\ J\ K\ l \end{bmatrix} =$$

$$= \begin{bmatrix} aA+bE+cI & aB+bF+cJ & aC+bG+cK & aD+bH+cL \\ dA+eE+fI & dB+eF+fJ & dC+eG+fK & dD+eH+fL \end{bmatrix}$$

1- rasm. *2* x *3 – matritsani 3* x *4 – matritsaga ko`paytirish.*

$a \times b$ o`lchamli G matritsani $b \times c$ o`lchamli H matritsaga ko`paytirishning standart algoritmi quyidagi ko`rinishda bo`ladi. Natija esa $a \times c$ o`lchamdagi R matritsaga yoziladi:

```
for i=1 to a do
        for j=1 to s do
        R[i,j]=0
        for k=l to b do
                R[i, j]>R[i, j]+G[i, k]*H[k, j]
                end for k
        end for j
end for i
```

Bir qarashda bu algoritm ikki matritsani ko`paytirish uchun zarur bolgan minimal hajmdagi ishdek tuyuladi. Ammo tadqiqotchilar bu algoritmning eng yaxshi ekanligini isbotlay olmadi, natijada matritsalarni ko`paytirish bo`yicha boshqa effektiv algoritmlar ham mavjud ekanligini ko`rsatishdi.

2.1. Matritsalarni Vinograd usulida ko`paytirish

Agar ikki matritsa ko`paytmalarining natijasiga qaraydigan bo`lsak, natija matritsasining har bir elementi ko`paytirilayotgan matritsalarning mos satr va ustun elementlarining skalyar ko`paytmasiga teng ekanligini ko`rish mumkin. Anglash mumkinki, matritsalarni ko`paytirishni amalga oshirishda oldindan tayyorgarlik ishlarini, ya’ni dastlabki qayta ishlashni tashkil qilish lozim.

Ikki vektorni qaraymiz: $\boldsymbol{V} = (v_1, v_2, v_3, v_4)$ va $\boldsymbol{W} = (w_1, w_2, w_3, w_4)$.

Ularnig skalyar ko`paytmasi

$\mathbf{V} \bullet \mathbf{W} = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$ ga teng.

Bu tenglikni quyidagi ko`rinishda yozishimiz mumkin:

$$\mathbf{V} \bullet \mathbf{W} = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_4) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4.$$

Oxirgi ikki tenglik ekvivalent ekanligini osongina tekshirib ko`rish mumkin. Balki ikkinchi ifoda birinchi ifodaga qaraganda ko`proq ishni talab qiladi: to`rtta ko`paytirish o`rniga biz oltita ekanligini, 3ta qo`shiluvchi o`rniga esa 10 tasini ko`ramiz. Bu ifodaning o`ng qismi dastlabki qayta ishlash imkonini beradi: uning o`ng qismini oldindan hisoblash va birinchi matritsaning satrlari uchun hamda ikkinchi matritsaning ustunlari uchun saqlab qo`yish kerak. Amaliyotda bu dastlabki qayta ishlangan elementlar ustida biz faqat ikki ko`paytirish va keyingi beshta qo`shishni hamda 2 ta qo`shimcha qo'shishni bajarishga to`g`ri keladi.

$a \times b$ o`lchamli G matritsani $b \times c$ o`lchamli H matritsaga ko`paytirishning Vinograd algoritmi quyidagi ko`rinishda bo`ladi. Natija esa $a \times c$ o`lchamdagi R matritsaga yoziladi:

```
d=b/2
// G ni hisoblash uchun rowFactors
for i=l to a do
        rowFactor[i]=G[i,1]*G[i,2]
        for j=2 to d do
                rowFactor[i]=rowFactor[i]+G[i,2j-1]*G[i,2j]
        end for j
end for i
// H ni hisoblash uchun columnFactors
for i=l to c do
        columnFactor[i]=H[1,i]*H[2,i]
        for j=2 to d do
                columnFactor[i]=columnFactor[i]+H[2j-1,i] *H[2j,i]
        end for j
end for i
// R matritsani hisoblash
for i=l to a do
        for j=l to s do
        R [i , j ] =-rowFactor [i] -columnFactor [ j ]
                for k=l to d do
                R[ i, j ]>R[ i, j ] + (G[ i, 2k-1]+H[ 2k, j ])*(G[ i, 2k ] + H[ 2k-l, j ])
                end for k
        end for j
end for i
```

```
// toq umumiy o`lcham holidagi hadlarni qo`shish
if (2*(b/2) /= b) then
        for i=l to a do
                for j=l to c do
                R[ i, j ]=R[ i, j ]+G[ i, b ]*H[ b, j ]
                end for j
        end for i
end if
```

2.2. Matritsalarini Shtrassen usulida ko`paytirish.

Shtrassen algoritmi kvadrat matritsalar bilan ishlaydi. Umuman olganda bu shunchalik samaraliki, ba'zi hollarda matritsani kvadrat matritsaga to`ldirish kerak bo`ladi, ya'ni qo`shimcha elementlarni kiritish hisobiga. Shu holda ham bu algoritm yutuq keltiradi.

Shtrassen algoritmida 2 x 2 matritsalarni ko`paytirish uchun 7 ta formuladan foydalanamiz. Bu formulalar tabiiy emas, afsuski, Shtrassen original maqolasida bu formulalar qanday kelib chiqqanligini tushuntirmagan. Yaxshi tomoni shundaki, na formulalarda va na ularni qo`llashda matritsaning elementlarini ko`paytirish kommutativligi talab qilinmaydi. Bu esa, xususiy holda bu elementlarning o`zlari ham matritsa bo`lishi muminligi, shuningdek, Shtrassen algoritmini rekursiv qo`llash mumkinligini anglatadi.

Bu esa Shtrassen formulasi:

$x_1 = (G_{1,1} + G_{2,2})(H_{1,1}+H_{2,2});$
$x2 = (G_{2,1} + G_{2,2})H_{1,1};$
$x_3 = G_{1,1}(H_{1,2}- H_{2,2});$
$x_4 = G_{2,2}(H_{2,1}— H_{1,1});$
$x_5 = (G_{1,1} + G_{2,2})H_{2,2};$
$x_6 = (G_{2,1}- G_{1,1})(H_{1,1} + H_{1,2});$
$x_7 = (G_{2,1}- S_{2,2})(H_{2,1}+H_{2,2}).$

Endi R matritsa elementlari quyidagi formulalar bilan hisoblanadi:

$R_{1,1} = x_1+x_4 - x_5+x_7;$
$R_{2,1}=x_2+x_4;$
$R_{1,2} = x_3 + x_5;$
$R_{2,2} = x_1+ x_3 - x_2+x_6.$

II. Amaliy qism

1. Ko`phadlarning qiymatini hisoblash algoritmlarining tahlili.

Umumiy ko`rinishdagi ko`phadlar to`g`risida so`z yuritmasdan, misolga e'tibor qaratamiz. Quyidagi ko`phadni qaraymiz:

$$p(x) = x^7 + 4x^6 - 8x^4 + 6x^3 + 9x^2 + 2x - 3.$$

Avval $x^j + b$ ko`paytuvchini aniqlaymiz. Ko`phad darajasi 7, ya'ni $2^3 - 1$, bundan esa, $k = 3$. Shundan kelib chiqadiki, $j = 2^2 = 4$. b ning qiymatini shunday tanlab olamizki, har ikki q va r ko`phadlar unimodal bo`lsin. Buning uchun $p(x)$ ko`phaddagi $a_{j-1}$ koeffitsiyentga qarash va uni $b = a_{j-1} - 1$ ga qo`yish kerak. Bizning holatda bu shuni anglatadiki, $b = a_3 - 1 = 5$. Endi biz

$$x^7 + 4x^6 - 8x^4 + 6x^3 + 9x^2 + 2x - 3 = (x^4 + 5)q(x) + r(x)$$

tenglikni qanoatlantiruvchi q va r qiymatlarni topishimiz kerak. q va r ko`phadlar $p(x)$ ni $x^4 + 5$ ga bo`lgandagi bo`linma va qoldiq bilan mos tushadi. Qoldiqli bo`lish

$$p(x) = (x^4 + 5)(x^3 + 4x^2 + 0 \cdot x + 8) + (x^3 - 11x^2 + 2x - 37)$$

ko`rinishni beradi. Keyingi qadamda biz xuddi shu protsedurani har bir $q$ va $r$ ko`phadlarga qo`llashimiz mumkin:

$$q(x) = (x^2 - 1)(x + 4) + (x + 12), \qquad r(x) = (x^2 + 1)(x - 11) + (x - 26).$$

Natijada esa quyidagiga ega bo`lamiz:

$$p(x) = (x^4 + 5)((x^2 - 1)(x + 4) + (x + 12)) + ((x^2 + 1)(x - 11) + (x - 26)).$$

Ushbu ko`phadga qarab, ko`rishimiz mumkinki, x^2 ni hisoblash uchun bitta ko`paytirish kerak bo`ladi; yana bir ko`paytirish $x^4 = x^2 \cdot x^2$ ni hisoblash uchun zarur. Bu ikki ko`paytirishlarda, ya'ni tenglamaning o`ng tomonini hisoblashda jami 3 ta ko`paytirish ishtirok etadi. Bundan tashqari, 10 ta qo`shish amali ham

bajariladi. Quyida keltirilgan 1-jadvalda ushbu hisoblash usuli va boshqa usullarning taqqoslash natijalari berilgan.

Ushbu protsedurani diqqat bilan o`rganib chiqib, umumiy murrakablik formulasini keltirib chiqarish mumkin. Avvalo shuni ko`ramizki, tenglamada bitta ko`paytirish va ikkita qo`shish amali qatnashgan. Shu sababli ko`paytirishlar soni *M=M(k)* va qo`shishlar soni *A = A(k)* uchun quyidagi rekurrent munosabatlarga ega bo`lamiz:

M(1)= 0 *A(1)*= 0

M(k)=2M(k-1)+1, agar *k* >1 *A(k)= 2A(k* - 1) + 2, agar *k* > 1.

Usul	**Ko`paytirish**	**Qo`shish**
Standart	13	7
Gorner sxemasi	7	7
Dastlabki qayta ishlash	5	10

1-jadval: *7-darajali ko`phad qiymatini hisoblashdagi amallar soni.*

Bu rekurrent munosabatlarni yechib, ko`paytirishlar soni taxminan $N/2$ ga, qo`shishlar soni esa $(3N-1)/2$ ga tengligini aniqlaymiz. Hali ko`paytma, ya'ni hisoblanishi zarur bo`lgan x^2, x^4, x^8,…, x^{2^k} qiymatlar ketma-ketligi qoldi va uning uchun qo`shimcha $k-1$ ta ko`paytirish amali talab etiladi. Shuning uchun ko`paytirish amallarining umumiy soni taxminan $N/2+\log_2 N$ ga teng.

Usul	**Ko`paytirish**	**Qo`shish**
Standart	$2N-1$	N
Gorner sxemasi	N	N
Dastlabki qayta ishlash	$\frac{N}{2}+\log_2 N$	$\frac{3N-1}{2}$

2-jadval: *N-darajali ko`phad qiymatini hisoblashdagi amallar soni.*

2-jadvalda standart algoritm, Gorner sxemasi va koeffitsiyentlarni datslabki qayta ishlash algoritmlarining qiyosiy tahlili keltirilgan. Oxirgi ikki algoritmni taqqoslaganda, ortiqcha $(N-1)/2$ qo`shish amallari hisobiga $N/2-\log_2 N$

ko`paytirishni tejadik. Barcha hisoblash tizimlarida ko`paytirishni qo`shishga almashtirish afzal hisoblanadi, shuning uchun oxirgi usul effektivlikni oshiradi.

1. Matritsalarni ko`paytirish algoritmlarining tahlili.

Vinograd algoritmining tahlili.

Umumiy rost hajmdagi *b* holatni jadval ko`rinishida ko`rib chiqamiz.

	Ko`paytirish	**Qo`shish**
G matritsani dastlabki qayta ishlash	*ad*	a(d-1)
N matritsani dastlabki qayta ishlash	*cd*	c(d-1)
R matritsani hisoblash	*acd*	*ac(2d+d+1)*
Umumiy	*(abc+ab+bc)/2*	*(a(b*-2)+*s(b*-2) +as(3b + 2))/2

Shtrassen algoritmining boshqa algoritmlariga nisbatan tahlili.

2 x 2 matritsani ko`paytmasida algoritm 7 ta ko`paytiruv va 18 ta qo`shish amalini bajaradi. Tejamkorlik ko`rinmaydi: Standart algoritmdagi bitta ko`paytirish bu algoritmdagi 14 ta qo`shishga almashtirildi. Umumiy holdagi tahlil shuni ko`rsatadiki, ikkita $N \times N$ matritsalarni ko`paytirishda ko`paytirish amallari soni taxminan $N^{2.81}$ ga, qo`shish amallari soni esa $6N^{2.81} - 6N^2$ ga teng. Ikkita 16 x 16 matritsaning ko`paytmasida Shtrassen algoritmi 9138 ta qo`shish amali hisobiga 1677 ta ko`paytirish amalini tejaydi.

Uchala algoritmni birgalikda qarab, quyidagi xulosaga ega bo`lamiz:

	Ko`paytirish	**Qo`shish**
Standart algoritmi	N^3	N^3-N^2
Vinograd algoritmi	$\frac{N^3 + 2N^2}{2}$	$\frac{3N^3 + 4N^2 - 4N}{2}$
Shtrassen algoritmi	$N^{2.81}$	$6N^{2.81}$-6 N^2

Shtrassen algoritmi amaliyotda kam qo`llaniladi: uni ishlatishda rekursiyahi ehtiyotkorlik bilan kuzatib boorish talab qilinadi. Uning muhim tomoni shundaki, u

matritsalarni ko`paytirishda $O(N^3)$ dan kam bo`lgan amallarni talab qiladigan birinchi algoritmdir. Matritsalarni ko`paytirish algoritmlarining samaradorligini oshirish va uning murakkabligini imkon qadar kamaytirish ustida ishlar davom ettirilmoqda.

MAVZU BO`YICHA TOPSHIRIQLAR

1. $x^7 + 2x^6 + 6x^5 + 3x^4 + 7x^3 + 5x + 4$ ko`phadni yoying:
   a) Gorner sxemasi bo`yicha;
   b) koeffitsiyentlarni dastlabki qayta ishlash usuli bo`yicha.
2. $x^7 + 6x^6 + 4x^4 - 2x^3 + 3x^2 - 7x + 5$ ko`phadni yoying:
   a) Gorner sxemasi bo`yicha;
   b) koeffitsiyentlarni dastlabki qayta ishlash usuli bo`yicha.
3. Vinograd algoritmi umumiy holdagi toq o`lchamli matritsalar uchun nechta ko`paytirish va qo`shish amalini bajaradi?
4. Shtrassen algoritmi ishini quyidagi matritsalar juftini ko`paytirish uchun qo`llang:

$$\begin{bmatrix} 1 & 9 \\ 7 & 3 \end{bmatrix} \text{ va } \begin{bmatrix} 5 & 2 \\ 4 & 11 \end{bmatrix}.$$

Olingan natijani standart algoritm bilan taqqoslang.

10 - AMALIY MASHG`ULOT: Determinallanmagan algoritmlar.

Ishning maqsadi: Determinallangan va determinallanmagan algoritmlar tahlili. P, NP va NP-to`liq masalalar sinfi. Ko`phadlarning qiymatlarini hisoblashda va matritsalar ustida amallar bajarish uchun mo`ljallangan algoritmlarning bajariladigan amallar soniga nisbatan mukammalligini tekshirish.

I. Nazariy qism

NP nima?

Avvalgi mashg’ulotlarimizda ko`zdan kechirilgan barcha algoritmlar polinomial murakkablikka ega edi. Undan tashqari, ularning hammasining

murakkabligi $O(N^3)$ edi, kompleks ko`paytirish algoritmi esa eng ko`p vaqt talab qiluvchi edi. Biroq, asosiysi, biz bu masalalarning aniq yechimini idrokli vaqt ichida topa olar edik. Bu masalalarning hammasi P sinfiga – polinomial murakkablikka ega masalalar sinfiga tegishli. Bunday masalalar deyarli yechiladigan hisoblanadi.

Yana boshqa masalalar sinfi ham bor: ular deyarli hal qilinmaydi va biz ularni idrokli vaqtda yecha oladigan algoritmlarni bilmaymiz. Bu masalalar NP sinfini hosil qiladi, ya'ni determinallanmagan polinomial murakkablikka ega. Faqat shuni aytish mumkinki, bu masalalarni yechuvchi barcha ma'lum determinallangan algoritmlar yoki eksponensial, yoki faktorial murakkablikka ega bo`ladi. Ulardan ba'zilarining murakkabligi $2^N$ ga teng, bu yerda $N$ – kiruvchi berilganlar soni. Bu holda kiruvchi berilganlar ro`yxatiga bitta elementni qo`shsak, algoritmning ishlash vaqti ikki barobar ortadi. Agar bunday masalani yechish uchun kiruvchi o`nta element uchun algoritmga 1024 ta amal talab qilingan bo`lsa, kiruvchi 11 ta element uchun amallar soni 2048 ni tashkil qiladi. Bu esa, kiruvchi berilganlarning soni kamgina oshirilganda vaqtning sezilarli ortishidir.

NP sinfidagi masalalarni xarakterlovchi «Determinallanmagan polinomial» so`z birikmasi uni yechishda quyidagi ikki qadamli yondashuv hisoblanadi. Birinchi qadamda bunday masalaning yechimini tasvirlovchi determinallanmagan algoritmi mavjud bo`lib, u yechimni topishga urinish deyish mumkin. Ba'zan bunday urinish muvaffaqiyatli bo`lib, biz optimal yoki optimalga yaqin yechimni topamiz, ba'zida esa muvaffaqiyatsiz (optimaldan javobdan ancha uzoq) yakunlanadi. Ikkinchi qadamda, birinchi qadamda olingan yechim qaralayotgan masalaning yechimi bo`la olishi tekshiriladi. Bu qadamlarning har biri alohida polinomial vaqtni talab qiladi. Muammo shundaki, izlangan yechimni olish uchun bu qadamlarni necha marta takrorlashimiz kerakligini biz bilmaymiz. Ikkala qadam polinomial bo`lsa ham, ularga murojaat ekspensial yoki faktorial bo`lishi mumkin.

Kommivoyajer masalasi NP sinfiga tegishli. Bizga shaharlar to`plami va ularning har bir ikkitasi orasidagi sayohat «narxi» berilgan. Shunday tartibni aniqlash kerakki, hamma shaharlarga tashrif buyurib, dastlabki shaharga qaytib

kelganda, sayohatning umumiy narxi minimal bo`lsin. Bu masalani, masalan, shahar ko`chalaridagi axlatni samarali yig`ishtirish tartibini aniqlash yoki kompyuter tarmog`ining hamma bo`g`inlarida axborotni tez tarqatish yo`lini tanlashda qo`llash mumkin. Sakkizta shaharni 40 320 mavjud usul bilan tartiblash mumkin, o`nta shahar uchun bu son 3 628 800 ga ortadi. Qisqa yo`lni tanlash ushbu hamma imkoniyatlarni ko`rib chiqishni talab qiladi. Faraz qilaylik, bizda ko`rsatilgan tartibda 15 ta shahar orqali sayohat bahosini hisoblovchi algoritm bor. Agar bir soniyada bunday algoritm o`zidan 100 ta variant o`tkazsa, hamma imkoniyatlar va qisqa yo`lni topish uchun unga to`rt asr kerak bo`ladi. Agar bizning xizmatimizda 400 ta kompyuter bo`lsa ham, baribir bunga bir yil ketadi, biz atigi 15 ta shaharni nazarda tutyapmiz. 20 ta shahar uchun qisqa yo`lni topish uchun milliard kompyuterlar 9 oy ichida parallel ravishda ishlashi lozim. Bizga tushunarliki, kompyuterlar optimal yechimni topishini kutgandan ko`ra, qanday bo`lsa ham tezroq va arzon sayohat qilish yo`lini topish lozim.

II. Amaliy qism

Grafni bo`yash masalasi

Yuqorida aytib o`tganimizdek, $G = (V, E)$ grafi uchlarni juft-jufti bilan bog`lovchi E qirralar to`plami, $V$ tugunlar yoki cho`qqilar to`plamidan iborat. Bu yerda biz faqat yo`naltirilmagan graflar bilan shug`ullanamiz. Grafning uchlarini butun sonlar bilan belgilanadigan turli ranglarga bo`yash mumkin. Bizni har bir qirraning uchi turli ranglarga bo`yalishi qiziqtiradi. N uchli grafda turli N ranglarga bo`yash mumkin, lekin bunda bo`yoq turini kam sonda ishlatish mumkinmi? Optimallashtirish masalasida bizni graf cho`qqisini bo`yash uchun zarur bo`ladigan bo`yoqlarning minimal miqdori qiziqtiradi. yechimni qabul qilish masalasida bizni cho`qqilarni s yoki kamroq bo`yoqlarga bo`yash mumkinligi qiziqtiradi.

Grafni bo`yash masalasida amaliy ilovalar bor. Agar grafning har bir cho`qqisi kollejda o`qiladigan kursni bildirsa va cho`qqilar qirralar bilan bog`lansa, agar ikkala kursni tanlovchi talaba bo`lsa, g`oyat murakkab graf hosil bo`ladi.

Agar har bir talaba 5 ta kurs tinglasa, bir talabaga 10 ta qirra to`g`ri keladi. 35000 talabaga 500 kurs to`g`ri keladi, deb faraz qilaylik. U holda hosil bo`lgan grafda 500 cho`qqi va 35000 qirra bo`ladi. Agar imtihonlarga 20 kun ajratilgan bo`lsa, talabaga bir kunda ikkita imtihon to`g`ri kelmasligi uchun graf cho`qqilarini 20 ta ranga bo`yash kerak bo`ladi.

Imtihonlar jadvalini tishlab chiqish graflarni bo`yashga ekvivalentdar. Ammo graflarni bo`yash masalasi NP sinfiga tegishli, shuning uchun jadvalni idrokli vaqtda to`g`ri ishlab chiqish mumkin emas. Bundan tashqari imtihonlarni rejalashtirganda, odatda, talabalarda bir kunda ikkitadan ortiq imtihon bo`lmasligi kerakligi talab etiladi, kursning turli bo`limlari bo`yicha imtihonlar esa bir kunga belgilanadi. Ko`rinib turibdiki, imtihonlarning «mukammal» rejasini ishlab chiqish mumkin emas, shuning uchun qulay rejalarni ishlab chiqish uchun boshqa texnika zarur.

Qutilarga joylashtirish masalasi

Bizda birlik hajmli bir nechta qutilar va turli o`lchamli $s_1, s_2, \ldots, s_N$ ob'ektlar to`plami mavjud bo`lsin. Optimallashtirish masalasida bizni barcha ob'ektlarni joylashtirish uchun zarur bo`lgan qutilarning eng kam soni, qaror qabul qilish masalasida esa barcha ob'ektlarni B yoki kam sondagi qutilarga joylash mumkinligi qiziqtiradi.

Bu masala axborotni diskka yoki kompyuterning bo`lakli xotirasiga yozishda, kemaga yukni samarali taqsimlashda, mijozlarning buyurtmasi bo`yicha materialni standart bo`laklarga kesishda paydo bo`ladi. Agar, masalan, bizda katta metal listlar va kichik listlarga buyurtmalar ro`yxati bo`lsa, buyurtmalarni iloji boricha yo`qotishlarsiz taqsimlashni hohlaymiz.

Ryukzakni joylashtirish masalasi

Bizda $s_1, s_2, \ldots, s_N$ hajmli $w_1, w_2, ..., w_N$ narxdagi ob'ektlar to`plami bor. Biz optimallashtirish malasida biz $K$ hajmli ryukzakni shunday joylashtirshimiz kerakki, uning bahosi maksimal bo`lsin. Qaror qabul qilish masalasida, bizni

joylashtirilgan ob’ektning jamlanma bahosi eng kamida W bo`lishiga erishish mumkinligi qiziqtiradi.

Bu masala pul qo`yish strategiyalarini tanlashda paydo bo`ladi: turli pul qo`yishlar bu yerda hajm, ko`zlangan foydaning miqdori – baho, ryukzakning hajmi rejalashtirilgan kapital qo`yilmalar hajmi bilan belgilanadi.

To`plamosti elementlarining yig`indisi masalasi.

Bizda turli o`lchamli $s_1, s_2, \ldots, s_N$ ob’ektlar to`plami va qandaydir musbat yuqori chegara L mavjud bo`lsin. Optimallashtirish masalasida biz shunday ob’ektlar to`plamini topishimiz kerakki, ularning o`lchamlari yig`indisi L ga yaqin bo`lib, bu yuqori chegaradan oshmasin. Qaror qabul qilish masalasida $L$ o`lchamlar yig`indisidan iborat ob’ektlar to`plami mavjudligi aniqlanishi kerak. Bu ryukzakni joylashtirish masalasining soddalashtirilgan ko`rinishidir.

KNF – ifodasining chin qiymatliligi haqidagi masala

Kon’yuktiv normal forma (KNF) o`zaro AND ($\wedge$ bilan belgilanadi) operatori bilan bog`langan Bul ifodalarining ketma-ketligidan tashkil topgan bo`lib, har bir ifoda OR ($\vee$ orqali belgilanadi) operatori bilan bog`langan Bul o`zgaruvchilarining bajarilishi yoki ularning inkoridan iborat bo`ladi. Quyida kon’yuktiv normal forma (inkor o`zgaruvchining tepasiga chiziqcha qo`yish bilan belgilanadi) sidagi Bul ifodasiga misol keltirilgan:

$$(a \vee b) \wedge (a \vee c) \wedge (a \vee b \vee c \vee d) \wedge (b \vee c \vee \bar{d}) \wedge (a \vee b \vee c \vee \bar{d} \vee e).$$

Kon’yuktiv normal formadagi Bul ifodasining chinligi masalasi faqat qaror qabul qilish variantida qo`yiladi: ifodaga kiruvchi o`zgaruvchilarda barcha ifodalarning qiymatlarini chinga aylantiruvchi o`rniga qo`yishlar mavjudmi? O`zgaruvchilar soni ham, ifodalarning murakkabligi ham cheklanmagan, shuning uchun chin qiymatlar kombinatsiyalari soni juda kata bo`lishi mumkin.

Ishlarni rejalashtirish masalasi

Bizda ishlar to`plami mavjud bo`lsin va bizga ularning har birining yakunlanishi uchun zarur bo`lgan $t_1, t_2, \ldots, t_N$ vaqt, bu ishlar tugashi kerak bo`lgan

$d_1, d_2, ..., d_N$ muddat, shuningdek, belgilangan muddatda tugallanmay qolgan har bir ish uchun $p_1, p_2, ..., p_N$ jarima ma'lum bo`lsin. Optimallashtirish masalasida solinadigan jarimalarni minimallashtiruvchi ishlar tartibi o`rnatilishi talab qilinadi. Qaror qabul qilish masalasida jarimaning miqdori **P** dan katta bo`lmagan ishlar tartibi mavjudligi so`raladi.

MAVZU BO`YICHA TOPSHIRIQLAR

1. NP sinfidagi masalalarni yechishning ikki bosqichli jarayoni bo`yicha quyidagi masala tahlil qilinsin. Javobda masalani yechishdagi barcha elementlarni o`zida mujassamlashtirgan determinallanmagan qadamlar tasvirlangan bo`lishi kerak. Masalan, kommivoyajer haqidagi masalada chiquvchi ma'lumotlar barcha shaharlarga tashrif ro`yxati tartibida tasvirlanishi lozim. Bundan tashqari, tekshirish jarayonini shunday tasvirlash kerakki, siz tomondan taklif qilingan variant haqiqatan ham masala yechimi ekanligi tasdiqlansin.
    a) Grafni bo`yash haqidagi masala;
    b) Qutilarga joylashtirish haqidagi masala;
    c) Ryukzakni qadoqlash haqidagi masala;
    d) To`plamosti elementlarining yig`indisi haqidagi masala;
    e) KNF-ifoda chinligi haqidagi masala
    f) Ishlarni rejalashtirish maasalasi.
2. Masalaning NP-to`liq ekanligini isbotlashda, bir masala boshqasiga akslantiriladi. Yuqorida ko`rib o`tilgan barcha masalalar NP-to`liq, shuning uchun ularning har biri boshqa ixtiyoriy masalaga keltirilishi mumkin. Quyidagi har bir masalalar juftini mos ravishda biridan ikkinchisiga keltirish akslantirishini yozing.
    a) Ryukzakni qadoqlash, qutilarga joylashtirish;
    b) Qutilarga joylashtirish, ishlarni rejalashtirish;
    c) Ishlarni rejalashtirish, to`plamosti elementlar yig`indisi;
    d) To`plamosti elementlar yig`indisi, kommivoyajer;
    e) Kommivoyajer, ishlarni rejalashtirish.

ADABIYOTLAR

1. Кнут Д. Искусство программирования для ЭВМ. т. 1-3. Москва. Мир. 1996-1998.
2. М. Гэри, Д.Джонсон. Вычислительные машины и труднорешаемые задачи. Москва, Мир, 1982.
3. Матрос Д.Ш., Поднебесова Г.Б. Теория алгоритм. Учебник для педагогического образования. М.: Бином. Лаборатория знаний, - 2008. - 202с.
4. Макконелл Дж. Основы современных алгоритмов. 2-доп.изд., М.: ТЕХНОСФЕРА, 366с., 2004.
5. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. Сер: Классические учебники: COMPUTER SCIENCE. М.: МЦНМО, – 960с., 2004.
6. Ахо А., Хопкрофт Дж. Построение и анализ вычислительных алгоритмов, М., Мир, 535 с., 1979.
7. Вирт Н. Алгоритмы и структуры данных. С примерами на Паскале. Санкт-Петербург, 352с., 2005.
8. Рейест Р. и др. Алгоритмы: построение и анализ. М., Мир, 1994.
9. Мальцев А.И. Алгоритмы и рекурсивные функции. М., Наука, 1968.
10. Носов В.А. Основы теории алгоритмов, анализа их сложности. Курс лекций. М., 139с., 1992.
11. Малышко В.В. Алгоритмы и алгоритмические языки. Конспект лекций для студентов Ташкентского филиала МГУ, 68с., 2006.
12. Пильщиков В.Н., Абрамов В.Г., Вылиток А.А., Горячая И.В. Машина Тьюринга и алгоритмы Маркова. Решение задач. (Учебно-методическое пособие) Московский государственный университет им. М.В.Ломоносова, Факультет вычислительной математики и кибернетики, Москва, 47с., 2006.
13. Yunusov A.S. Matematik mantiq va algoritmlar nazariyasi elementlari. Samarqand davlat universiteti nashriyoti, 182 b., 2012 y.
14. www.de.uspu.ru/Informatics/metodes/DPP/F/08/1/Index.htm.
15. http://www.krf.bsu.by/ELib/Genetic/GenAlg_2/index.htm.
16. http://www.neuroproject.ru.
17. http://www.aic.nrl.navy.mil/galist/.
18. http://iridia.ulb.ac.be/dorigo/ACO/ACO.html.
19. http://www.agp.ru/projects/.
20. http://www.swarm.org.

MUNDARIJA

"Algoritmlar nazariyasi" fanidan amaliy mashg'ulotlar uchun uslubiy qo`llanma

Ilmiy nashr

Alimjan Artikbayevich Ibragimov

O`zbek tilida