8-MA’RUZA. REKURSIYA VA ITERATSIYA. REKURSIV ALGORITMLARNING SAMARADORLIGI

Boshlang‘ich va oraliq ma’lumotlarni masalani yechish natijasiga aylantiradigan jarayonni bir qiymatli qilib, aniqlab beradigan qoidalarning biror bir chekli ketma-ketligi algoritm ekanligi yuqoridagi mavzulardan bizga ma’lum.

Buning mohiyati shundan iboratki, agar algoritm ishlab chiqilgan bo‘lsa, uni yechilayotgan masala bilan tanish bo‘lmagan biron bir ijrochiga, shu jumladan kompyuterga ham bajarish uchun topshirsa bo‘ladi va u algoritmning qoidalariga aniq rioya qilib masalani yechadi. Quyida pekursiv algoritmga to‘xtalamiz. Avvalo, rekursiv o‘zi nima degan savolga javob beramiz. Agar obyektning tarkibiy qismlari obyektning o‘zi orqali aniqlansa, u holda bu obyekt rekursiv deyiladi. Rekursiya nafaqat matematikada, balki kundalik hayotimizda ham ushrab turadi.

Rekursiya o‘z kuchini ayniqsa, matematik ta’riflarda namoyon etadi. Eng tanish misollar sifatida natural sonlar, daraxtsimon tuzilmalar va ba’zi funktsiyalarni keltirishimiz mumkin:

1. Natural sonlar:

a) 0 natural son hisoblanadi.

b) Natural sondan keyin keluvchi son natural hisoblanadi.

2. Daraxtsimon tuzilmalar:

a) ∅ daraxt hisoblanadi (va bo‘sh daraxt deyiladi).

b) Agar t1 va t2 –daraxtlar bo‘lsa, t1 va t2ning avlodlaridan iborat tugundan tashkil topgan tuzilma ham daraxt deyiladi (ikkilik yoki binar daraxt).

3. Faktorial funktsiya f(n):

f(0)=1

n>0 bo‘lganda f(n)///

Ko‘rinib turibdiki, rekursiyaning kuchi cheksiz obyektlar to‘plamini chekli mulohaza orqali aniqlash imkoniyatida namoyon bo‘ladi. Xuddi shunday chekli sonli hisoblashlarni chekli dastur orqali tavsiflash mumkin. Bunda dastur oshkor tsikllarni o‘z ichiga olmasligi ham mumkin. Ammo rekursiv algoritmlar, avvalo,

yechilayotgan masala, hisoblanayotgan funktsiya yoki qayta ishlanayotgan ma’lumotlar tuzilmasi rekursiv ravishda berilgan holda o‘rinlidir.

Umumiy holda, R rekursiv dastur (R ni o‘z ishiga olmagan) S ko‘rsatmalarning va R ni o‘zining ketma-ketligidan iborat R birlashma (kompozitsiya) sifatida ifodalanishi mumkin:

R=...

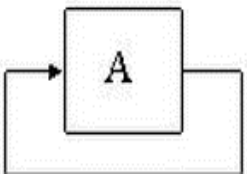
Dasturlarning rekursiv ifodasi uchun zaruriy va yetarli vosita bo‘lib protseduralar xizmat qiladi. Bu protseduralar ko‘rsatmalar to‘plamiga nom berish imkonini yaratib, bu nom orqali dastur bajarilish jarayonida ko‘rsatmalar chaqirilishi mumkin.

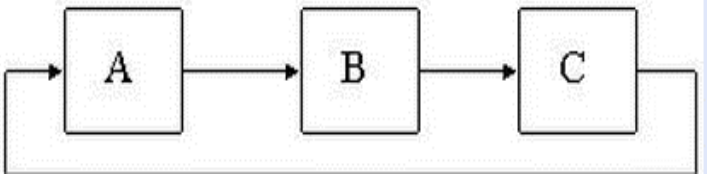
Agar R protsedura oshkor holda o‘ziga murojaatni o‘z ichiga olsa, bunday protsedura oshkor rekursiv deyiladi. Agar R protsedura R ga to‘g‘ridan-to‘g‘ri yoki bilvosita murojaatni o‘z ichiga olgan boshqa bir Q protsedurani tarkibiga olsa, u holda R bilvosita rekursiv deyiladi.

Rekursiv protseduralar cheksiz hisoblashlarga yo‘l oshganligini hisobga olsak, uni to‘xtatish muammosini ham hal etishimiz zarur. Fundamental talablar shundan iboratki, qaysidir vaqt momentiga kelib, bajarilmay qoladigan shunday V shart bajarilgandagina R rekursiv protseduraga murojaatlar amalga oshirilishi kerak.

Ma’lumki, har qanday protsedura yoki funktsiya boshqa protsedura yoki funktsiyalarga murojaatni o‘z ichiga olishi mukin. Xususiy holda, agar bu murojaat protsedura yoki funktsiyaning o‘ziga murojaatdan iborat bo‘lsa, bu protsedura yoki funktsiya rekursiv deyiladi.

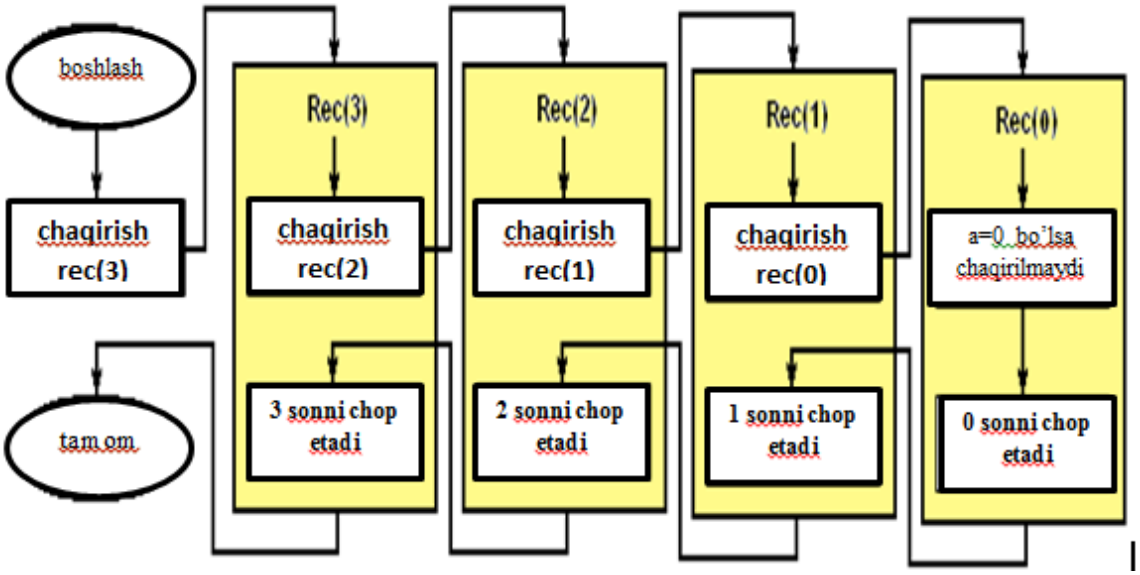
Umumiy holda, rekursiv prtsedura va funktsiyalarni quyidagi sxema ko‘rinishida ifodalashimiz mumkin:





Rec(3) ko‘rinishdagi murojaatda protsedura qanday ishlashini ko‘rib chiqaylik.

Quyidagi blok-sxemada operatorlarning bajarilish ketma-ketligi tasvirlangan:

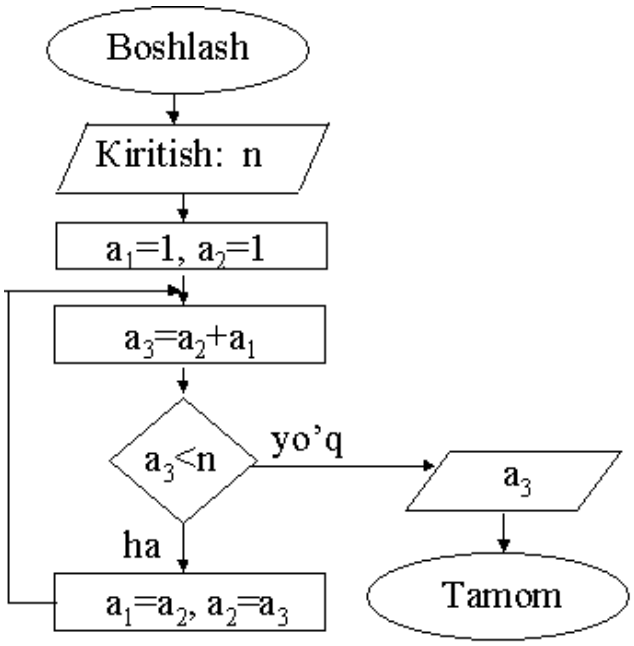


8.1-rasm. Boshlanish Rec(3)ga murojaat tugash.

8.1-misol: Rekursiv protsedura ishini tasvirlovchi blok-sxema tuzilsin. Res protsedurasi dastlab a=3 parametr bilan chaqiriladi, ya'ni Res(3) tarzidagi murojaat amalga oshiriladi. Uning tarkibida esa a=2 parametrli Res(2) tarzidagi murojaat mavjud. Hali dastlabki murojaat tugamasdan keyingi protseduraga murojaat tashkil etiladi, u tugamasa dastlabki Res(3) protsedura ishini tugatmaydi. Protseduraga murojaat a=0 ga davom etadi. Demak, bir vaqtning o‘zida protsedura 4 marta chaqirilib ishlaydi. Bir vaqtda bajariladigan protseduralar soni rekursiya chuqurligi deyiladi.

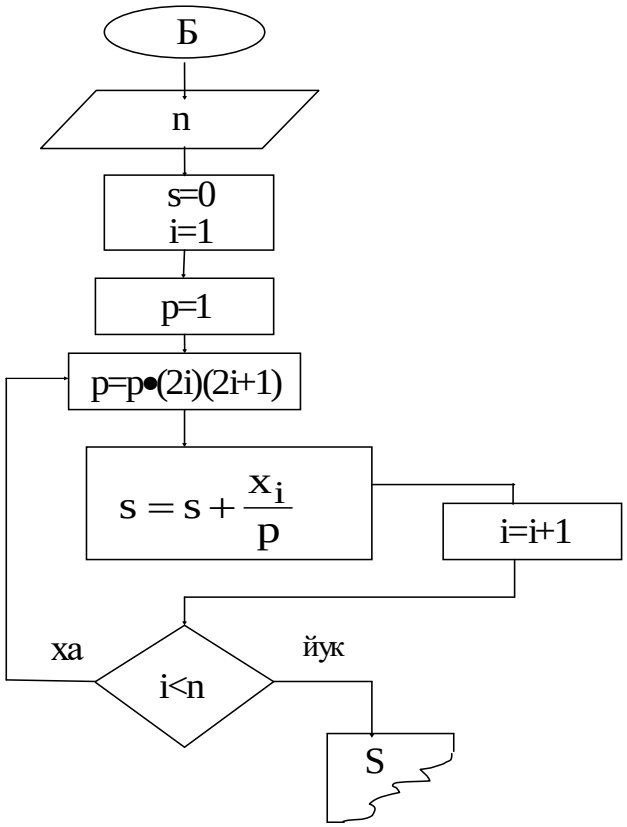
To‘rtinchi marta chaqirilgan Res(0) protsedura 0 sonini chop etadi va o‘z ishini yakunlaydi. Shundan so‘ng boshqaruv Res(1)ni chaqirgan protseduraga o‘tadi va 1 ni chop etadi. Xuddi shu tariqa barcha protseduralar tugaguncha jarayon davom etadi. Natijada to‘rtta 0, 1,2,3 sonlarini chop etiladi.

Fibonachi sonlarining *n*-hadini hisoblash algoritmi. Agar Fibonachi sonining nomerini ham aniqlash zarur bo‘lsa, birorta parametr-kalit kiritish kerak bo‘ladi.



8.2-rasm. Fibonachi sonlarining n- hadini hisoblash algoritmi.

8.2-misol. $S = \sum_{i=1}^{n} \frac{x_i}{(2i+i)!}$



“Taqsimla va boshqar” ko‘rinishidagi algoritmlar. “Taqsimla va boshqar” ko‘rinishidagi algoritmlar turli masalalarni yechish uchun ixcham va kuchli qurolni ta’minlaydi. Bu bo‘limda biz bunday algoritmlarni ishlab chiqish emas, balki ularning tahlili bilan shug‘ullanamiz. Sikllarda taqqoslashlar sonini hisoblashda siklning bir iteratsiyasidagi taqqoslashlar sonini hisoblab, uni iteratsiyalar soniga ko‘paytirish yetarli. Agar ichki siklning iteratsiyalar soni tashqi parametrlarga bog‘liq bo‘lsa, hisoblash qiyinroq bo‘ladi. “Taqsimla va boshqar” ko‘rinishidagi algoritmlar iteratsiyasini hisoblash oddiy emas: u rekursiv chaqiriqlar, tayyorlov va yakunlanuvchi harakatlarga bog‘liq. Rekursiv chaqiruvda u yoki bu funksiyaning necha marta bajarilishi, odatda, aniq emas. Misol sifatida quyidagi “taqsimla va boshqar” ko‘rinishidagi algoritmni ko‘rib chiqamiz:

```
DivideAndConquer (data, N, solution)data //kiruvchi ma’lumotlar to‘plami
N //to‘plamdagi qiymatlar soni
solution //masala yechimi
if (N <= SizeLimit) then
DirectSolution(data,N,solution) else
DivideInput(data,N,smallerSets,smallerSizes,numberSmaller) for i=1 to numberSmaller do
DivideAndConquer(smallerSets[i],smallerSizes[i],smallSol[i]) end for
CombineSolutions(smallSol.numberSmaller,solution)
end if
```

Bu algoritm yechimini oddiy rekursiv bo‘lmagan algoritm (matnda **DirectSolution** deb ataluvchi) yordamida topish uchun vazifaning hajmi kichik emasligini avval tekshiradi va agar shunday bo‘lsa, uning chaqirig‘i sodir bo‘ladi. Agar vazifa juda katta bo‘lsa, avval **Divide Input** amali chaqiriladi, u kiruvchi raqamlarni bir qancha kichik to‘plamlarga bo‘ladi (ularning soni **numberSmaller** ga teng). Kiruvchi dastlabki to‘plamning har bir qismi kichikroq to‘plamlardan biriga tushadi, ammo bir elementning o‘zi bir nechta to‘plamga tushishi mumkin. So‘ng har bir kichik kiruvchi to‘plamda **DivideAndConquer** algoritmining rekursiv

chaqiruvi sodir bo‘ladi, **CombineSolutions** funksiyasi olingan natijalarni birlashtiradi.

Natural sonning faktorialini siklda hisoblash qiyin emas, biroq quyida berilgan misolda bizga ushbu hisoblashning rekursiv varianti kerak bo‘ladi. JV sonining faktoriali *N* ni, *N*-1 sonining faktorialiga ko‘paytirilganiga teng. Shuning uchun quyidagi algoritm kerak bo‘ladi:

```
Factorial(N)
N
Factorial
if (N=1) then
return 1 else
smaller = N-1
answer=Factorial(smaller)
return (N*answer) end if
```

Bu algoritmning qadamlari oddiy va tushunarli va biz ularni yuqorida keltirilgan standart algoritm bilan solishtirishimiz mumkin. Biz avvalroq bu bo‘limda ko‘paytirish amali qo‘shishdan murakkabroq, deb eslatgan bo‘lsakda, ko‘paytirishni alohida hisoblash kerak. Misolni soddalashtirish uchun biz bu chegarani hisobga olmaymiz.

Bu ikki algoritmni taqqoslasak, raqamlar o‘lchamining chegarasi 1 ekanligi ko‘rinadi va bunda hech qanday arifmetik amal bajarilmaydi. Boshqa barcha hollarda biz **else** darajasiga o‘tamiz. Umumiy algoritmdagi birinchi qadam “kirishning kichik qismlarga bo‘linishi” bo‘ladi; hisoblash algoritmida bu qadamning faktorialiga bitta ayirishni talab qiluvchi **smaller** o‘zgaruvchisini hisoblash to‘g‘ri keladi. Umumiy va boshqar algoritmining keyingi qadami kichikroq raqamlarni qayta ishlash amalini rekursiv chaqirish hisoblanadi; faktorialni hisoblash algoritmida – bu bir rekursiv chaqiriq va undagi vazifaning hajmi avvalgidan bitta kam.

Umumiy algoritmdagi oxirgi qadam tenglamalarni birlashtirishdan iborat; faktorialni hisoblash algoritmida bu ko‘paytirish oxirgi **return** operatorida bajariladi.

Rekursiv algoritm samaradorligi. Rekursiv algoritm qanchalik samarador? Agar biz algoritmni hisoblash bevosita ikkinchi darajali, kiruvchi berilganlarning bo‘linishi logarifmik, yechimlarning birlashishi esa chiziqli (hammasi kiruvchi berilganlarning hajmiga bog‘liq) desak, kiruvchi berilganlar sakkiz qismga bo‘linadi va ularning har biri boshlang‘ich berilganlarning to‘rtdan bir qismiga teng ekanligini bilib uning samaradorligini ko‘rsata olamizmi? Bu masalani yechish oson emas, hatto unga qanday kirishish ham aniqmas. Biroq, “taqsimla va boshqar” ko‘rinishidagi algoritmlar tahlili juda sodda ekan, agar algoritmdagi qadamlar yuqorida ko‘rsatilgan umumiy holdagi algoritm qadamlariga mos bo‘lsa, bevosita hisoblash, kirishning bo‘linishi, rekursiv chaqiruvlarning ba’zi miqdorlari va hosil qilingan natijalarning birlashuvi, agar bu qadamlar bir-biri bilan qanday mos tushishini va har bir qadamning murakkabligi aniq bo‘lsa, u holda “taqsimla va boshqar” ko‘rinishidagi algoritm murakkabligini aniqlash uchun quyidagi formuladan foydalanish mumkin:

$$DAC(N)=\begin{cases} DIR(N), & agar \quad N \le SizeLimit, \\ DIV(N)+\sum_{i=1}^{numberSaller} DAC(smallerSize[i]+COM(N), & agar \quad N > SizeLimit \end{cases}$$

bu yerda **DAC** – **DivideAndConquer** algoritm murakkabligi**,**

DIR – Direct Solution algoritm murakkabligi,

DIV – Divide Input algoritm murakkabligi**,**

COM – CombineSolutions algoritm murakkabligi.

Bu umumiy formula asosida yuqorida qo‘yilgan savolga javob berish juda oson. Biz faqat umumiy formulaga har bir qismning ma’lum murakkabliklarini qo‘yib chiqishimiz kerak. Natijada quyidagiga erishamiz:

$$DAC(N)=\begin{cases} N^2, & agar\ N \le SizeLimit \\ \log_2 N+\sum_{i=1}^{8} DAC(N/4)+N, & agar\ N > SizeLimit \end{cases}$$

yoki, barcha kichik to‘plamlar hajmi bir xil bo‘lganligi uchun bundan ham osonroq usuli:

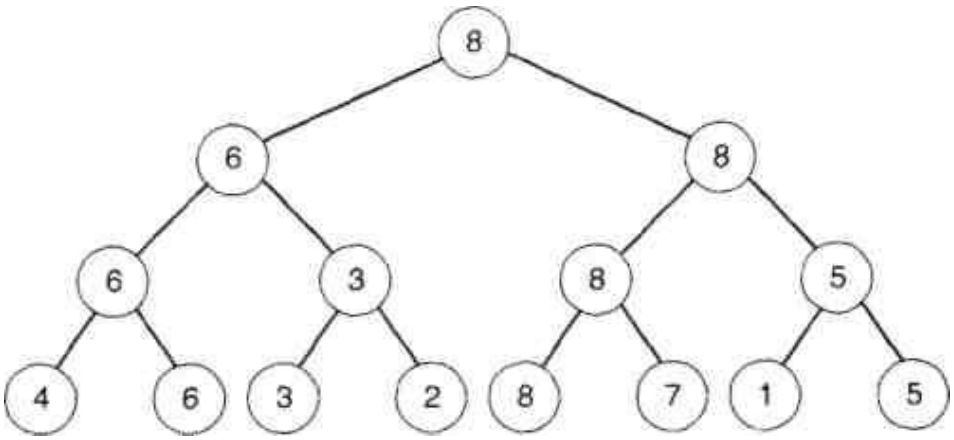
$$DAC(N) = \begin{cases} N^2, & agar\ N \le SizeLimit \\ \log_2 N + 8DAC(N/4) + N, & agar\ N > SizeLimit \end{cases}$$

Bunday ko‘rinishdagi tenglik rekurrent deb nomlanadi, chunki funksiya qiymati o‘z atamasida ifodalangan. Biz faqat *N* ga bog‘liq bo‘lgan va xuddi shu funksiyaning boshqa elementlariga bog‘iq bo‘lmagan murakkablik uchun ifodani topishimiz kerak.

Faktorial bilan bog‘liq misolga qaytamiz. Biz faktorial bilan hisoblanadigan algoritmning barcha bosqichlarini umumiy **DivideAndConquer** algoritmi bilan taqqosladik. Bu taqqoslashdan foydalanamiz va yuqorida keltirilgan umumiy formulaga qo‘yish kerak bo‘lgan qiymatni aniqlaymiz. **Factorial** funksiyasidagi bevosita hisoblashlar amallarni talab qilmaydi, kiruvchi berilganlarning bo‘linishi va natijalarning birlashuvi algoritmlarining har biri bittadan amalni talab etadi va rekursiv chaqiruv masalani yechadi, berilganlar hajmi boshlang‘ich berilganlardan bittaga kamayadi. Natijada **Factorial** funksiyasida hisoblashlar miqdori uchun quyidagi rekurrent munosabatni hosil qilamiz:

$$Calc(N) = \begin{cases} 0 & agarN = 1, \\ 1 + calc(N-1) + 1 & agarN > 1. \end{cases}$$

Turnirlar metodi. Turnirlar metodi rekursiyaga asoslangan va uning yordamida ko‘plab masalalarni yechish mumkin, ma’lumotlar bo‘yicha birinchi o‘tish natijasida olingan axborot keyingi proyutlarni yengillashtiradi. Agar biz eng katta qiymatni izlash uchun undan foydalansak, hamma elementlari bargli binar daraxtni tuzish talab qilinadi. Har bir bosqichda ikki element juftlarga birlashgan bo‘lib, ulardan eng kattasi otalik religa nusxalanadi. Jarayon ildiz tugunga yetguncha takrorlanadi. Qayd etilgan berilganlar to‘plami uchun to‘liq turnir daraxti 8.3-rasmda tasvirlangan.



8.3-rasm. Sakkiz qiymatdan iborat to‘plam uchun turnir daraxti.

Yuqorida eslatib o‘tilgan ediki, *N* tartibli taqqoslashlar miqdori hisobiga ro‘yxatdagi kattaligi bo‘yicha ikkinchi elementni topish algoritmi ishlab chiqiladi. Bunda bizga turnirlar metodi yordam beradi. Har qanday taqqoslash natijasida biz “g‘olib” va “mag‘lub”ga ega bo‘lamiz. Mag‘lublarni unutamiz, daraxt bo‘ylab yuqoriga faqat g‘oliblar harakat qiladi. Eng katta elementdan tashqari har qaysi element hech bo‘lmaganda bitta taqqoslashda mag‘lubiyatga uchraydi. Shuning uchun turnir daraxtini qurish uchun *N-1* ta taqqoslash talab etiladi.

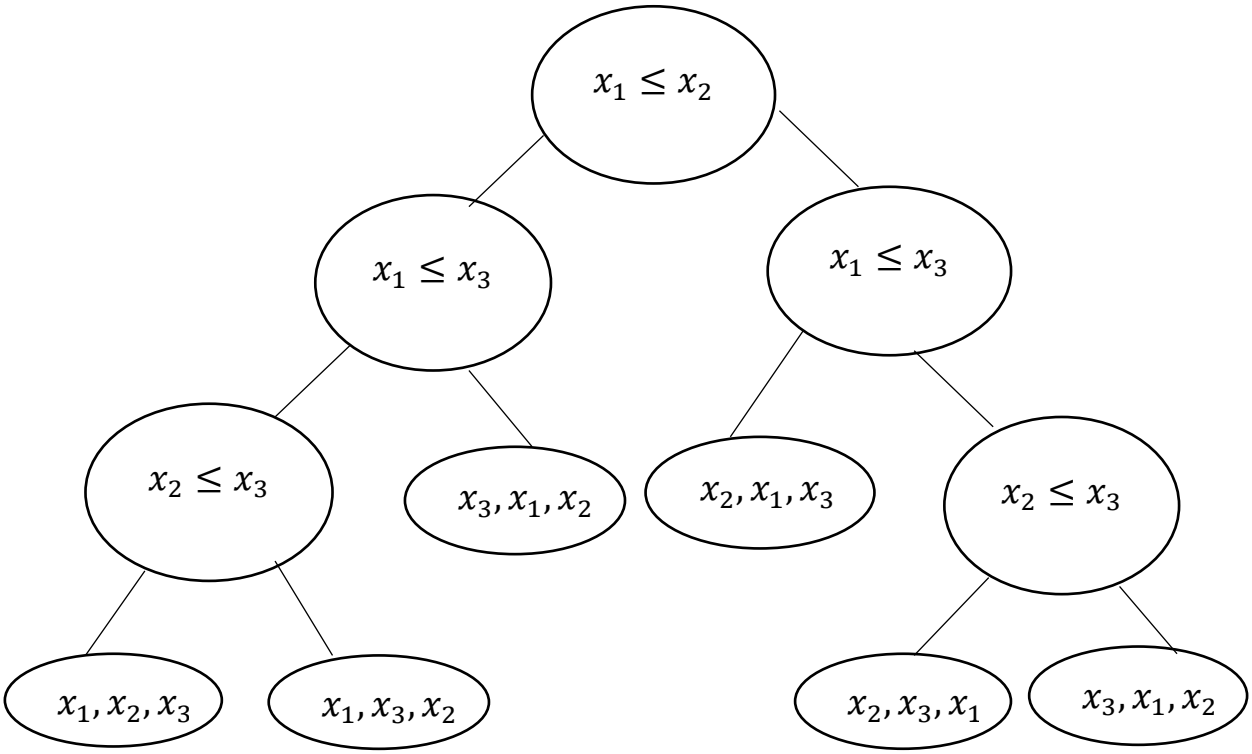
Kattaligi bo‘yicha ikkinchi element eng katta elementdan mag‘lub bo‘lishi mumkin. Daraxtdan pastga tomon yurish davomida eng katta elementga yutqazgan elementlar ro‘yxatini tuzamiz. Daraxt uchun formula bunday elementlar soni $[\log_2 N]$ dan ortib ketmasligini ko‘rsatadi. Ularning daraxt bo‘yicha qidiruvi $[\log_2 N]$ taqqoslashni talab qiladi; yana $[\log_2 N] - 1$ taqqoslashlar ulardan eng kattasini tanlash uchun kerak. Hammasi bo‘lib bu ishga $N + 2[\log_2 N] - 2$ ketadi, ya’ni $O(N)$ ta taqqoslash.

Turnirlar metodi yordamida qiymatlar ro‘yxatini saralash mumkin.

Quyi chegaralar. Algoritm, agar undan tezroq ishlaydigan algoritm mavjud bo‘lmasa, optimal hisoblanadi. Bizning algoritm optimal ekanligini qanday bilishimiz mumkin? Yoki optimal emas, faqatgina yetarlicha yaxshi bo‘lishi ham mumkin? Bu savollarga javob berish uchun biz aniq masalani yechish uchun kerak bo‘lgan amallar miqdorini eng kam holatda bo‘lsa ham bilishimiz kerak. Buning uchun masalani yechuvchi algoritmni emas, balki aynan masalani o‘rganish kerak. Natijada hosil qilingan quyi chegara bu masalani yechish uchun kerak bo‘lgan ish

hajmini ko‘rsatadi va bunga nomzod bo‘lgan tezroq ishlovchi ixtiyoriy algoritm ba’zi jarayonlarga noto‘g‘ri ishlov berishi shart.

Uchta sondan iborat ro‘yxatni saralash jarayonini tahlil qilish uchun yana binar daraxtidan foydalanamiz. Daraxtning ichki tugunlarini taqqoslangan elementlar juftligi bilan belgilaymiz. Daraxt novdasidan o‘tishi ta’minlangan elementlar ro‘yxati daraxtning mos bargida tasvirlanadi. Uch elementdan iborat ro‘yxat uchun daraxt 8.4-rasmda tasvirlangan. Bunday daraxt yechim daraxti deb nomlanadi.



8.4-rasm. Uch elementli ro‘yxatni saralash uchun yechim daraxti.

Har bir saralash algoritmi qaysi elementlarini taqqoslashiga bog‘liq holda o‘zining yechimlar daraxtini yaratadi. Yechimlar daraxtidagi ildizdan bargga borishning eng uzun yo‘li eng yomon holga mos keladi. Qisqa yo‘l eng yaxshi hol hisoblanadi. O‘rta hol yechimlar daraxtidagi qirralar sonini undagi barglar soniga bo‘lish bilan ifodalanadi. Bir qarashda yechimlar daraxtini chizish va kerakli sonlarni hisoblash qiyinmasdek tuyuladi. Lekin 10 ta soni saralashdagi yechimlar daraxtining hajmini tasavvur qiling. Ta’kidlanganidek, mumkin bo‘lgan tartiblar soni 3 628 800 ga teng. Shuning uchun daraxtda kamida 3 628 800 ta barg bo‘ladi,

undan ko‘proq ham bo‘lishi mumkin, chunki turli taqqoslashlar ketma-ketligi natijasida bitta tartibga takroriy kelish mumkin. Bunday daraxtda darajalar soni 22 dan kam bo‘lmasligi kerak.

Algoritm murakkabligi uchun yechimlar daraxti yordamida qanday qilib chegaralarni hosil qilish mumkin? Bilamizki, aniq algoritm elementlarning boshlang‘ich tartibiga bog‘liq bo‘lmagan holda ixtiyoriy ro‘yxatni tartiblashi kerak. Kiruvchi qiymatlarning ixtiyoriy o‘rin almashuvi uchun kamida bitta barg mavjud bo‘lishi kerak, bu esa yechimlar daraxtida barglar soni *N!* dan kam bo‘lmasligi kerakligini bildiradi. Agar algoritm haqiqatda samarali bo‘lsa, u holda undagi har bir o‘rin almashtirish bir marta uchraydi. *N!* bargli daraxtda nechta daraja bo‘lishi kerak? Biz har bir navbatdagi darajada oldingisiga qaraganda tugunlar ikki marotaba ko‘pligini ko‘rdik. *K* darajadagi tugunlar soni 2^{K-1} ga teng, shuning uchun yechimlar daraxtimizda *L* ta daraja bo‘ladi, bu yerda *L* — $N! \le 2^{L-1}$ uchun eng kichik son. Bu tengsizlikni logarifmlab, quyidagini hosil qilamiz:

$$log_2 N! \le L - 1.$$

L ning eng kichik qiymatini topish uchun logarifmdan qutulish mumkinmi? Faktorial xossalariga murojaat qilamiz. Quyidagidan foydalanamiz

$$Log_2 N! = log_2(N(N-1)(N-2)...\ 1),$$

Bu yerda (1.5) ga teng

$$log_2(N(N - 1)(JV - 2)...\ 1) = \log_2 N + \log_2 (N - 1) + \bullet\bullet\bullet + \log_2 1 = \sum_{i=1}^{N} \log_2 i .$$

(5.21) tenglikdan quyidagini hosil qilamiz:

$$\sum_{i=1}^{N} \log_2 i \approx N \log_2 N - 1.5, \log_2 (N!) \approx N \log_2 N$$

Bundan kelib chiqadiki, yechimlar daraxtidagi *L* ning minimal quyi chegarasi saralash uchun $0(N \log_2 N)$ tartibga ega. Endi biz bilamizki, $0(N \log N)$ tartibga ega bo‘lgan saralash algoritmi eng yaxshi hisoblanadi va uni optimal desa bo‘ladi.

Bundan tashqari, *0(N* log *N)* ta amaldan ko‘ra tezroq ishlaydigan algoritm to‘g‘ri ishlashi mumkinmas.

Saralash algoritmi uchun quyi chegarani chiqarishda algoritm ro‘yxat elementlari juftliklarini kema-ket taqqoslash asosida ishlaydi deb faraz qilingan edi. Bu algoritmda maqsadga erishish uchun kalit qiymatlari taqqoslanmaydi, balki ularni uyumlarga bo‘lib tashlaydi.

Rekurrent munosabat. Algoritm murakkabligi uchun rekkurent munosabatlar algoritm ko‘rinishidan chiqariladi, lekin ular yordamida bu murakkablikni darhol hal etish oson emas. Buning uchun rekurrent munosabatni rekurrent tabiatidan voz kechuvchi yopiq ko‘rinishga keltirish kerak. Bu usulni tushunish uchun bir qator misollar ko‘rib chiqamiz.

Rekurrent munosabat ikki ko‘rinishda berilishi mumkin. Birinchisi sodda holatlar kam bo‘lganda:

T(p) = 2T(n-2) - 15;

T(2) = 40;

T(1) = 40.

Ikkinchi shakli sodda holatlar miqdori nisbatan ko‘proq bo‘lgan holda ishlatiladi:

$$T(p) = \begin{cases} 4, & agar\, n \le 4 \\ 4T(n/2) - 1 & акс\ холда \end{cases}$$

Bu shakllar ekvivalent. Oddiy qiymatlar ro‘yxatini chiqarib tashlab, ikkinchisidan birinchisiga o‘tish mumkin. Bundan kelib chiqadiki, yuqorida keltirilgan rekurrent munosabatlardan ikkinchisini quyidagi ko‘rinishda yozish mumkin:

T(p) = 4T(n/2) - 1;

T(4) = 4;

T(3) = 4;

T(2) = 4;

T(1) = 4;

Quyidagi rekkurent munosabatni ko‘rib chiqamiz:

T(p) = 2T(n-2)-15;

T(2) = 40;

T(1) = 40.

Biz birinchi tenglikka *T(p*–2) uchun ekvivalent bo‘lgan ifodani qo‘ymoqchimiz. Buning uchun har bir *p* ni *p* – 2 bilan almashtiramiz:

T(p - 2) = 2T(p - 2 - 2) - 15 = 2T(n - 4) - 15.

Endi *T(p-4*) ni chiqarib tashlash kerak. Xuddi shunday almashtirishlarni bajarishimiz kerak. Buni katta bo‘lmagan qiymatlar ketma-ketligida bajaramiz:

T(p-2) = 2T(n-4)-15;

T(p-4) = 2T(n-6)-15;

T(p-6) = 2T(p-8)-15;

T(p-8) = 2T(n-10)-15;

T(p - 10) = 2T(n - 12) - 15.

Endi hisoblash natijalarini boshlang‘ich tenglamaga qo‘yamiz. Bunda natijani oxirigacha qisqartirmaymiz, aks holda umumiy sxemaga ega bo‘lmasligimiz mumkin. O‘rin almashtirishlar quyidagini beradi:

T(p) = 2T(n-2)-15 =2(2T(n- 4)-15)-15,

T(p) = 4T(n – 4) – 2•15 – 15;

T(p) = 4(2T(n – 6) – 15) – 2 •15 – 15,

T(p) = 8T(n – 6) – 4•15 – 2 •15 – 15;

T(p) = 8(2T(n – 8) – 15) – 4•15 – 2 •15 – 15,

T(p) = 16T(n – 8) – 8 • 15 – 4•15 – 2 • 15-15;

T(p) = 16(2T(n – 10) – 15) – 8•15 – 4 •15 – 2 •15 – 15,

T(p) = 32T(n – 10) – 16•15 – 8•15 – 4 •15 – 2 •15 – 1 5;

T(p) = 32(2T(n – 12) – 15) – 16•15 – 8 •15 – 4 •15 – 2 •15 – 15,

T(p) = 64T(n – 12) – 32•15 – 16•15 – 8 •15 – 4 •15 – 2 •15 – 15.

Balki siz umumiy sxemani tushunib yetgandirsiz. Avvalambor, har bir tenglikdagi qo‘shiluvchi ikkining navbatdagi darajasiga ko‘paytirilgan -15 ni ko‘rsatib turibdi. Ikkinchidan, koeffitsiyent T funksiyani rekursiv chaqiruvda ikkining darajasi bo‘lib hisoblanadi. Bundan tashqari, T funksiya argumenti har safar 2 taga kamayib bormoqda.

Bu jarayon qachon tugashini o‘zimizdan so‘rashimiz mumkin. Boshlang‘ich tenglikka qaytib T(1) va T(2) qiymatlarni qayd qilib olganimizni eslashimiz mumkin. Bu kattaliklardan biriga yetish uchun nechta almashtirish bajarish kerak? Juft p da 2=p-(n- 2) ga ega bo‘lamiz. Biz (n-2)/2-1 ta almashtirish bajarishimiz kerak, bu n/2-1 ta -15 ko‘paytuvchili qo‘shiluvchi va oldinda ikkining n/2-1 darajasini beradi. n=14 da nima bo‘lishini ko‘rib chiqamiz. Bu holda beshta almashtirish bajarishimiz kerak: 15 ko‘paytuvchisi bilan oltita qo‘shiluvchiga ega bo‘lamiz va T(2) da koeffitsiyent 2^6 ga teng bo‘ladi. Oxirgi tenglikda p=14 ta almashtirishda xuddi shunday natija hosil bo‘ladi.

Agar *p* toq son bo‘lsachi? Bu formulalar ilgarigidek ishlaydimi? *p*=13 bo‘lganda ko‘rib chiqamiz. Tenglikda bitta o‘zgarish sodir bo‘ladi —T funksiya argumenti 2 emas 1 bo‘ladi, lekin n/2-1 qiymati 5 ga teng bo‘ladi (6 ga emas). Toq n da n/2-1 o‘rniga n/2 ni olamiz. Javobda ikki holatni ko‘rib chiqishimiz kerak.

$$T(n) = 2^{2(n/2)-1} - T(2) - 15\sum_{i=0}^{(n/2)-1} 2^i, \text{ juft n da;}$$

$$T(n) = 2^{2(n/2)-1} \cdot 40 - 15\sum_{i=0}^{(n/2)-1} 2^i$$

$$T(n) = 2^{2(n/} T(2) - 15\sum_{i=0}^{(n/2)} 2^i,$$

toq *p* da.

$$T(n) = 2^{2(n/2)} \cdot 40 - 15\sum_{i=0}^{(n/2)} 2^i$$

Endi, juft n soni uchun T(n) formulani qo‘llab, quyidagiga ega bo‘lamiz

$T(p) = 2^{(p-2)-1} \cdot 40-15 \cdot (2^{n/2}-1) = 2^{n/2}20-2^{n/2} \cdot 30+15 = 2^{n/2}(20-15)+15 = 2^{n/2} \cdot 5+15,$

toq *p* da

$$T(p) = 2^{n/2} \cdot 40-15 \cdot (2^{n/2}-1) = 2^{n/2} \cdot 40-2^{n/2} \cdot 30+15 = 2^{n/2} \cdot 10+15.$$

Yana bir rekurrent munosabatni ko‘rib chiqamiz:

$$T(n) = \begin{cases} 5, & agar\ n \le 4 \\ 4T(n/2) - 1, & aks\ holda\ n = -1 \end{cases}$$

Yuqoridagi holdagidek ish yuritamiz. Avval faqat *p* qiymatlarni qo‘yamiz*;* bunda har bir navbatdagi tenglikda oldingi argumentning yarmi hosil bo‘ladi. Natijada quyidagi tengliklarga ega bo‘lamiz:

$T(n/2) = 4T(n/4) - 1;$

$T(n/4) = 4T(n/8) - 1;$

$T(n/8) = 4T(n/16) - 1;$

$T(n/16) = 4T(n/32) - 1;$

$T(n/32) = 4T(n/64) - 1.$

Bu tengliklarni boshlang‘ich tenglamaga qo‘ysak:

$T(n) = 4T(n/2) - 1 = 4(T(n/4) - 1) - 1,$

$T(n) = 16T(n/4) - 4 \cdot 1 - 1;$

$T(n) = 16(4T(n/8) - 1) - 4 \cdot 1 - 1,$

$T(n) = 64T(n/8) - 16 \cdot 1 - 4 \cdot 1 - 1;$

$T(n) = 64(T(n/16) - 1) - 16 \cdot 1 - 4 \cdot 1 - 1,$

$T(n) = 256T(n/16) - 64 \cdot 1 - 16 \cdot 1 - 4 \cdot 1 - 1;$

$T(n) = 256(4T(n/32) - 1) - 64 \cdot 1 - 16 \cdot 1 - 4 \cdot 1 - 1,$

$T(n) = 1024T(n/32) - 256 \cdot 1 - 64 \cdot 1 - 16 \cdot 1 - 4 \cdot 1 - 1;$

$T(n) = 1024(4T(n/64) - 1) - 256 \cdot 1 - 64 \cdot 1 - 16 \cdot 1 - 4 \cdot 1 - 1,$

$T(n)$ = 4096T(n/64) – 1024•1 – 256•1 – 64•1 – 16•1 – 4•1–1 .

Ko‘rinib turibdiki, chto pri -1da koeffitsiyent har bir almashtirishda biror to‘rt darajaga oshadi, argumentga bo‘layotgan ikkilik darajasi -1da eng katta to‘rt darajadan har safar 1ga ko‘p bo‘ladi. Bundan tashqari, T da koeffitsiyentdagi to‘rtning darajasi biz argumentni bo‘layotgan ikkining darajasi kabi $T(n/2^i)$ da bu koeffitsiyent 4^g ga teng, keyin esa -4^{i-1} dan -1gacha bo‘lgan qo‘shiluvchilar keladi. i ning qanday qiymatida almashtirishlar to‘xtatilishi kerak? Qiymatlar $p \le 4$ da berilganligi uchun, $T(4) = T(n/2^{1\log_2 n-2})$ ga yetib kelganda to‘xtashi mumkin. Natijada:

$$T(n) = 4^{\log_2 n-2} T(4) - \sum_{i=0}^{\log_2 n-3} 4^i$$

Endi aniq qiymatlar va (3.18) tenglikdan foydalanamiz:

$$T(N) = 4^{\log_2 n-2} \cdot 5\frac{4^{\log_2 n-2} - 1}{4-1};$$

$$T(N) = 4^{\log_2 n-2} \cdot 5\frac{4^{\log_2 n-2} - 1}{3};$$

$$T(N) = \frac{15 \cdot 4^{\log_2 n-2} - 4^{\log_2 n-2} + 1}{3};$$

$$T(N) = \frac{4^{\log_2 n-2}(15-1)+1}{3};$$

$$T(N) = \frac{4^{\log_2 n-2} \cdot 14+1}{3}.$$

Ko‘rinib turibdiki, chto v zamknutom vide rekurrent munosabatning yopiq ko‘rinishi juda ham sodda bo‘lmasligi mumkin, biroq unga o‘tishda rekursiv “chaqiruv”dan xoli bo‘lamiz va shuning uchun hosil qilingan natijalarni tezda taqqoslab, ularning tartibini aniqlay olamiz.

Dasturlar tahlili. Faraz qiling, bizda katta murakkab dastur bor, u o‘ylaganimizdan ham sekin ishlaydi. Uning ishlashini sezilarli darajada tezlashtiradigan dasturning sozlovchi qismlarini qanday bilish mumkin?

Dasturga qarab ko‘p hisoblashlar yoki sikllar mavjud bo‘lgan qismdasturlarni (protsedura yoki funksiya deb ham nomlanadi) topish mumkin. E’tiborliroq bo‘lsak, tanlangan qismdasturlar doim ishlatilmaganligi uchun samara sezilarli emasligini anglashimiz mumkin. Dastlab doimiy ishlatiladigan qismdasturlarni topib, ularni yaxshilashga harakat qilish ma’qulroq. Izlash usullaridn biri bu har bir qismdasturga bittadan global hisoblagichlar to‘plamini kiritishdan iborat. Dastur ish boshlashida hisoblagichlar nolga tenglashtiriladi. So‘ngra har bir qismdasturning birinchi qatoriga mos hisoblagichni 1 ga oshirish buyrug‘i qo‘yiladi. Qismdasturga har qanday murojaatda hisoblagich ortib boradi va ish yakunida hisoblagich har bir qismdasturga nechta chaqiruv bo‘lganligini ko‘rsatadi. Shunda qaysi qismdasturlar ko‘proq, qaysilari kam chaqirilganligini ko‘rish mumkin.

Faraz qilamiz, bizning dasturda ba’zi oddiy qismdasturlar 50 000 marta, murakkab qismdasturlar esa bir marta chaqirildi. U holda oddiy dasturda bitta amalni o‘chirganda hosil bo‘ladigan natijaga erishish uchun murakkab dasturlarda amallar sonini 50 000 martaga kamaytirish kerak. Bitta dasturda oddiy yaxshilanishni topish qism dasturlar guruhidagi 50 000 yaxshilanishni topishga ko‘ra yengilroq ekanligi tushunarli.

Hisoblagichlarni qismdasturlar darajalarida ham ishlatish mumkin. Bu holatda biz oldindan topa olishimiz mumkin bo‘lgan har bir kerakli nuqtada bittadan global hisoblagichlar to‘plamini yaratamiz. Faraz qilamiz, biror **if** operatorining qismlari bo‘lgan **then** va **else** ning har biri necha marta bajarilishini bilmoqchimiz. U holda ikkita hisoblagich yaratish mumkin va ulardan birinchisini then qismiga tushganda, ikkinchisini – **else** qismiga tushganda kattalashtirish mumkin. Dastur oxirida hisoblagichlarda bizni qiziqtiradigan axborot bo‘ladi. Umuman olganda, hisoblagichlarni boshqarish imkoniyati bo‘lgan ixtiyoriy joyda o‘rnatish kerak.

Dastur o‘z ishini tugatgach hisoblagichlarda har bir qismdasturlarning bajarilish soni haqidagi ma’lumot bo‘ladi. Keyin ishning eng katta hajmini bajarayotgan qismdasturlarni yaxshilash imkoniyatini ko‘rib chiqish kerak.

Bu jarayon ko‘plab kompyuter va tizimlar ishlab chiqishda juda muhim, dasturlar haqida ma’lumot olishning avtomatik vositasi dasturiy ta’minoti bor.

Savol va topshiriqlar

1. ***Rekursiya nima?***
2. ***Iteratsiya tushunchasini tushuntiring?***
3. ***Rekursiv algoritmlarning samaradorligini izohlang.***
4. ***Rekurrent munosabatga misol keltiring?***
5. Rekursiya va rekurrent algoritmlarga doir misollar keltiring?
6. Dasturlar tahlilini tushuntiring?
7. $S = \sum_{i=1}^{n} \frac{x_i}{(2i+i)!}$ ifodani hisoblash uchun rekkurent ifodasini keltiring.