ՆԵՐԱԾՈՒԹՅՈՒՆ

Առարկայի ուսումնասիրության նյութը տվյալների կառույցներն են, նրանցում տվյալների ներկայացման, ապա մշակման եղանակները:

Շատ հարցեր են ծնվում, օրինակ, ծրագրավորման ժամանակ հետևյալ գործողություններն իրականացնելիս.

ա) մեծ ինֆորմացիոն զանգվածների ներմուծում,

բ) ԷՀՄ-ի հիշողությունում նրանց ներկայացման և պահպանման կազմակերպում,

գ) մեծաքանակ տվյալների մշակում:

Պատկերացնենք, թե ինչքան դժվար կլիներ աշխատել հայերեն ուղղագրական բառարանի հետ, եթե նրանում բառերը չտեղադրվեին որոշակի կարգով (օրինակ՝ այբբենական): Համապատասխանաբար, ԷՀՄ-ում տվյալների ներկայացման և պահպանման ձևից կախված, որոշվում է նրանց մշակումն իրականացնող ալգորիթմների բարդությունը:

Տվյալների պահպանման համար կառաջարկվեն տարբեր կառույցներ, նրանցից յուրաքանչյուրի համար կդրվեն հետևյալ երկու խնդիրները՝ ***կարգավորման և փնտրման***, որոնց իրականացման ընթացքում կդիտարկվեն հետևյալ հարցերը.

ա) տվյալների ներկայացման համար ինչպե՞ս ընտրել անհրաժեշտ կառույցը,

բ) տվյալ խնդրի իրականացման համար ինչպե՞ս գտնել լավ ալգորիթմներ,

գ) ինչպե՞ս լավացնել տրված ալգորիթմը,

դ) կոնկրետ խնդիր իրականացնող ալգորիթմներից ինչպե՞ս ընտրել մեկը,

ե) ինչպե՞ս կարելի է ապացուցել, որ որոշ ալգորիթմներ ինչոր իմաստով «հնարավորներից լավագույնն են»:

Դիտարկենք ուսումնական հաստատության ուսանողների անձնական գործերը, որոնցում յուրաքանչյուր ուսանողի մասին նշված են հետևյալ կարգի տվյալներ՝ ազգանունը, անունը, հայ-

րանունը, ստուգման գրքույկի համարը, ֆակուլտետը, ծննդյան թիվը, սեռը, հասցեն, տարբեր քննությունների արդյունքները և այլն:

Նմանատիպ ինֆորմացիայի համախումբը անվանենք **ինֆորմացիոն զանգված (մասսիվ)**:

Մեկ ուսանողի մասին պահվող ինֆորմացիան անվանենք **գրառում**:

Բերենք մեկ գրառման օրինակ՝ 61301, Գասպարյան Կարեն Գրիգորի, 1979 թ., բնագիտական, արական, ք.Իջևան , 4:

Այս գրառման մեջ 61301-ը Գասպարյան Կարեն Գրիգորի ազգանուն, անուն, հայրանունն ունեցող բնագիտական ֆակուլտետի ուսանողի ստուգման գրքույկի համարն է, նա ծնվել է 1979 թվականին, արական սեռի ներկայացուցիչ է, բնակվում է Իջևանում, «Տվյալների կառույցներ» առարկայից ստացել է 4 գնահատական:

Ինֆորմացիան ներկայացնող գրառման որոշակի հատկությունը կոչվում է ***ատրիբուտ***: Այդ հատկություններով օժտված են ինֆորմացիոն զանգվածի բոլոր գրառումները:

Մեր ինֆորմացիոն զանգվածի ատրիբուտներն են՝ *«ազգանուն», «անուն», «հայրանուն», «ստուգման գրքույկի համարը», «ֆակուլտետ», «ծննդյան թիվ», «սեռ», «հասցե», «գնահատական»:*

Վերևում բերված գրառման համար Գասպարյանը *«ազգանուն»* ատրիբուտի ընդունած արժեքն է, բնագիտականը՝ *«ֆակուլտետ»* ատրիբուտի ընդունած արժեքն է, արականը՝ *«սեռ»* ատրիբուտի ընդունած արժեքն է և այլն:

<u>ՈՒշադրություն</u>. ստացանք՝

*Ատրիբուտների արժեքների համախումբը կոչվում է **գրառում**:*

*Գրառումների համախումբը կոչվում է **ինֆորմացիոն զանգված**:*

Մտցնենք ինֆորմացիոն զանգվածի գրառումները մեկը մյուսից տարբերակող բնութագրիչ:

Նկատենք, որ մեր ինֆորմացիոն զանգվածի գրառումներն իրարից տարբերվում են ընտրված ատրիբուտներից գոնե մեկի ընդունած արժեքների վրա:

Մեկ կամ մի քանի ատրիբուտների համախումբը, որոնց ընդունած արժեքների վրա գրառումները դառնում են տարբեր անվանենք ***հայտանիշ (բանալի):***

Մեր օրինակում որպես հայտանիշ կարող է հանդես գալ «ստուգման գրքույկի համարը» ատրիբուտը, կամ «ազգանուն», «անուն», «հայրանուն» ատրիբուտների համախումբը:

Տրված ինֆորմացիոն զանգվածի համար առանձնացնելով բանալին՝ գրառման մեջ համախմբենք մնացած ատրիբուտների արժեքները և անվանենք «ուղեկցող ինֆորմացիա», որը կարգավորման և փնտրման խնդիրներում կարող է մշակման գործընթացին չմասնակցել, սակայն միշտ մնա գրառման հետ:

1. ԿԱՐԳԱՎՈՐՄԱՆ ԵՎ ՓՆՏՐՄԱՆ ԽՆԴԻՐՆԵՐԻ ԴՐՎԱԾՔՆԵՐԸ

Ինֆորմացիոն մասսիվի գրառումները դիտարկենք ըստ իրենց հանդիպելու հաջորդականության, այսինքն ինֆորմացիոն մասսիվի i-րդ տեղում գտնվող գրառումը նշանակենք R_i-ով, նրա հայտանիշի արժեքը՝ K_i-ով, ուղեկցող ինֆորմացիան՝ INF_i-ով.

R_i :	K_i	INF_i

Այդ դեպքում ինֆորմացիոն մասսիվը դիտարկում ենք որպես տրված կառուցվածքն ունեցող գրառումների հաջորդականություն:

Գրառման **կարգահամար** ասելով կհասկանանք գրառումների հաջորդականության մեջ տվյալ գրառման տեղի համարը:

Գրառումների հայտանիշների արժեքների բազմության մեջ մտցնենք **կարգի գաղափարը** հետևյալ կերպ.

ա)կամայական երեք a, b, c հայտանիշների համար տեղի ունի a=b, a>b կամ a<b հարաբերություններից միայն մեկը,

բ) եթե a<b, b<c ապա a<c:

Հայտանիշների վրա դրված այս կարգը հնարավորություն է տալիս ինֆորմացիոն մասսիվի գրառումների հայտանիշների համեմատման արդյունքում կատարել եզրահանգումներ:

Դիցուք ունենք գրառումների $R_1,R_2,...,R_n$ հաջորդականությունը: Տանք կարգավորման և փնտրման խնդիրների դրվածքները:

<u>Կարգավորման խնդիրը.</u> $R_1,R_2,...,R_n$ գրառումների հաջորդականության համար գտնել գրառումների այնպիսի $R_{i_1}, R_{i_2},..., R_{i_n}$ դասավորություն, որում գրառումների հայտանիշների համար տեղի ունի հետևյալ կարգը.

$$K_{i_1} < K_{i_2} < ... < K_{i_n}$$

որից հետո հաջորդականությունը կոչվում է կարգավորված ըստ հայտանիշների աճման կարգի:

<u>Օրինակ</u>, ուսանողների ցուցակն ստանալ ըստ *«ազգանուն, անուն, հայրանուն» հայտանշի* աճման կարգի (այբբենական):

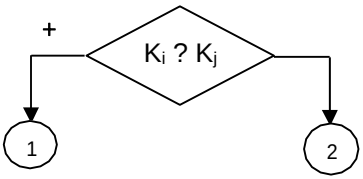
<u>Փնտրման խնդիրը.</u> $R_1,R_2,...,R_n$ գրառումների հաջորդականության մեջ ստանալ հայտանիշի տրված արժեքն ունեցող գրառման կարգահամարը (եթե այն գոյություն ունի):

Օրինակ, ստանալ 61035 «ստուգման գրքույկի համարը» հայտանիշով ուսանողի հասցեն:

2. ՀԱՄԵՄԱՏՈՒԹՅՈՒՆՆԵՐԻ ՔԱՆԱԿԻ ՏԵՍԱԿԱՆ ԳՆԱՀԱՏԱԿԱՆԸ

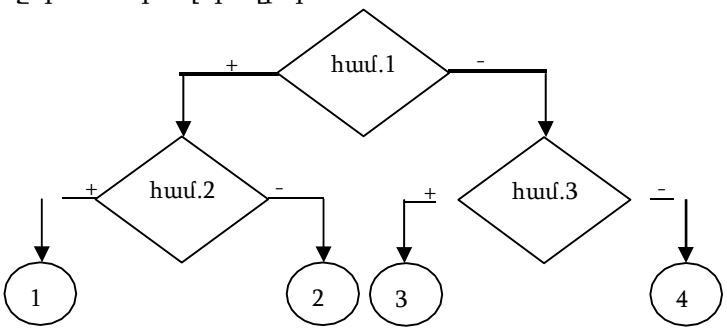
Դիցուք ունենք n տարրից կազմված $R_1,R_2,...,R_n$ գրառումների հաջորդականություն: Ակնհայտ է, որ երկու R_i և R_j գրառումների հայտանիշների համեմատման ժամանակ դրվում է հետևյալ հարցերից որևէ մեկը. «համընկնու՞մ են արդյոք երկու գրառումների K_i և K_j հայտանիշների արժեքները», «փո՞քր է արդյոք R_i գրառման K_i հայտանիշի արժեքը R_j գրառման K_j հայտանիշից» և այլն:

Յուրաքանչյուր այսպիսի համեմատում տալիս է երկու հնարավոր ելքեր՝ «այո» (+) կամ «ոչ» (-).



Սխեմայում «?» նշանը համընկնում է թույլատրելի հարաբերության գործողություններից որևէ մեկի հետ (մասնավորապես «=» կամ «<>»):

Տեսականորեն երկու հաջորդական համեմատումները տալիս են չորս հնարավոր ելքեր՝



Նույն տրամաբանությամբ՝ երեք հաջորդական համեմատումները տալիս են ութ հնարավոր ելքեր, m հաջորդական համեմատումները՝ 2^m հնարավոր ելքեր:

Եթե կոնկրետ խնդրի իրականացման համար պահանջվում է ճանաչել N տարբեր վիճակներ, ապա կատարվող համեմատութ-

յունների արդյունքում հնարավոր ելքերի քանակը պետք է փոքր չլինի վիճակների քանակից, հետևաբար համեմատությունների m քանակը պետք է բավարարի հետևյալ անհավասարությանը՝

$N \le 2^m$, որտեղից՝ $m \ge \log_2 N$:

Այժմ կոնկրետ խնդիրների համար տանք համեմատությունների m քանակի գնահատականը:

Փնտրման խնդրի համար համեմատությունների քանակի տեսական գնահատականը.

n գրառումներ պարունակող ինֆորմացիոն մասսիվում հայտանիշի տրված արժեքն ունեցող գրառման փնտրման ժամանակ առանձնացնենք բոլոր հնարավոր վիճակները.

«հայտանիշի տրված արժեքն ունի ինֆորմացիոն մասսիվի 1-ին գրառումը»,

«հայտանիշի տրված արժեքն ունի ինֆորմացիոն մասսիվի 2-րդ գրառումը»,

. . . . . .

«հայտանիշի տրված արժեքն ունի ինֆորմացիոն մասսիվի n-րդ գրառումը»,

«հայտանիշի տրված արժեքը չունի ինֆորմացիոն մասսիվի ոչ մի գրառում»:

Այստեղից ստանում ենք $N = n+1$, հետևաբար

$$m \ge \log_2 (n+1).$$

Օրինակ, եթե մեր ինֆորմացիոն մասսիվը կազմված է 1023 գրառումից, ապա տրված հայտանիշն ունեցող գրառման փնտրման համար ըստ այս գնահատականի կկատարվի $\log_2 1024 = 10$ համեմատությունից ոչ քիչ համեմատություն:

Ինչու՞մն է կայանում $m \ge \log_2 (n+1)$ բանաձևի էությունը: Այս բանաձևից հետևում է, որ համեմատությունների քանակը կախված է փնտրման մեթոդից, եթե համեմատությունների քանակը բավականին մեծ է (քան $\log_2 (n+1)$), ապա դա նշանակում է, որ ընտրված մեթոդը վատն է: $\log_2 (n+1)$-ին մոտ թիվ ստանալը կախ-

ված կլինի համեմատման ժամանակ ճիշտ հարցեր դնելու ունակությունից (հնարավորությունից):

Ուշադրություն. Այստեղ համեմատությունների քանակը դիտարկվում է առանց հաշվի առնելու հայտանիշի կառուցվածքը: Հայտանիշի բարդ կառուցվածքի դեպքում երկու հայտանիշների համեմատումն արդեն կպահանջի միգուցե մեկից ավելի համեմատումներ:

m □ log₂ (n+1) բանաձևը է թույլ է տալիս կատարել հետևյալ եզրահանգումը. գոյություն ունի փնտրման ալգորիթմ, որը հայտանիշի տրված արժեքն ունեցող գրառման փնտրման համար պահանջում է կատարել հենց **□log₂ (n+1)□** համեմատում, այդ ալգորիթմը կանվանենք ***փնտրման լավագույն ալգորիթմ***:

Կարգավորման խնդրի համար համեմատությունների քանակի տեսական գնահատականը.

n գրառում պարունակող ինֆորմացիոն մասսիվում գրառումների ըստ հայտանիշի նոր $K_{i_1} < K_{i_2} < \ldots < K_{i_n}$ դասավորություն ստանալու համար առանձնացնենք բոլոր հնարավոր վիճակները. հաջորդականության n գրառումները կարող են ունենալ n! տարբեր դասավորություններ: Հետևաբար, N=n! և ստանում ենք համեմատությունների m քանակի համար հետևյալ անհավասարությունը.

$$m \geq \log_2 n!$$

Գնահատենք m-ը, ունենալով, որ $n! \leq n^n$:

Ստանում ենք $m \geq \log_2 n^n \geq \log_2 n!$, հետևաբար,

m □ nlog₂ n :

Այս բանաձևի մեկնաբանությունը բերում է հետևյալ եզրահանգման. հնարավոր է գոյություն ունի կարգավորման ալգորիթմ, որը գրառումների կարգավորման համար պահանջում է կատարել հենց **□n log₂ n□** համեմատում, այդ ալգորիթմը կանվանենք ***կարգավորման լավագույն ալգորիթմ***:

Փնտրման և կարգավորման խնդիրների իրականացման համար կդիտարկենք տարբեր ալգորիթմներ, կստանանք նրանց գնահատականներն ըստ ալգորիթմում իրականացվող համեմատությունների քանակի և օգտագործած հիշողության ծավալի, նաև կատարվող տեղափոխությունների քանակի: Նշենք, որ խնդիրներից յուրաքանչյուրում համեմատությունների քանակը կախված է ԷՀՄ-ի հիշողությունում **գրառումների ներկայացման ձևից (տվյալների կառույցից)**, ապա կառույցից կախված խնդիրների իրականացման համար ընտրված մեթոդներից: Մեթոդի ընտրությունը ծնում է մշակող ալգորիթմը, որի կատարման ժամանակը ակնհայտ է, պայմանավորված է նրանում կատարվող համեմատման գործողությունների վրա ծախսած ժամանակով: Այդ ժամանակը կարելի է կրճատել միայն փոքրացնելով հայտանիշների արժեքների համեմատման վրա ծախսվող ժամանակը: Յուրաքանչյուր ալգորիթմի աշխատանքային բնութագրերի անալիզը թույլ կտա տրամաբանորեն ճիշտ ընտրություն կատարել թվում է համազոր մեթոդների մեջ:

Ալգորիթմները բաժանենք երկու խմբի՝ ներքին և արտաքին ալգորիթմներ:

Եթե ալգորիթմի աշխատանքի ժամանակ ինֆորմացիոն մասսիվը ամբողջությամբ գտնվում է ներքին հիշողությունում, ապա ալգորիթմը կոչվում է **ներքին**:

Եթե ալգորիթմի աշխատանքի ժամանակ ինֆորմացիոն մասսիվը կամ նրա մի մասը գտնվում է արտաքին հիշողությունում, ապա ալգորիթմը կոչվում է **արտաքին**:

ԷՀՄ-ի հիշողությունում ինֆորմացիոն մասսիվի պահպանման համար կառաջարկվեն ***տվյալների հաջորդական կազմակերպման, ցուցակային և ծառային կառույցները:***

3. ՏՎՅԱԼՆԵՐԻ ՀԱՋՈՐԴԱԿԱՆ ԿԱԶՄԱԿԵՐՊՈՒՄ

Դիցուք տրված է $R_1,R_2,...,R_n$ ինֆորմացիոն մասսիվը: Տվյալների հաջորդական կազմակերպման ժամանակ հաջորդականության գրառումները զբաղեցնում են հիշողության հաջորդական դաշտեր.

A:	R_1	R_2	...	R_n

որտեղ A-ն հաջորդականության տեղադրման սկզբնական հասցեն է և յուրաքանչյուր գրառման հասցեն որոշվում է իր կարգահամարից կախված A սկզբնական հասցեի նկատմամբ շեղումով.

$$Adress(R_i)=A+(i-1)*p,$$

այստեղ $Adress(R_i)$-ն R_i գրառման հասցեն է, p-ն՝ տվյալ ինֆորմացիոն մասսիվի մեկ գրառման զբաղեցրած հիշողության ծավալը՝ հաշված բայթերով:

Օրինակ, ծրագրավորման C++ լեզվում ինֆորմացիոն մասսիվի հաջորդական կազմակերպումը կներկայացվի հետևյալ նկարագրումով.

```
struct Node
{
    Key K;
    Information INF;
} ;
Node inf_mas [n] ;
```

Այստեղ **Key**-ը և **Information**-ը տիպերի անուններն են, որոնք որոշում են համապատասխանաբար գրառման բանալու և ուղեկցող ինֆորմացիայի տիպը, և պետք է նույնպես նկարագրված լինեն ծրագրավորողի կողմից: Այսպիսի նկարագրությամբ ինֆորմացիոն մասսիվի յուրաքանչյուր գրառմանը դիմում են իր կարգահամարով, օրինակ **inf_mas[i].K**, որը նշանակում է մասսիվի i-րդ գրառման բանալին:

Քանի որ երկու խնդիրներում համեմատմանը մասնակցում են բանալիները, ապա ուղեկցող ինֆորմացիան կարող ենք ան-

տեսել, և մասնավորապես դիտարկել այն դեպքերը, երբ բանալին թիվ է, իսկ բանալիների $K_1,K_2,...,K_n$ հաջորդականության փոխարեն կդիտարկենք թվերի $a_1,a_2,...,a_n$ հաջորդականությունը:

3.1. Փնտրման ալգորիթմներ տվյալների հաջորդական կազմակերպման համար

3.1.1. Հաջորդական փնտրում

Դիցուք տրված է $a_1,a_2,...,a_n$ հաջորդականությունը: Ստանալ հաջորդականության այն տարրի կարգահամարը, որի արժեքը հավասար է տրված M թվին (եթե այդպիսինը կա):

Ալգորիթմը. Տրված M արժեքով տարրի փնտրման համար հաջորդականության յուրաքանչյուր a_i տարրը համեմատում ենք տրված M-ի հետ: Եթե նրանք համընկնում են, ապա ալգորիթմն ավարտում է աշխատանքը արտածելով այդ տարրի կարգահամարը, հակառակ դեպքում նույն քայլը իրականացվում է հաջորդականության հաջորդ տարրի հետ, քանի դեռ $i<n$:

Ալգորիթմի ավարտից հետո, եթե $i \le n$, ապա դա նշանակում է, որ փնտրվող տարրը կա և նրա կարգահամարը i-ն է (հաշված 1-ից), իսկ եթե $i=n+1$, ապա դա նշանակում է, որ տրված արժեքով տարր չկա:

```
#include <iostream.h>
const int n=1000;
int a[n],M;
void main()
{
    cout<<"Ներմուծել հաջորդականությունը.";
    for(int i=0;i<n;i++) cin>>a[i];
    cout<<"Ներմուծել փնտրվող թիվը –"; cin>>M;
    i=0;
    while ((i<n) && (a[i]!=M)) i++;
    if (i==n) cout<<'այդպիսի տարր չկա հաջ. մեջ<<endl;
```

```
    else cout<<'այդպիսի տարր կա և << i+1<<' -րդն է<<endl;
}
```

Այս ծրագրի վերլուծությունը ցույց է տալիս, որ այս ալգորիթմի կատարման ժամանակը կախված է 2 մեծություններից՝ համեմատությունների քանակից (նշանակենք C(n)) և գծային գործողությունների քանակից (նշանակենք A(n)): Հետևաբար, գործողությունների քանակը՝ C=C(n)+A(n):

Ալգորիթմում ցիկլի յուրաքանչյուր քայլում, երբ դեռ $M \neq a_i$, կատարվում է համեմատման 2 գործողություն ($M==a_i$ և $i<n$) և մեկ վերագրման գործողություն (i++), իսկ երբ տրված արժեքով տարր մասսիվում կա, այսինքն ($a_i==M$), ապա այդ i-րդ քայլում կատարվում է միայն մեկ համեմատման գործողություն: Հետևաբար, երբ տարրը գտնվում է (այսինքն՝ լավագույն ելքի դեպքում), ապա գործողությունների քանակը. C=2i-1+i-1=3i+2, իսկ երբ տարրը չի գտնվում (վատագույն ելքի դեպքում), ապա գործողությունների քանակը հավասար է. C=2n+n=3n:

Զբաղեցրած հիշողության ծավալը հավասար է. M(n)=n*p, որտեղ p-ն հիշողության մեջ մեկ գրառման զբաղեցրած ծավալն է:

Այս ալգորիթմն, անկասկած, հայտնի է բոլոր ծրագրավորողներին, սակայն նշենք, որ հաջորդաբար փնտրման այս մեթոդը ակնհայտ է որ ամենալավը չէ:

3.1.2. Արագ հաջորդական փնտրում

Ի տարբերություն նախորդ ալգորիթմի, այստեղ ենթադրվում է, որ մասսիվի վերջում կանգնած է ֆիկտիվ a_{n+1} տարրը, որին վերագրվում է հենց փնտրվող թիվը ($a_{n+1}=M$).

```
#include <iostream.h>
const int n=1000;
int a[n+1],M;
void main()
{
    cout<<"Ներմուծել հաջորդականությունը.";
    for(int i=0;i<n;i++) cin>>a[i];
```

```
    cout<<"Ներմուծել փնտրվող թիվը –"; cin>>M;
    a[n]=M;
    i=0;
    while (a[i]!=M) i++;
    if (i==n) cout<<'այդպիսի տարր չկա հաջ. մեջ<<endl;
    else cout<<'այդպիսի տարր կա և << i+1<<' -րդն է<<endl;
}
```

Այս ալգորիթմի դեպքում նույն սկզբունքով, երբ անհրաժեշտ տարրը գտնվում է ($M==a_i$), ապա կատարված գործողությունների քանակը՝ $C=i+i-1=2i-1$ (յուրաքանչյուր քայլում մեկ համեմատում և մեկ գծային գործողություն), երբ անհրաժեշտ գրառումը չի գտնվում, այսինքն $i=n+1$, ապա գործողությունների քանակը՝ $C=n+1+n=2n+1$:

Զբաղեցրած հիշողության ծավալը՝ $M(n)=(n+1)*p=n*p+p$:

Առաջին ալգորիթմից 2-ին անցնելուց օգտագործվել է կարևոր արագացնող սկզբունք. եթե առաջին ալգորիթմի ներքին ցիկլում ստացվում են համեմատման 2 պայմաններ, ապա 2-րդ ալգորիթմում՝ միայն մեկը:

3.1.3. Հաջորդական փնտրում կարգավորված մասիվում

Այժմ ներկայացնենք հաջորդական փնտրման ալգորիթմը կարգավորված հաջորդականության համար:

Սահմանում: $a_1,a_2,...,a_n$ հաջորդականությունը կոչվում է կարգավորված աճման կարգով, եթե նրա կամայական երկու տարրերի համար տեղի ունի հետևյալը. $\forall$ i,j համար i,j =1,2,...,n, երբ $i<j$, ապա $a_i<a_j$, այսինքն տեղի ունի՝ $a_1<a_2<...<a_n$:

Ալգորիթմը. Տրված M արժեքով տարրի փնտրման համար հաջորդականության յուրաքանչյուր a_i տարրի համար ստուգում ենք $a_i>M$, եթե այո, ապա ալգորիթմն ավարտում է աշխատանքն արտածելով «ոչ»՝ այսինքն հաջորդականությունում փնտրվող տարրը չկա, հակառակ դեպքում ստուգում ենք, եթե $a_i==M$, ապա ալգորիթմն ավարտում է աշխատանքը արտածելով «այո»՝ այսինքն հաջորդականությունում փնտրվող տարրը կա և նրա կար-

գահամարը i-ն է, հակառակ դեպքում ($a_i<M$) նույն քայլը իրականացվում է հաջորդականության հաջորդ տարրի հետ, քանի դեռ i<n:

```
#include <iostream.h>
const int n=1000;
int a[n],M;
void main()
{
    cout<<"Ներմուծել կարգավորված հաջորդականությունը.";
    for(int i=0;i<n;i++) cin>>a[i];
    cout<<"Ներմուծել փնտրվող թիվը –"; cin>>M;
    i=0;
    while (i<n && a[i]<M) i++;
    if (i<n || a[i]==M) cout<<'այդպիսի տարր կա և << i+1
    <<' -րդն է<<endl;
    else cout<<i+1<<' և այդպիսի տարր հաջ. մեջ չկա '<<endl;
}
```

Այս ալգորիթմում բարեհաջող ելքի դեպքում (i<n և $a_i==M$)՝ C(n)=2*i+2, A(n)=i, հետևաբար C=2*i+i+2=3*i+2; անհաջող ելքի դեպքում. երբ i<n և $a_i\neq M$, ապա C=3*i+2, A(n)=i, իսկ երբ i>n, C=2*n+2:

3.1.4. Արագ հաջորդական փնտրում կարգավորված մասսիվում

Հարմարության և ալգորիթմի արագության մեծացման համար կարգավորված հաջորդականության վերջում ավելացնենք ֆիկտիվ a_{n+1} =maxint տարրը:

```
#include <iostream.h>
const int n=1000;
int a[n+1],M;
void main()
{
    cout<<"Ներմուծել հաջորդականությունը.";
```

```
    for(int i=0;i<n;i++) cin>>a[i];
    cout<<"Ներմուծել փնտրվող թիվը –"; cin>>M;
    a[n]=maxint;
    i=0;
    while (a[i]<M) i++;
    if (i<n && a[i]==M) cout<<'այդպիսի տարր կա և << i+1
       <<' -րդն է<<endl;
    else cout<<'այդպիսի տարր հաջ. մեջ չկա <<endl;
}
```

Այս ալգորիթմում բարեհաջող ելքի դեպքում ($i<n$ և $a_i==M$)՝ $C(n)=i+2$, $A(n)=i$, հետևաբար $C=i+i+2=2*i+2$; անհաջող ելքի դեպքում. երբ $i<n$ և $a_i \neq M$, ապա $C=i+2$, $A(n)=i$, իսկ երբ $i>n$, $C=n+3$, $M(n)=n*p+p$:

Այս ալգորիթմը թույլ է տալիս մոտավորապես 2 անգամ արագ գուշակել այդ փաստը (տարրերի կարգավորված լինելու և օժանդակ տարրի առկայության հաշվին):

Այս բոլոր ալգորիթմներում գնահատականները հիմնված են ոչ միայն տարրերի տրամաբանորեն պահանջվող հիմնական համեմատման, այլ նաև ռեալ կատարվող համեմատությունների քանակի հաշվման վրա:

3.1.5. Փնտրման կիսման մեթոդը

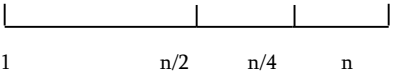
Փնտրման կիսման մեթոդը կիրառվում է կարգավորված մասսիվների համար: Դիցուք ունենք $a_1,a_2,...,a_n$ հաջորդականությունը, որի տարրերը դասավորված են աճման կարգով, այսինքն՝ $a_1<a_2<...<a_n$:

Փնտրումը կատարվում է միջակայքերով: Նախապես, որպես փնտրման միջակայք վերցվում է ինդեքսների 1..n միջակայքը: Ապա այդ միջակայքը տրոհվում է 2 մասի (կիսվում է $k=(n+1)/2$), ապա միջակայքի կենտրոնի a_k տարրի արժեքը համեմատվում է տրված M արժեքի հետ, որի արդյունքում փնտրման համար ընտրվում է այդ 2 միջակայքերից միայն մեկը.

եթե $a_k>M$, ապա 1.. k-1 միջակայքը,

եթե $a_k<M$, ապա k+1..n միջակայքը,
որում նորից կիրառվում է կիսման մեխանիզմը:

Ալգորիթմի առանձնահատկությունն այն է, որ ցիկլի յուրաքանչյուր կրկնման ժամանակ դեն են նետվում դիտարկվող միջակայքի տարրերի մոտավորապես կեսը:



1..n միջակայքը որպեսզի կիսվի m անգամ (իսկ դա նշանակում է, որ կկատարվի m համեմատում), ապա պետք է $2^m \geq n$, որտեղից ստացվում է

$$m \geq \log_2 n, \text{ այսինքն } C(n) = \lfloor \log_2 n \rfloor :$$

Զբաղեցրած հիշողության ծավալը՝ $M(n)= n^*p$:

Կարգավորված մասսիվում կիսման մեթոդով փնտրումը տալիս է փնտրման լավագույն ալգորիթմի գնահատականը:

3.2.Կարգավորման ալգորիթմներ տվյալների հաջորդական կազմակերպման համար

Խնդրի դրվածքը. Դիցուք տրված է $a_1,a_2,...,a_n$ հաջորդականությունը: Վերադասավորել հաջորդականությունն այնպես, որ $\forall$ i,j համար i,j=1,2,...,n, եթե $i<j$, ապա $a_i<a_j$: Այս պայմանի դեպքում մասսիվը կոչվում է կարգավորված աճման կարգով:

Կդիտարկենք ներքին կարգավորման հետևյալ ալգորիթմները՝

1.փոխատեղումով կարգավորում կամ պղպջակի մեթոդ,
2.ընտրումով կարգավորում,
3.տեղադրումով կարգավորում,
4.կարգավորում լրացուցիչ հիշողությամբ,
5.կարգավորում ըստ էլեմենտի տեղի,
6.կարգավորման Ֆոն-Նեյմանի մեթոդ,
7.քառակուսային կարգավորման մեթոդ,
8. կարգավորման Շելլի մեթոդ:

3.2.1. Պղպջակի մեթոդ

Մեթոդի նկարագրությունը. $a_1,a_2,...a_n$ հաջորդականության կարգավորումը կատարվում է քայլերով, յուրաքանչյուր քայլում դիտարկվում է տարրերի 1..j (j=n-1,n-2,...,2) միջակայքը, հաջորդաբար համեմատվում են 2 հարևան տարրերը՝ a_k և a_{k+1} (k=1,2,3,...j), և եթե $a_k>a_{k+1}$, ապա նրանք փոխում են իրենց տեղերը, և այդպես մինչև միջակայքի վերջին տարրին հասնելը, որի արդյունքում հաջորդականության ամենամեծ տարրը հայտնվում է իր տեղում՝ դիտարկվող միջակայքի վերջում և այլևս չի մասնակցում կարգավորման պրոցեսին: Նկատենք, որ այս մեխանիզմը կիրառվում է n-1 անգամ:

Հաշվենք կարգավորման պրոցեսում կատարվող համեմատությունների և փոխատեղումների քանակը.

1-ին քայլում կատարվում է n-1 համեմատում և ամենաշատը n-1 փոխատեղում,

2-րդ քայլում կատարվում է n-2 համեմատում և ամենաշատը n-2 փոխատեղում,

k-րդ քայլում կատարվում է n-k համեմատում և ամենաշատը n-k փոխատեղում,

(n-1)-րդ քայլում կատարվում է 1 համեմատում և 1-ից ոչ ավել փոխատեղում:

Ընդհանուր համեմատությունների թիվը կլինի՝

$C(n)=(n-1)+(n-2)+...+1=n(n-1)/2=n^2/2-n/2\approx n^2/2$,

ընդհանուր փոխատեղումների թիվը՝

$T(n)\le(n-1)+(n-2)+...+1=n^2/2-n/2\approx n^2/2$

Օրինակ՝ երբ n=1000, ապա կստանանք C(n) ≈500000 համեմատություն, T(n) ≈500000 փոխատեղում, իսկ լավագույն կարգավորման ալգորիթմի համար m=10000 համեմատություն:

Այս մեթոդով կարգավորման ժամանակ ծախսվում է $M(n)=n^*p$ ծավալի հիշողություն:

3.2.2. Ընտրումով կարգավորում

Մեթոդի նկարագրությունը. Առաջին քայլում $a_1,a_2,\dots,a_n$ հաջորդականության մեջ գտնել ամենամեծ տարրը և տեղադրել n-րդ տեղում, 2-րդ քայլում մնացած n-1 տարրերի մեջ գտնել ամենամեծ տարրը և տեղադրել (n-1)-րդ տեղում, և այդպես մինչև վերջ:

Այսինքն՝

1-ին քայլում գտնում ենք $a_n=\max\{a_i\}$, $1\le i\le n$,

2-րդ քայլում գտնում ենք $a_{n-1}=\max\{a_i\}$, $1\le i\le n-1$,

k-րդ քայլում գտնում ենք $a_{n-k+1}=\max\{a_i\}$, $1\le i\le n-k+1$,

(n-1)-րդ քայլում՝ $a_2=\max\{a_1,a_2\}$:

Հաշվենք կարգավորման պրոցեսում կատարված համեմատությունների և փոխատեղումների քանակը.

1-ին քայլում կատարվում է n-1 համեմատում, 1 փոխատեղում,

2-րդ քայլում կատարվում է n-2 համեմատում, 1 փոխատեղում,

k-րդ քայլում կատարվում է n-k համեմատում, 1 փոխատեղում,

(n-1)-րդ քայլում կատարվում է 1 համեմատում, 1 փոխատեղում:

Ընդհանուր համեմատությունների թիվը հավասար է՝

$$C(n)=(n-1)+(n-2)+\dots+1=n(n-1)/2=n^2/2-n/2)\approx n^2/2,$$

(մեծ n-երի դեպքում քանի որ n/2 այնքան էլ էական կշիռ չունի, ապա այն անտեսվել է):

Յուրաքանչյուր քայլում կատարվում է 1 փոխատեղում, հետևաբար ընդհանուր փոխատեղումների թիվը հավասար կլինի՝

$T(n)=1+1+\dots+1=n-1$:

Օրինակ, երբ n=1000, ապա կստանանք C(n)=500000 համեմատություն, իսկ T(n)=999:

Այստեղից հետևում է, որ պղպջակի և ընտրումով կարգավորման մեթոդներում համեմատությունների քանակները հավա-

սար են, իսկ փոխատեղումների քանակը պղպջակի մեթոդում բավականին շատ է:

Այս մեթոդով կարգավորման ժամանակ ծախսվում է M(n)=n*p ծավալի հիշողություն:

3.2.3. Տեղադրումով կարգավորում

Մեթոդի նկարագրությունը. Դիցուք $a_1,a_2,..,a_n$ հաջորդականության առաջին k տարրերն արդեն դասավորված են աճման կարգով՝ $a_1<a_2<...<a_k,a_{k+1},...,a_n$ k>1: k-րդ քայլում վերցվում է (k+1)-րդ տարրը և տեղադրվում առաջին k հատ կարգավորված տարրերի մեջ իր տեղում:

Տեղադրման մեխանիզմը. a_{k+1} տարրը հաջորդաբար համեմատվում է իրեն նախորդող տարրերի հետ, եթե $\forall$ a_i (i≤k) տարրի համար $a_{i-1}>a_i$, ապա այդ 2 տարրերը փոխում են իրենց տեղերը, հենց ($a_{i-1}<a_i$) կամ (i=1), ապա դա նշանակում է, որ դիտարկվող a_{k+1} տարրն արդեն գտնվում է իր տեղում և մեր հաջորդականությունն ունի հետևյալ տեսքը.

$$a_1<a_2,<...<a_{k+1,}a_{k+2},...,a_n:$$

Հաշվենք կարգավորման պրոցեսում կատարվող համեմատությունների քանակը.

1-ին քայլում համեմատությունների քանակը =1, փոխատեղումների քանակը ≤1,

2-րդ քայլում համեմատությունների քանակը ≤2, փոխատեղումների քանակը ≤2,

k-րդ քայլում համեմատությունների քանակը ≤ k, փոխատեղումների քանակը≤ k,

(n-1)-րդ քայլում համեմատությունների քանակը ≤ n-1, փոխատեղումների քանակը ≤ n-1,

հետևաբար $a_1,a_2,...,a_n$ հաջորդականության տեղադրումով կարգավորում մեթոդով կարգավորման համար պահանջվում է.

$$C(n)\le1+2+...+n-1=n(n-1)/2\approx n^2/2$$

համեմատություն, և

$$T(n) \le1+2+...+n-1\approx n^2/2$$

փոխատեղում, զբաղեցրած հիշողության ծավալը հավասար է M(n)=n*p:

Եթե առաջին k հատ կարգավորված տարրերի մեջ *նոր տարրի տեղի փնտրման համար կիրառվի կիսման մեթոդը*, ապա այդ դեպքում համեմատությունների քանակը՝

$$C(n) = \sum_{k=2}^{n} \log_2 k = \log_2(1^*2^*...^*n^*) = \log_2 n! \approx n\log_2 n,$$

իսկ $T(n) \approx \frac{n^2}{2}$:

Այսինքն այս դեպքում տեղադրումով կարգավորումը համեմատությունների քանակով տալիս է կարգավորման լավագույն ալգորիթմի տեսական գնահատականը:

3.2.4. Կարգավորում լրացուցիչ հիշողությամբ

Մեթոդի նկարագրությունը. Դիցուք ունենք $a_1,a_2,...,a_n$ հաջորդականությունը: n տարրերի համար վերցնենք օժանդակ $b_1,b_2,...,b_n$ հաջորդականությունը: Կարգավորումն իրականացվում է էտապներով: Յուրաքանչյուր քայլում a հաջորդականության հերթական տարրը տեղափոխվում է b մասսիվ, իսկ b մասսիվում տարրերը տեղադրվում են կարգավորված: Տեղադրվող յուրաքանչյուր տարր b մասսիվում իր տեղը գտնում է կիսման մեթոդով:

k-րդ քայլում a[k] տարրը b հաջորդականության k-1 հատ տարրերի մեջ տեղադրվում է իր տեղում, որի իրականացման ժամանակ b հաջորդականությանում կատարվում է նաև տարրերի տեղաշարժ: Օրինակ՝

a	b I	II	III	IV	V	VI	VII	ելք
-10	-10	-10	-10	-10	-10	-10	-10	-10
15		15	2	2	-6	-6	-6	-6
2			15	7	2	2	2	2
7				15	7	3	3	3
-6					15	7	7	7
3						15	12	12
12							15	15

Ծախսած հիշողության ծավալը՝ M(n)=2*n*p

Այս ալգորիթմի համեմատությունների թիվը՝

$$C(n)=\sum_{k=1}^{n}\log_2 k=\log_2(1*2*...*n)=\log_2 n!\approx n\log_2 n$$

Ինչպես տեսնում ենք, այս մեթոդի համար մենք ստացանք կարգավորման լավագույն ալգորիթմի գնահատականը, այսինքն այս ալգորիթմը պահանջում է ամենաքիչ համեմատություններ, սակայն այս ալգորիթմն ունի այլ թերություն՝ b հաջորդականությանում կատարվում են տարրերի բավականին մեծ քանակությամբ տեղաշարժեր՝

$$T(n)=\sum_{k=1}^{n}\frac{k-1}{2}=\frac{1}{4}n(n-1)\approx\frac{n^2}{4}:$$

այստեղ (k-1)/2-ը դա k-րդ քայլում b մասսիվում տեղաշարժերի միջին թիվն է:

Խնդիրը հետևյալն է. հնարավո՞ր է արդյոք տեղաշարժերի թիվը քչացնել այնպես, որ ամեն մի համեմատմանը զուգահեռ կատարվի միայն մեկ տեղաշարժ, այդ դեպքում տեղաշարժերի թիվը մոտավորապես կլինի nlog2n կարգի:

Այդպիսի հատկությամբ օժտված է կարգավորման Ֆոն-Նեյմանի մեթոդը:

3.2.5. Կարգավորում Ֆոն-Նեյմանի մեթոդով

Մեթոդի նկարագրությունը. Մեթոդը հիմնված է երկու կարգավորված հաջորդականություններից մեկ կարգավորված հաջորդականություն ստանալու սկզբունքի վրա:

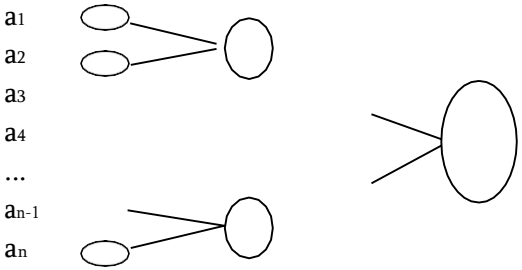
Դիցուք ունենք $a_1,a_2,....a_n$ և $b_1,b_2,....,b_m$ կարգավորված հաջորդականությունները: Այս երկու կարգավորված հաջորդականություններից ստանալ նոր $c_1,c_2,....,c_{n+m}$ կարգավորված հաջորդականությունը:

Այս ալգորիթմից հետևում է, որ կարգավորման ժամանակ կատարվում է ամենաշատը n+m-1 համեմատություն:

Այժմ նկարագրենք կարգավորման Ֆոն-Նեյմանի մեթոդը:

Կարգավորումը կատարվում է փուլերով: *Առաջին* փուլում հաջորդականությունը բաժանվում է կարգավորված բլոկների, յուրաքանչյուրում 2-ից ոչ ավել տարր: *Երկրորդ* փուլում կարգա-

վորված բլոկները միաձուլվում են և առաջացնում նոր կարգավորված բլոկներ՝ յուրաքանչյուրում 4-ից ոչ ավել տարր, ընդհանուր դեպքում *k-րդ* փուլից հետո կարգավորված բլոկներում տարրերի քանակը կլինի 2^k-ից ոչ ավել:



Ամեն փուլի արդյունքում կարգավորված բլոկների քանակը պակասում է երկու անգամ, իսկ նոր առաջացող բլոկներում տարրերի քանակը ավելանում է երկու անգամ:

Նախապես հաջորդականությունը պարունակում է n բլոկ, յուրաքանչյուրում 1 տարր:

1-ին փուլում առաջանում են **n/2 բլոկներ**, յուրաքանչյուրում՝ **2 տարրից** ոչ ավել (կարգավորման համար յուրաքանչյուր բլոկում կատարվում է **1 համեմատում**),

2-րդ փուլում առաջանում են **n/4 բլոկներ**, յուրաքանչյուրում՝ **4 տարրից** ոչ ավել (կարգավորման համար յուրաքանչյուր բլոկում կատարվում է **3 համեմատում**),

k-րդ փուլում առաջանում են **$n/2^k$ բլոկներ**, յուրաքանչյուրում՝ **2^k տարրից** ոչ ավել (կարգավորման համար յուրաքանչյուր բլոկում կատարվում է **2^k-1 համեմատում**):

Այժմ հաշվենք համեմատությունների քանակը.

- փուլերի թիվը հավասար է $\lceil \log_2 n \rceil = \log_2 n+1$ (ոչ ավել),
- k-րդ փուլում համեմատությունների թիվը հավասար է.

$$C_k(n) \le \frac{n}{2^k}(2^k - 1) = n - n / 2^k \approx n$$

- ըստ բոլոր փուլերի.

$$C(n) \le \sum_{i=1}^{\log_2 n+1} \frac{n}{2^i}\left(2^i - 1\right) = n\left(\log_2 n + 1\right) - n \cdot \sum_{i=1}^{\log_2 n+1} \frac{1}{2^i} =$$

$$= n\left(\log_2 n + 1\right) - n\left(1 - \frac{1}{2n}\right) = n \log_2 n + \frac{1}{2} \approx n \log_2 n$$

Յուրաքանչյուր փուլում համեմատությունների և տեղաշարժերի քանակը հավասար է n, կարգավորումը կկատարվի ոչ ավել քան $\log_2 n+1$ քայլից հետո, ուրեմն համեմատությունների թիվը հավասար կլինի.

$$C(n) \approx n \log_2 n :$$

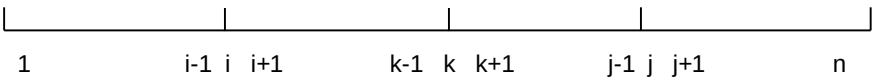
Այսինքն Ֆոն-Նեյմանի մեթոդը ապահովեց կարգավորման ալգորիթմի լավագույն արդյունքը (գնահատականը), ընդ որում տեղաշարժերի թիվը հավասար է համեմատությունների թվին:

Այս մեթոդը լրացուցիչ հիշողություն չի պահանջում (բացի անհրաժեշտ աշխատանքային հիշողությունից՝ n*p):

3.2.6. Կարգավորում ըստ տարրի տեղի

Մեթոդի նկարագրությունը. $a_1, a_2, \ldots a_n$ հաջորդականության մեջ վերցվում է հաջորդականության առաջին տարրը՝ a_1-ը և հաջորդաբար համեմատվում է հաջորդականության մնացած տարրերի հետ: Համեմատման ընթացքում a_1-ից փոքր տարրերը տեղադրվում են հաջորդականության սկզբից (տեղափոխվում են), իսկ a_1-ից մեծ տարրերը՝ հաջորդականության վերջից: Համեմատությունների ավարտից հետո a_1-ը տեղադրվում է այդ 2 մասերի միջև, որն ըստ էության իր վերջնական տեղն է, և այն այլևս չի տեղափոխվելու:

Ալգորիթմը իրականացվում է ռեկուրսիվ:



1-ին քայլում 1..n միջակայքում միջակայքի առաջին տարրը գրավում է իր տեղը. դիցուք k-րդ դիրքում, դրա համար կատարվում է $(n-1) = (n-2^1+1)$ համեմատություն,

2-րդ քայլում դիտարկվում են 1..k-1 և k+1..n միջակայքերը առանձին (եթե նրանցում տարրերի քանակը ≥2): Կիրառելով ալգորիթմն այդ 2 միջակայքերում, կատարվում է (k-1-1) + (n-k-1) =n-3= $n-2^2+1$ համեմատություն,

3-րդ քայլում դիտարկվում են 1..i-1, i+1..k-1, k+1..j-1, j+1..n միջակայքերը, յուրաքանչյուր միջակայքի առաջին տարրը իր տեղը տեղադրելու համար կպահանջվի՝

(i-1-1) + (k-1-I-1) + (j-1-k-1) + (n-j-1) = $n-2^3 + 1$ համեմատություն,

k-րդ քայլում կպահանջվի $n-2^k + 1$ համեմատություն:

Այս մեթոդով կարգավորումը կկատարվի ոչ պակաս քան ($\log_2 n+1$) և ոչ ավել քան (n-1) քայլում, հետևաբար ընդհանուր համեմատությունների քանակը՝

$$C(n) \geq (n-1) + \underbrace{(n-3)}_{\log_2 n+1} + (n-7) + \ldots + \underbrace{\left(n-2^k+1\right)}_{\log_2 n+1} + \ldots + \underbrace{\left(n-2^{\log_2 n+1}+1\right)}_{\log_2 n+1} = \sum_{i=1} \left(n-2^i+1\right) = \sum_{i=1} (n+1) - \sum_{i=1} 2^i =$$

$$= (n+1)\cdot(\log_2 n+1) - 2\cdot 2^{\log_2 n} = (n+1)\log_2 n - 2\cdot n =$$

$$= n\log_2 n + \log_2 n - 2n = n\log_2 n\left(1 + \frac{1}{n} - \frac{1}{2\log_2 n}\right) \approx C\cdot n\log_2 n$$

որտեղ C=const:

Այս ալգորիթմը լավագույնն է ըստ համեմատությունների քանակի (**Cn** $\log_2 n$) , ըստ զբաղեցրած հիշողության ծավալի (այն լրացուցիչ հիշողություն չի պահանջում և հավասար է n*p-ի) և ըստ տեղաշարժերի քանակի (մեկ համեմատությանը մեկ տեղաշարժ):

3.2.7. Կարգավորման քառակուսային մեթոդ

Մեթոդի նկարագրությունը: Դիցուք $a_1, a_2, \ldots a_n$ հաջորդականության համար n-ը լրիվ քառակուսի է՝ $n=k^2$: Հաջորդականությունը տրոհենք բլոկների, յուրաքանչյուր բլոկում k հատ տարր, կունենանք k հատ բլոկ՝

$a_1 a_2 \ldots a_k\ a_{k+1} \ldots a_{2k} \ldots a_{(k-1)k+1} \ldots a_n$:

Վերցնենք լրացուցիչ հիշողություն՝ $b_1,b_2,...,b_n$, որտեղ ստանալու ենք կարգավորված հաջորդականությունը և վերցնենք $c_1,c_2,...,c_k$ լրացուցիչ հիշողություն, որտեղ յուրաքանչյուր c_i -ն օժանդակ դաշտ է համապատասխան i-րդ բլոկի համար: Այն անվանենք զոնա.

b_1 b_2 ... b_n

a_1

a_2

... c_1

a_k

a_{k+1}

a_{k+2}

... c_2

a_{2k}

:

$a_{(k-1)k+1}$

$a_{(k-1)k+2}$

... c_k

a_n

Քառակուսային մեթոդով կարգավորման համար կատարվում են հետևյալ քայլերը.

ա) յուրաքանչյուր բլոկում գտնում ենք ամենափոքր տարրը, այն նշում ենք և ուղարկում համապատասխան c զոնան.

բ) գտնում ենք $c_1,c_2, ... ,c_k$ զոնաների մինիմումը, դիցուք դա c_j -ն է, ուղարկում ենք b_n հաջորդականության հերթական տարրի տեղը, ապա a_n հաջորդականության j-րդ բլոկում գտնում ենք մնացած տարրերի մեջ մինիմումը և նորից ուղարկում c_j զոնա,

գ) կրկնում ենք բ) քայլն այնքան, մինչև ամբողջ a_n հաջորդականությունը տեղափոխվի b_n հաջորդականության մեջ, ընդ որում b հաջորդականությունը արդյունքում լինում է կարգավորված:

b_n / a_n	-5 I	-4 II	-3 III	-1 IV	0 V	2 VI	3 VII	5 VIII	6 IX
-1									
-5	-5	-1			5				
5									
2									
-4	-4		2				6		
6									
3									
0	-3			0		3			
-3									

Քանի համեմատություն է պահանջվում այս մեթոդով կարգավորման համար:

Յուրաքանչյուր բլոկում մինիմումը գտնելու համար ըստ էտապների կկատարվի (k-1), (k-2),..,1 համեմատություն, հետևաբար ***յուրաքանչյուր բլոկում համեմատությունների քանակը՝***

$$C(k)=(k-1)+(k-2)+..+1=k(k-1)/2$$

բլոկների քանակը հավասար է k, ուրեմն բլոկներում համեմատությունների քանակը՝

$$C_b(k)=k*k(k-1)/2=k^2(k-1)/2$$

Զոնաներում նույնպես ամենափոքր տարրը գտնելու համար կատարվում են համեմատություններ, ընդ որում յուրաքանչյուր քայլում ոչ ավել k-1 համեմատություն, իսկ քայլերի քանակը n-ից ավել չէ, հետևաբար զոնաներում համեմատությունների քանակը՝

$$C_z(k)=n(k-1):$$

Ընդհանուր համեմատությունների քանակը՝

$$C(n)=C_b(k)+C_z(k)=k^2(k-1)/2+n(k-1)=n(k-1)/2+n(k-1)=3/2n(k-1)=3/2n(\sqrt{n}-1) \approx 3/2n^{3/2}$$ (որտեղ $k=n^{1/2}$):

Ստացանք՝

$$C(n) \sim 3/2n^{3/2}$$

Այս մեթոդի ժամանակ ծախսվում է $(2*n+n^{1/2}*p)$ հիշողություն:

Օրինակ, եթե մեր հաջորդականությունը ունի 100 տարր, ապա այս մեթոդով կարգավորման ժամանակ համեմատություն-

ների թիվը հավասար է $3/2*100^{3/2}=1500$, իսկ լավագույն կարգավորման ալգորիթմում՝ $n\log_2 n= 100*\log_2 100\approx 700$, պղպջակի մեթոդով $n^2/2=10000/2=5000$:

Այժմ ներկայացնենք քառակուսային մեթոդի մոդիֆիկացված տարբերակը: Էությունը նույնն է, միայն բլոկների տրոհելուց հետո, յուրաքանչյուր բլոկում նախապես բոլոր տարրերը կարգավորվում են կարգավորման որևէ լավագույն ալգորիթմով՝ յուրաքանչյուր բլոկում ամենաշատը $k\log_2 k$ համեմատություն, բլոկների քանակը հավասար է k, հետևաբար բլոկների կարգավորման համար կպահանջվի $C_b(k)=k*\log_2 k=k^2\log_2 k$ համեմատություն:

Որից հետո աշխատում է նույն քառակուսային կարգավորման ալգորիթմը զոնաների միջոցով (ընդ որում բլոկներում արդեն մինիմումը գտնելու կարիքը չկա, որովհետև նրանք արդեն կարգավորված են): Արդյունքում a_n հաջորդականության կարգավորման համար կկատարվի

$C(n)=C_b(k)+C_z(k)=k^2\log_2 k+n(k-1)=k^2\log_2 k+k^2(k-1)=k^2(\log_2 k+k-1)=n(\log_2 n^{1/2}+n^{1/2}-1)$

$C(n)\ \square\ n/2\log_2 n+n^{3/2}$ համեմատություն:

Օրինակ, երբ n=100, ապա $C(n)=100(1/2\log_2 100+100^{1/2})=$
$=100(7/2+10)=1350$, իսկ նախկին եղանակով՝ 1500:

3.2.8. Կարգավորում Շելլի մեթոդով

Շելլի մեթոդը պղպջակի մեթոդի լավացված տարբերակն է: Պղպջակի մեթոդը դանդաղ է աշխատում, որովհետև կարգավորման ժամանակ համեմատվում են հարևան 2 տարրերը, և եթե տարրը պետք է իր դիրքից տեղափոխվի k դիրքով, ապա կարող է դա կատարել միայն k քայլից հետո:

Մեթոդի նկարագրությունը. Առաջնահերթ դիտարկվում և կարգավորվում են իրարից հեռու գտնվող 2 տարրերը: Այսինքն ընտրվում է համեմատության քայլը, որը ցույց է տալիս համեմատվող տարրերի միջև եղած հեռավորությունը: $a_1,a_2,....a_n$ հա-

ջորդականության համար նախապես որպես համեմատման քայլ ընտրվում է՝ h = [n/2]:

Կարգավորումը կատարվում է էտապներով: Յուրաքանչյուր էտապում կարգավորումը կատարվում է հաջորդականության հետևյալ կարգահամարներն ունեցող տարրերի խմբերի համար.

I, I+h, I+2h, ...

որտեղ l=1,2,...,h-1 իրենից ներկայացնում է տվյալ էտապում տարրերի խմբի համարը:

Օրինակ, երբ h=5, ապա տվյալ էտապում կարգավորում կատարվում է հետևյալ խմբերում.

1,6,11,...

2,7,12,...

3,9,13,...

4,9,14,...

Յուրաքանչյուր խմբում կարգավորումը կատարվում է պղպջակի մեթոդով: Բոլոր խմբերի կարգավորումից հետո հաջորդ էտապի համար որպես համեմատման քայլ վերցվում է **h=[(h-1)/2]**, քանի դեռ h≥2: Պրոցեսի վերջին էտապում (h=1) կարգավորման ժամանակ դիտարկվում են արդեն հարևան տարրերը (այստեղ կիրառվում է իսկապես պղպջակի մեթոդը), սակայն մինչ այդ կատարված փոխատեղումների հաշվին այս էտապում փոխատեղումների քանակը էապես քչանում է:

Այս մեթոդի համար տրված է հետևյալ փորձնական գնահատականը.

C(n) □ 1/2 n$^{3/2}$, T(n) □ 1/2 n$^{3/2}$:

Այս մեթոդը լրացուցիչ հիշողություն չի պահանջում:

Օրինակ՝ 3,-7,-8,2,-3,7,24,-17,0,6; n=10;

1-ին էտապ. h=[n/2]=5,

խմբերն ըստ կարգահամարների հետևյալն են՝ 1,6; 2,7; 3,8; 4,9; 5,10: Խմբերում կարգավորում կատարելուց հետո կստանանք.

a_n= 3, -7, -17, 0, -3, 7, 24, -8, 2, 6:

2-րդ էտապ. h=[(h-1)/2]=2,

խմբերն ըստ կարգահամարների հետևյալն են՝ 1,3,5,7,9; 2,4,6,8,10: Խմբերում կարգավորում կատարելուց հետո կստանանք.

$$a_n = -17, -8, -3, -7, 2, 0, 3, 6, 24, 7:$$

3-րդ էտապ՝ h=[(h-1)/2]=1,

խմբերն ըստ կարգահամարների հետևյալն են՝ 1,2,3,4,5,6,7,8,9,10: Խմբում կարգավորում կատարելուց հետո կստանանք.

$$a_n = -17, -8, -7, -3, 0, 2, 3, 6, 7, 24:$$

Կարգավորումն ավարտված է:

3.3. Արտաքին կարգավորում գրառումների մեկ հավաքածուի համար

Արտաքին կարգավորման ալգորիթմի սահմանումից երևում է, որ կարգավորման պրոցեսում կարևոր է նաև մուտքի/ելքի գործողությունների վրա ծախսված ժամանակը:

Դիցուք մեր ինֆորմացիոն մասսիվը գրված է որևէ արտաքին կրողի վրա և պարունակում է n տարր, և դիցուք ներքին հիշողության մեջ ունենք 2m<<n ազատ տեղ (այսինքն ներքին հիշողությունում կարելի է պահել ընդամենը 2m հատ տարր, ներքին հիշողության մեջ m հատ տարրերի զբաղեցրած ծավալին անվանենք բլոկ).

1-ին բլոկ	2-րդ բլոկ
m հատ տարր	m հատ տարր

այսինքն ամբողջ մասսիվը չի տեղավորվում ներքին հիշողությանում, իսկ դա նշանակում է, որ կարգավորման ժամանակ ամբողջ մասսիվը միանգամից ներքին հիշողություն չի տեղափոխվում:

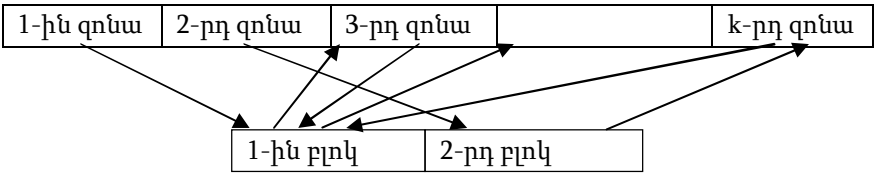
Տարրերի զբաղեցրած արտաքին հիշողությունը տրոհենք **k=n/m զոնաների**, յուրաքանչյուր զոնայի մեջ m հատ տարր:

1-ին զոնա	2-րդ զոնա	3-րդ զոնա		k-րդ զոնա
m տարր	m տարր	m տարր		m տարր

Համեմատել կարելի է ներքին հիշողության մեջ: Կարգավո-

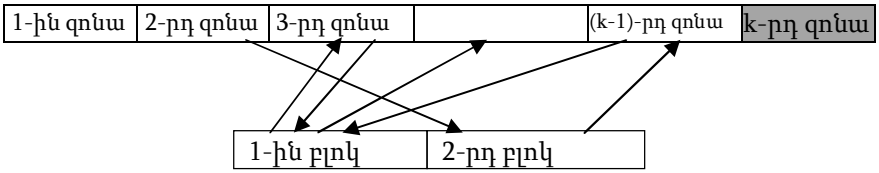
րումը իրականացվում է փուլերով։

1-ին փուլ. 1-ին զոնայի պարունակությունը տեղափոխվում է ներքին հիշողության 1-ին բլոկ, 2-րդ զոնայինը՝ 2-րդ բլոկ: Ներքին հիշողության այդ 2 բլոկների պարունակությունները կարգավորվում են ներքին կարգավորման որևէ լավագույն ալգորիթմով (օրինակ, ըստ աճման): Այնուհետև, 1-ին բլոկում կունենանք դիտարկվող տարրերի մեջ ամենափոքր տարրերը, իսկ 2-րդ բլոկում՝ մեծ տարրերը: Ապա 1-ին բլոկի պարունակությունը տեղափոխվում է 2-րդ զոնա և 3-րդ զոնայի պարունակությունը տեղափոխվում է 1-ին բլոկի մեջ: Նորից բլոկների նոր պարունակություննները կարգավորվում են և 1-ին բլոկի պարունակությունը հետ է ուղարկվում 3-րդ զոնա: Վերջին քայլում k-րդ զոնան ուղարկվում է 1-ին բլոկ, կարգավորվում են, որի արդյունքում 2-րդ բլոկում վերջապես հայտնվում են մեր ինֆորմացիոն մասսիվի ամենամեծ m հատ տարրերը, հետևաբար 2-րդ բլոկի պարունակությունը ուղարկվում է k-րդ զոնա:



Յուրաքանչյուր հաջորդ փուլում արտաքին հիշողության զոնաների քանակը պակասում է 1-ով:

2-րդ փուլ. 2-րդ զոնայի պարունակությունը տեղափոխվում է 2-րդ բլոկ, բլոկներում կատարվում է կարգավորում, 1-ին բլոկի պարունակությունը տեղափոխվում է 2-րդ զոնա, ապա 3-րդ զոնայի պարունակությունը տեղափոխվում է 1-ին բլոկ, կարգավորվում են, 1-ին բլոկը հետ է ուղարկվում 3-րդ զոնա: Այս փուլի վերջին քայլում 2-րդ բլոկի պարունակությունը տեղափոխ-վում է (k-1)-րդ զոնա:



(k-1)-րդ փուլ. Պրոցեսին մասնակցում են միայն 1-ին երկու զոնաները և ներքին հիշողության 2 բլոկները: 2-րդ զոնայի պարունակությունը տեղափոխվում է 2-րդ բլոկ, բլոկներում կատարվում է կարգավորում, որից հետո 1-ին բլոկի պարունակությունը տեղափոխվում է 1-ին զոնա, իսկ 2-րդ բլոկի պարունակությունը տեղափոխվում է 2-րդ զոնա: Այսքանով արտաքին կարգավորման ալգորիթմն ավարտում է իր աշխատանքը:

Քանի՞ համեմատություն է պահանջում արտաքին կարգավորման այս ալգորիթմը:

Ներքին հիշողության 2 բլոկներում 2m տարրերի կարգավորման համար կպահանջվի՝ $2m\log_2(2m)$ համեմատություն, հետևաբար նշված էտապներից հետո բլոկներում համեմատությունների ընդհանուր քանակը հավասար է՝

$$C(n)=(k-1)2m\log_2(2m)+(k-2)2m\log_2(2m)+\ldots+2m\log_2(2m)=$$

$$=2m\log_2(2m)\sum_{i=1}^{k-1} i = 2m\log_2(2m)\,k(k-1)/2 \approx n^2/m\log_2(2m)$$

Ստացանք՝ $\mathbf{C(n) \approx n^2/m \log_2(2m),}$ այս գնահատականից հետևում է, որ m-ը ինչքան մեծ լինի (այսինքն, ներքին հիշողությունում ինչքան շատ ազատ տեղ ունենանք), համեմատությունների քանակն այնքան փոքր կլինի:

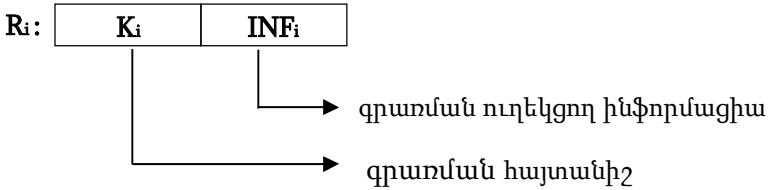
Արտաքին կարգավորման ժամանակ անհրաժեշտ է հաշվի առնել նաև արտաքին կրողին դիմումների քանակը: Յուրաքանչյուր էտապում կատարվում է և կարդալու, և գրելու գործողություններ, հետևաբար

$$\mathbf{RW(n)=2(k-1)+(k-2)+..+1=2k(k-1)/2=k(k-1) \approx n^2/m^2;}$$

որից հետևում է, որ ինչքան m-ը մեծ լինի, այնքան արտաքին կրողին դիմումների քանակը կլինի քիչ, հետևաբար ելքի/մուտքի գործողությունների վրա ծաղսվող ժամանակը կլինի փոքր:

4. ԻՆՖՈՐՄԱՑԻՈՆ ՄԱՍՍԻՎՆԵՐՈՒՄ ՏԵՂԱՓՈԽՈՒԹՅՈՒՆՆԵՐԻ ԻՐԱԿԱՆԱՑՄԱՆ ՄԻ ՄԵԽԱՆԻԶՄ

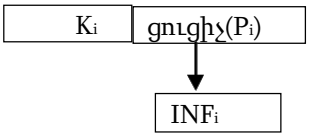
Ինֆորմացիոն մասսիվի յուրաքանչյուր R_i գրառում ունի հետևյալ կառուցվածքը՝



Գրառման ուղեկցող ինֆորմացիան կարող է կոնկրետ խնդիրներում բարդ կառուցվածք ունենալ և զբաղեցնել հիշողության մեծ ծավալ: Հետևաբար, կարգավորման ալգորիթմներում տեղափոխություններ կատարելուց անիմաստ է այդ ամբողջ ծավալով ինֆորմացիան տեղափոխել մեքենայի հիշողության մեջ տեղից տեղ, և ակնհայտ է, որ այդպիսի տեղափոխություն կատարելու ժամանակ կպահանջվի բավականին ժամանակ, որը զումարային կանդրադառնա կարգավորման ալգորիթմի ժամանակի վրա:

Դրա համար, տեղափոխման ինչպիսի՞ մեխանիզմ կարելի է առաջադրել, որ ապահովի ուղեկցող ինֆորմացիայի պահպանումը, սակայն քիչ ժամանակ պահանջի տեղափոխման համար:

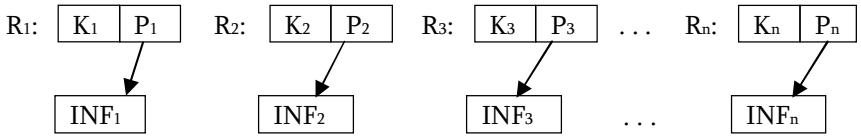
Մեխանիզմերից մեկը հետևյալն է. յուրաքանչյուր R_i գրառման կառուցվածքը ձևափոխվում է և ներկայացվում հետևյալ տեսքով՝



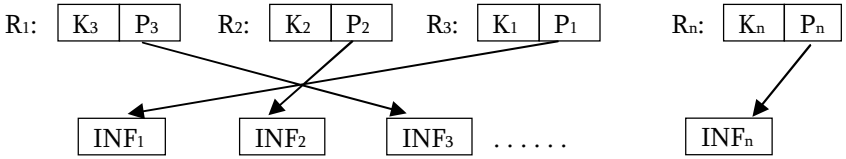
որտեղ K_i-ն R_i գրառման հայտանիշն է, իսկ ցուցիչ (P_i)-ին՝ ցուցիչ է ուղեկցող ինֆորմացիայի վրա՝ մեքենայի ներքին հիշողությունում:

Տեղափոխությունների ժամանակ բավական է տեղափոխել միայն K_i հայտանիշը և INF դաշտի ցուցիչը, որի արդյունքում ցուցիչ(P_i)-ին միշտ ցույց կտա R_i գրառման INF ուղեկցող ինֆորմացիայի տեղը, իսկ INF դաշտը կմնա նույն տեղում: Այսինքն գործողությունները կատարվում են միայն հայտանիշի և ցուցիչի հետ:

Օրինակ՝ $R_1,R_2,..,R_n$ հաջորդականության մեջ փոխել R_1 և R_3 գրառումների տեղերը.



Տեղափոխման գործողությունից հետո՝



5. ԻՆՖՈՐՄԱՑԻԱՅԻ ՊԱՀՊԱՆՄԱՆ ՅՈՒՑԱԿԱՅԻՆ ԿԱՌՈՒՑՎԱԾՔՆԵՐ

Մինչ այժմ դիտարկվել են ֆիքսված չափով տվյալների կառույցները, ինչպիսիք են միաչափ մասսիվները: Այժմ մտցնենք տվյալների դինամիկ կառույցներ, որոնք երկարում և կարճանում են (այսինքն տեղի է ունենում չափի փոփոխություն) ծրագրի կատարման ընթացքում:

Ինֆորմացիայի պահպանման դինամիկ կառույցները կիրառվում են, երբ ինֆորմացիոն մասսիվի տարրերը մեքենայի ներքին հիշողությունում անհրաժեշտ է պահել ոչ հաջորդական դաշտերում (այսինքն հիշողության մեջ գտնվում են կամայական տեղում): Այդ դեպքում ինֆորմացիոն մասսիվի յուրաքանչյուր գրառումը կարգահամարով չի որոշվում :

Այդպիսիք են տվյալների ցուցակային կառույցները, պահունակները (ստեկները), հերթերը և ծառային կառույցները:

Կապակցված ցուցակները տվյալների այնպիսի հավաքածուներ են, որում տարրերը դասավորված են «մի գծով» և տարր ավելացնելու կամ հեռացնելու գործողություններ կարող են կատարվել կապակցված ցուցակի կամայական տեղում:

Պահունակները էական դեր են խաղում օպերացիոն համակարգերում և թարգմանիչներում, տարրի ավելացման և հեռացման գործողությունները կատարվում են ստեկի վերջից, այսինքն գագաթից (LIFO սկզբունք):

Հերթերը իրենցից ներկայացնում են այսպես ասած «մի գծով» դասավորված տվյալներ, որոնցում տարրի ավելացման գործողությունը կատարվում է հերթի վերջից, իսկ հեռացման գործողությունը կատարվում է հերթի սկզբից (FIFO սկզբունք):

Ցուցակներ, նրանց տեսակները: Ցուցակը դա ինքնահասցեավորմամբ օբյեկտների վերջավոր համախումբ է, որոնք միմյանց հետ կապված են կապի ցուցիչի միջոցով:

Ցուցակը հասանելի է հատուկ ցուցիչի միջոցով, որը ցույց է տալիս ցուցակի առաջին էլեմենտի վրա, իսկ ցուցակի յուրաքանչյուր հանգույցը հասանելի է կապի ցուցիչի միջոցով, որը

գտնվում է հանգույցներում: Ցուցակի վերջին հանգույցում, ըստ ընդունված համաձայնության, կապի ցուցիչը դրվում է հավասար 0, որը ցույց է տալիս ցուցակի վերջը: Կապակցված ցուցակում տվյալները պահվում են դինամիկ, այսինքն յուրաքանչյուր հանգույց ստեղծվում է անհրաժեշտության դեպքում:

Կապակցված ցուցակները հարմար է օգտագործել այն դեպքերում, երբ տվյալների քանակը նախապես հայտնի չէ, և ընդհանրապես մշակման պրոցեսում կարող է նաև անընդհատ փոփոխվել:

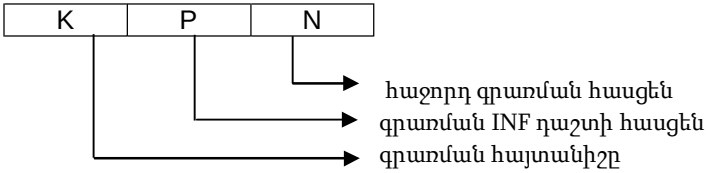
Ուշադրություն: Սովորական **ստատիկ** մասսիվի չափերը ֆիքսվում են թարգմանության ժամանակ և մնում անփոփոխ, **դինամիկ** մասսիվների չափերը թեկուզ կարելի է փոխել, սակայն նրանք պահանջում են միանվագ անհրաժեշտ չափի հիշողություն, հետևաբար եթե ներքին հիշողությունում այն բավարարված չէ (գերհագեցում), ապա կառաջանան խնդիրներ: **Կապակցված ցուցակները** գերհագեցում կառաջացնեն միայն այն ժամանակ, երբ ներքին հիշողությունը լցված կլինի ամբողջությամբ:

Ցուցակների առավելությունը դինամիկ մասսիվների նկատմամբ ակնհայտ է տարրի ավելացման և հեռացման գործողությունների ժամանակ: Այդ գործողությունների կատարման ժամանակ մասսիվներում կատարվում է տարրերի մեծ քանակությամբ տեղափոխություններ, իսկ ցուցակներում ընդհանրապես տեղափոխություններ չեն կատարվում:

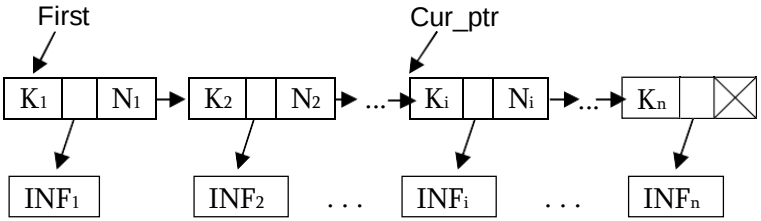
Գոյություն ունեն ցուցակների տարբեր տեսակներ: Նրանցից յուրաքանչյուրի առավելությունը պայմանավորված է տարրերի մշակման եղանակով:

5.1. Միակապ գծային ցուցականեր

Ցուցակում յուրաքանչյուր հանգույց, որը ներկայացնում է ինֆորմացիոն մասսիվի մեկ գրառում, ունի հետևյալ կառուցվածքը՝



Օրինակ՝ դիցուք ունենք $R_1,R_2,...,R_n$, ինֆորմացիոն մասսիվը, որի գրառումների քանակը հաստատուն չէ: Հիշողության մեջ այս գրառումները ցուցակային կառույցի միջոցով կներկայացվեն հետևյալ կերպ:



Գրառման N դաշտում դրված X նշանը, որը հատուկ հասցե է (NULL), և նշում է ցուցակի վերջը: Ինչպես նախկինում նշել ենք, ուղեկցող ինֆորմացիան սրանից հետո չենք դիտարկի, կդիտարկենք միայն բանալիները:

Ցուցակի հետ աշխատելուց սովորաբար օգտագործվում են 2 ցուցիչներ՝ First-ը, որը պարունակում է ցուցակի առաջին հանգույցի հասցեն, Cur_ptr-ը՝ որը պարունակում է դիտարկվող հերթական հանգույցի հասցեն:

Ցուցակի հետ աշխատանքը սկսելու համար առաջին հանգույցի հասցեն (այսինքն First-ի պարունակությունը) ուղարկվում է Cur_ptr-ի մեջ: Երբ Cur_ptr-ի մեջ հայտնվում է հատուկ NULL հասցեն, ապա դա նշանակում է, որ ցուցակի հանգույցները վերջացել են:

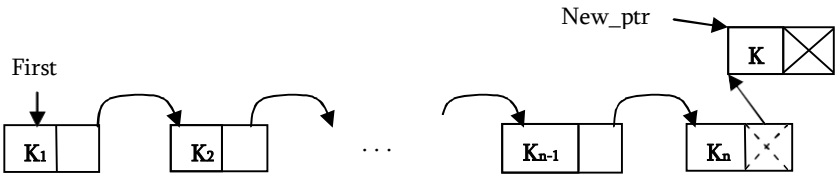
Այսպիսի ցուցակում հաջորդ գրառման մշակմանը անցնելու համար բավական է Cur_ptr ցուցիչի մեջ ուղարկել ընթացիկ գրառման N դաշտի պարունակությունը՝ Cur_ptr=Cur_ptr→N:

Ցուցակի այսպիսի կազմակերպումը բավականին պարզ է և ունիվերսալ: Նրա հետ կարող են կատարվել հետևյալ գործողությունները.

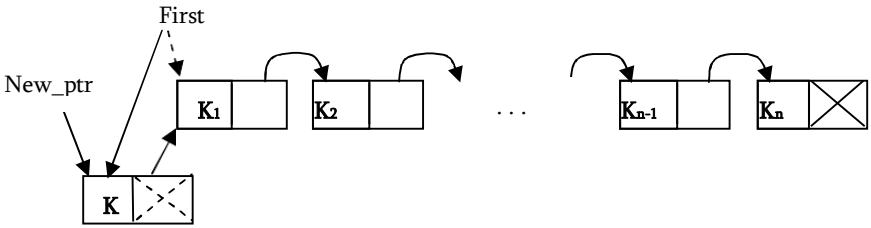
1) նոր գրառման ավելացում ցուցակի վերջում,
2) նոր գրառման ավելացում ցուցակի սկզբում,
3) նոր գրառման ավելացում ցուցակի կամայական տեղում,
4) ցուցակի առաջին գրառման հեռացում,
5) ցուցակի վերջին գրառման հեռացում,
6) ցուցակի կամայական տեղից գրառման հեռացում,
7) ցուցակի մշակում (օրինակ՝ գրառման փնտրում, գրառումների կարգավորում):

Ավելացման գործողությունների ժամանակ բավական է ստեղծել ինքը գրառումը (New_ptr), ապա որոշել կապի ցուցիչների նոր արժեքները համապատասխան հանգույցներում (*ցուցիչների փոփոխությունները նկարներում ներկայացված են գծիկներով*).

1.ցուցակի վերջում ավելացնելուց.

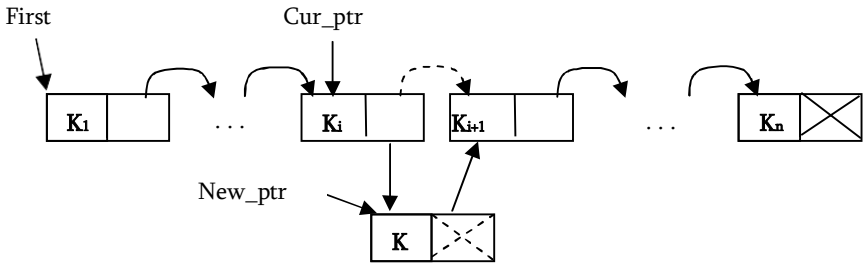


2.ցուցակի սկզբում ավելացնելուց.



3.ցուցակի կամայական տեղում ավելացնելուց (այս գործողությունն օգտագործվում է գծային կարգավորված ցուցակներում, երբ

ցուցակի հանգույցների բանալիների միջև գոյություն ունի որոշակի կարգ, օրինակ աճման կարգով. $K_1<K_2<...<K_n$).



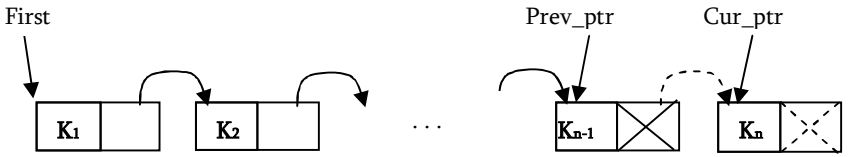
New_ptr=new Node(K);

New_ptr->N=Cur_ptr->N;

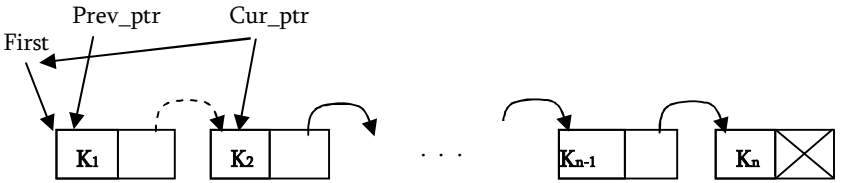
Cur_ptr->N=New_ptr;

Ցուցակից գրառման հեռացումը համեմատաբար բարդ իրականացվող գործողություն է: Սովորական զծային ցուցակում ընթացիկ հանգույցից (Cur_ptr) կարելի է անցնել միայն նրա հաջորդին, իսկ նախորդին վերադառնալու համար ոչ մի ցուցիչ գոյություն չունի, հետևաբար դա պետք է կազմակերպվի օժանդակ ցուցիչ-փոփոխականում (Prev_ptr) նախորդ հանգույցի հասցեն պահելու միջոցով, ապա որոշել կապի ցուցիչների նոր արժեքները համապատասխան հանգույցներում.

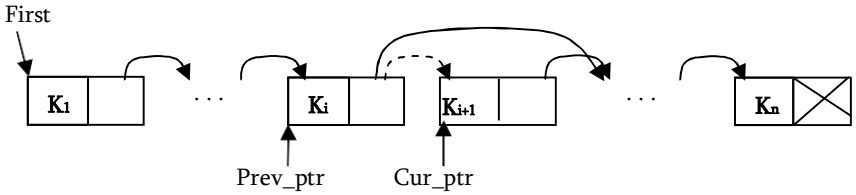
1. ցուցակի վերջին հանգույցը հեռացնելուց.



2. ցուցակի առաջին հանգույցը հեռացնելուց.



3. ցուցակի կամայական հանգույցը հեռացնելուց.




```
Node *temp=Cur_ptr;
Prev_ptr->N=Cur_ptr->N;delete temp;
```

Փնտրում սովորական գծային ցուցակում: Սովորական ցուցակում հաշտանիշի տրված արժեքն ունեցող տարրի (հանգույցի) փնտրման ալգորիթմը նույնն է, ինչոր հաջորդական փնտրման ալգորիթմը հաջորդական կազմակերպված մասսիվում: Փնտրումն սկսվում է ցուցակի առաջին հանգույցից: Ցուցակի հերթական հանգույցի հայտանիշի արժեքը համեմատվում է փնտրվող հայտանիշի հետ, եթե նրանք համընկնում են, ապա փնտրումն ավարտվում է, հակառակ դեպքում դիտարկման համար վերցվում է ցուցակի հաջորդ հանգույցը և կրկնում նկարագրված քայլերը այնքան ժամանակ, քանի դեռ ցուցակում հանգույցներ կան (չենք հասել ցուցակի վերջին): Փնտրման ալգորիթմում համեմատությունների քանակը պայմանավորված է ցուցակում հայտանիշի տրված արժեքն ունեցող հանգույցի առկայությունից. եթե փնտրվող տարրը ցուցակում կա և այն դիցուք i-րդն է, ապա կկատարվի i հատ համեմատում *(առանց ավարտի ստուգման հրամանի)*, իսկ չլինելու դեպքում համեմատությունների քանակը հավասար է ցուցակում հանգույցների քանակին:

Եթե ցուցակը կարգավորված է ապա փնտրման խնդրի գնահատականը նույն է, ինչոր հաջորդական կազմակերպմամբ մասսիվների «Փնտրում կարգավորված մասսիվում» ալգորիթմի դեպքում:

Կարգավորման խնդիրը կարգավորված գծային ցուցակում: Ցուցակում նոր տարրի ավելացման մեխանիզմը իմանալուց հե-

տո առաջ է գալիս հերթական խնդիրը. ներմուծել $R_1,R_2,...,R_n$ ինֆորմացիոն մասսիվը և **կարգավորել** ըստ հայտանիշի (աճման կարգով)։ Ելնելով ցուցակի կազմակերպման սկզբունքից, ակնհայտ է, տարրերի կարգավորումը իրականացվելու է հենց ներմուծման ժամանակ, այսինքն յուրաքանչյուր նոր տարրը ցուցակում ավելացվելու է իր տեղում, հետևաբար կարգավորման ալգորիթմի գնահատականն ըստ համեմատությունների քանակի պայմանավորված է կարգավորված ցուցակում հայտանիշի տրված արժեքն ունեցող հանգույցի փնտրման համար անհրաժեշտ համեմատությունների քանակով։ Ներկայացնենք այդ ալգորիթմը։

<u>Ալգորիթմի նկարագրությունը</u>**:** Ներմուծվում է ինֆորմացիոն մասսիվի հերթական տարրը, ապա սկսելով ցուցակի սկզբից համեմատվում են նոր ստեղծված տարրի և ցուցակի ընթացիկ հանգույցի հայտանիշները, եթե նոր տարրի հայտանիշի արժեքը փոքր է ընթացիկ հանգույցի հայտանիշի արժեքից, ապա նոր տարրը ավելացնում ենք ցուցակում նախորդ հանգույցից հետո (կամ իրենից առաջ, եթե նա առաջինն էր), հակառակ դեպքում՝ անցնում ենք ցուցակի հաջորդ հանգույցի դիտարկմանը, եթե դեռ հանգույցներ կան, եթե չկան նոր տարրը ավելացվում է ցուցակի վերջում։ Արդյունքում, ցուցակում տարրերը յուրաքանչյուր քայլից հետո մնում են կարգավորված։

Այստեղից հետևում է, որ ինֆորմացիոն մասսիվի կարգավորման համար կկատարվի՝

$$C(n)\le 1+2+...+(n-1) = n(n-1)/2 \approx n^2/2$$

համեմատություն, իսկ տեղափոխություններ գոյություն չունեն, կատարվում են միայն տեղադրումներ։

5.2. LIFO և FIFO տիպի հերթեր

Հերթերի տեսության մեջ տարբերում են 2 հիմնական տիպի հերթեր.

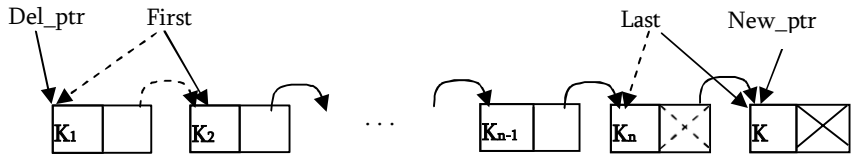
ա) FIFO («first in - first out»՝ «առաջինը եկավ, առաջինը գնաց») տիպի հերթեր, որտեղ հերթից մշակվում է այն տարրը, որը առաջինն է հերթում,

բ) LIFO («last in-first out»՝ վերջինը եկավ, առաջինը գնաց) տիպի հերթեր, որտեղ հերթից մշակվում է այն տարրը, որը վերջինն է հերթում (ստեկ):

Կդիտարկենք այս հերթերի իրականացման ցուցակային տարբերակը:

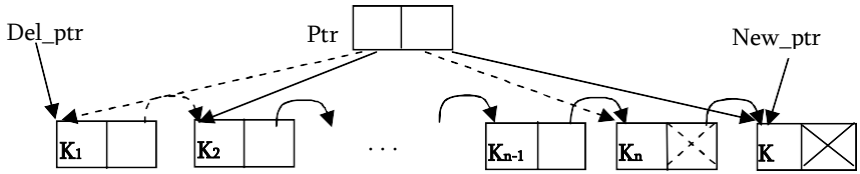
1) Կազմակերպման 1-ին ձև

FIFO տիպի հերթի իրականացումը պահանջում է ցուցակային այնպիսի կազմակերպում, որի համար պետք է հասանելի լինի ցուցակի (հերթի) և սկիզբը *(First ցուցիչը)*, և վերջը *(Last ցուցիչը):* Նոր հանգույցը (New_ptr) ավելանում է հերթի վերջից, և հերթական հանգույցը հեռացվում է (Del_ptr) հերթի սկզբից (հասցեների փոփոխությունները ներկայացված են կետ-գծերով).



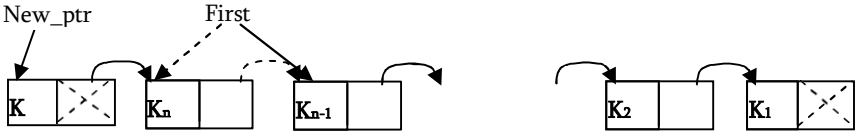
2) Կազմակերպման 2-րդ ձև

FIFO տիպի հերթի հանգույցների ավելացման և հեռացման կարգը մնում է նույնը, փոխվում է հերթի վերջը և սկիզբը ցույց տվող ցուցիչի (Ptr) կառուցվածքը, այն դառնում է կազմված 2 դաշտից (2 ցուցիչներից), առաջին ցուցիչը ցույց է տալիս ցուցակի սկիզբը, իսկ 2-րդ ցուցիչը ցույց է տալիս ցուցակի վերջը.



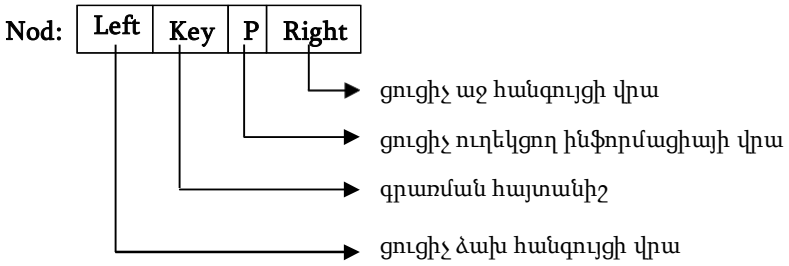
Հերթը նման է խանութում մարդկանց հերթին, որտեղ նոր հաճախորդը տեղ է վերցնում վերջում և սպասում, մինչև իրեն կսպասարկեն, իսկ սպասարկումը կատարվում է հերթի սկզբից՝ առաջին տեղում կանգնած մարդուց:

LIFO տիպի հերթը *(ստեկը)* պահանջում է ցուցակային այնպիսի կազմակերպում, որի համար կարևոր է ցուցակի վերջը: Նոր տարրը (New_ptr) ավելանում և հեռանում է (դուրս է մղվում) վերջից՝ *ստեկի գագաթից First*: Ստեկի վերջին հանգույցը ցույց է տալիս *ստեկի հատակը*: Ստեկի հետ աշխատելու հիմնական անդամ-ֆունկցիաներն են push(ավելացնել ստեկի գագաթում) և pop(հեռացնել, հանել ստեկի գագաթից) ֆունկցիաները:



5.3. Երկկողմանի գծային ցուցակ

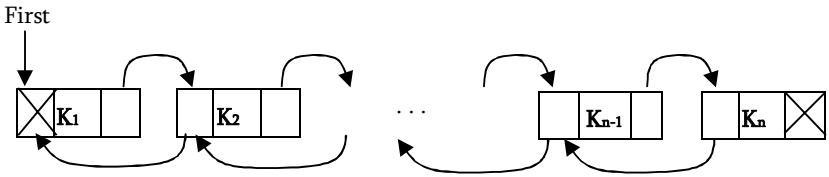
Ցուցակների հետ աշխատելուց հաճախակի հարկ է լինում այդ պահին մշակել հերթական հանգույցի ձախ և աջ հարևան հանգույցները: Դրա համար ցուցակի յուրաքանչյուր հանգույցի կառուցվածքը փոխում ենք հետևյալ կերպ՝




```
Class Node
{
```

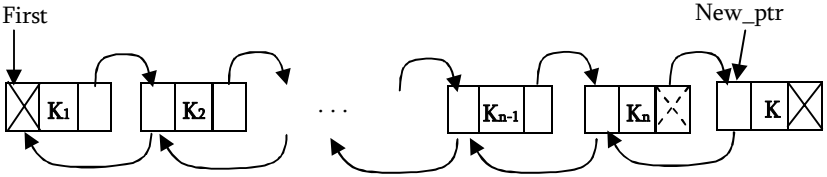
```
    int key;
    INF *p;
    Node *Left,*Right;
- - - - - - - - - - - -
}
```

Հանգույցների այսպիսի կառուցվածքի դեպքում ցուցակը կունենա հետևյալ տեսքը.

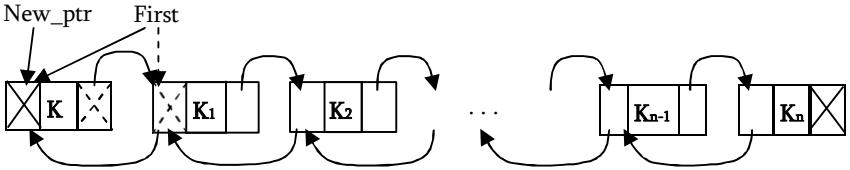


Երկկողմանի ցուցակների համար իրականացվում են սովորական միակապ գծային ցուցակների համար ներկայացված բոլոր գործողությունները: Ավելացման գործողությունների ժամանակ նորից բավական է ստեղծել նոր հանգույցը (New_ptr)՝ բանալու K արժեքով, ապա որոշել կապի ցուցիչների նոր արժեքները համապատասխան հանգույցներում, սակայն արդեն ունենալով երկկողմանի ցուցիչներ.

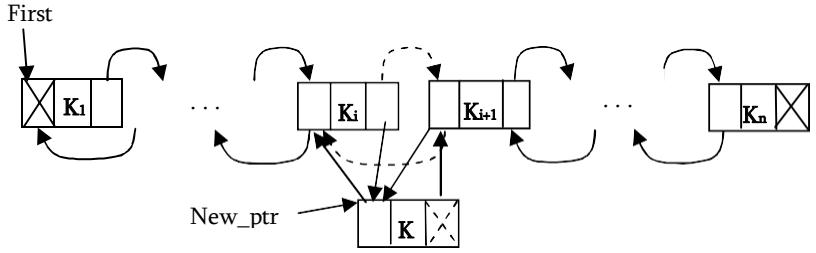
1. ցուցակի վերջում ավելացնելուց.



2. ցուցակի սկզբում ավելացնելուց.

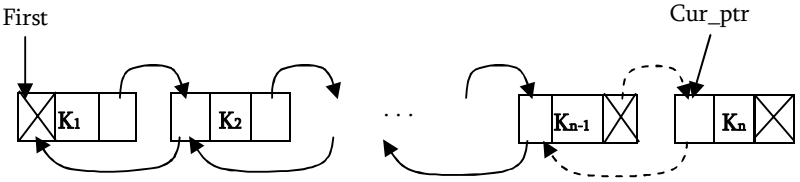


3.ցուցակի կամայական տեղում ավելացնելուց (այս գործողությունն օգտագործվում է գծային կարգավորված ցուցակներում, երբ ցուցակի հանգույցների բանալիների միջև գոյություն ունի որոշակի կարգ, օրինակ աճման կարգով. $K_1<K_2<...<K_n$).

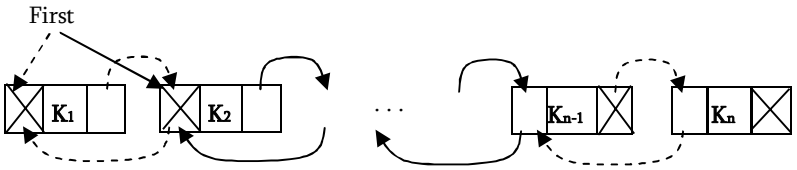


Երկկողմանի գծային ցուցակում ընթացիկ հանգույցից (Cur_ptr) կարելի է անցնել և նրա հաջորդին, և նրա նախորդին, որը հեշտացնում է հեռացման ժամանակ նոր կապերի որոշման խնդիրը.

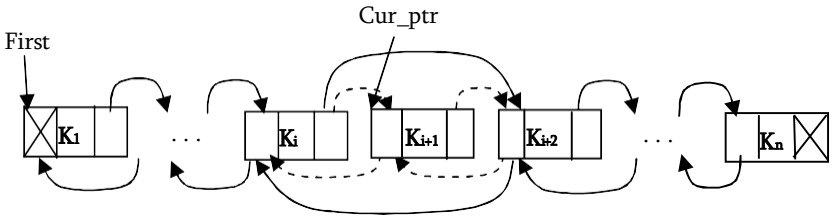
1. ցուցակի վերջին հանգույցը հեռացնելուց.



2. ցուցակի սկզբից հանգույցը հեռացնելուց.



3. ցուցակի կամայական հանգույցը հեռացնելուց.

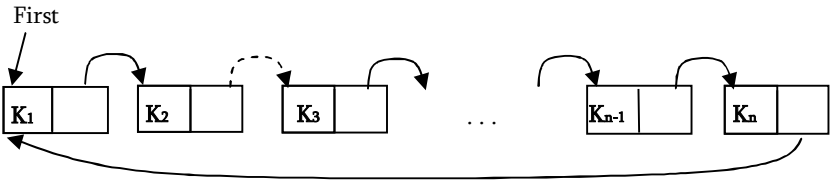


5.4. Ցիկլիկ գծային ցուցակ

Սովորական գծային ցուցակից տարբերվում է նրանով, որ ցուցակի վերջին հանգույցը ցույց է տալիս առաջին հանգույցի վրա, և ակնհայտ է ցուցակի First ցուցիչը սահում է հանգույցների վրայով, ցույց տալով ընթացիկ հանգույցի տեղը.

- որից հետո պետք է ավելացնել նոր հանգույցը,
- որին պետք է հեռացնել:

Այսինքն ցիկլիկ ցուցակում ցուցակի սկիզբ և վերջ հանդիսանում է First ցուցիչի ցույց տված հանգույցը:



6.ԻՆՖՈՐՄԱՑԻԱՅԻ ՊԱՀՊԱՆՄԱՆ ԾԱՌԱՅԻՆ ԿԱՌՈՒՑՎԱԾՔՆԵՐ

Ներկայացնենք ծառային կառույցը:

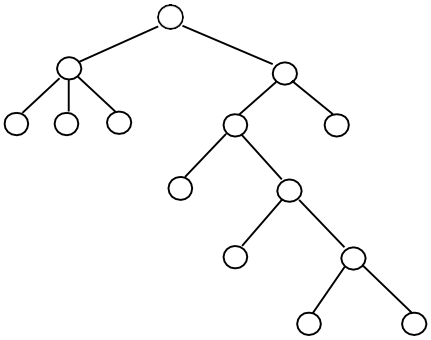
Ծառ-գրաֆի գագաթների բազմությունից առանձնացնենք մեկը, որին անվանենք ծառի **արմատ:**

Ծառի u գագաթից v գագաթին տանող գագաթների և ճյուղերի հաջորդականությունը կոչվում է u և v գագաթները միացնող **ճանապարհ**:

Երկու գագաթները միացնող ճանապարհի մեջ մտնող կողերի քանակը կոչվում է **ճանապարհի երկարություն**:

Կասենք ծառի **x գագաթը գտնվում է k-րդ մակարդակի** վրա, եթե արմատից այդ գագաթին բերող ճանապարհի երկարությունը հավասար է (k-1):

Ծառի արմատը գտնվում է 1-ին մակարդակի վրա:



Ծառը կոչվում է **m-ար ծառ**, եթե յուրաքանչյուր գագաթից դուրս եկող կողերի քանակը չի գերազանցում m-ին, և գոյություն ունի գոնե մեկ գագաթ, որից դուրս է գալիս ճիշտ m հատ կող:

Վերևում ներկայացված ծառը 3-ար ծառ է:

Եթե m=2, ապա ծառը կոչվում է **բինար ծառ:**

Կասենք m-ար ծառի x գագաթը **ածի գագաթ է և ունի ածի ուղղություն**, եթե x գագաթից կարելի տանել նոր կող առանց ծառի արությունը խախտելու:

Գագաթն ունի այնքան աճի ուղղություն, ինչքան կող կարելի է ավելացնել այդ գագաթին:

Խնդիր: *Քանի՞ աճի ուղղություն ունի n գագաթանի m-ար ծառը:*

n գագաթանի m-ար ծառում հնարավոր աճի ուղղություններ քանակը հավասար է n*m, մյուս կողմից, n գագաթ ունեցող ծառը ունի n-1 կող, այստեղից հետևում է, որ այդ ծառի աճի ուղղությունների քանակը հավասար է՝

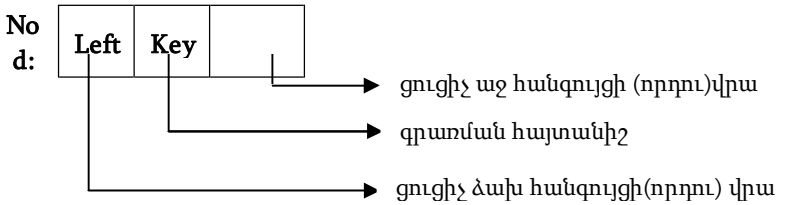
$$n*m-(n-1)=n(m-1)+1:$$

Վերևում բերված ծառի համար.

m=3, n=14, հետևաբար աճի ուղղությունների քանակը հավասար է՝ 14(3-1)+1=29:

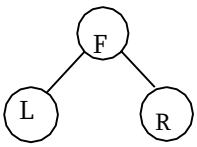
6.1. Սովորական բինար ծառեր

Բինար ծառային կառույցում ծառի յուրաքանչյուր հանգույց (գագաթ) ունի հետևյալ կառուցվածքը.



և ծառի յուրաքանչյուր գագաթ պարունակում է դիտարկվող $R_1, R_2, ..., R_n$ ինֆորմացիոն մասսիվի մեկ տարր:

n տարր պարունակող ինֆորմացիոն մասսիվը բինար ծառի տեսքով ներկայացումից հետո մեր ծառը կունենա n հատ գագաթ: Բինար ծառի կառուցման համար նախապես նրա գագաթների միջև մտցնենք ***որոշակի կարգ.*** ընթացիկ գագաթի (***F-հայր***) ձախ ճյուղի բոլոր գագաթների հայտանիշների



արժեքները փոքր են իր հայտանիշի արժեքից, իսկ աջ ճյուղի բոլոր գագաթների հայտանիշների արժեքները մեծ են իր հայտանիշի արժեքից՝ $L< F<R$:

Այսպիսի ծառերը կոչվում են *որոնման բինար ծառեր*:

Բինար ծառի կառուցման ալգորիթմը: Դիցուք ունենք $R_1, R_2, ..., R_n$ հաջորդա-կանությունը: Հաշվի առնելով վերևում նշված կարգը կառուցենք ծառը հետևյալ ալգորիթմով.

1) հաջորդականության առաջին (R_1) տարրը տեղադրում ենք ծառի **արմատում,** ապա հաջորդականության յուրաքանչյուր հաջորդ R_i տարրի ավելացման համար սկսելով ծառի արմատից իրականացնում ենք հետևյալ ռեկուրսիվ սխեման.

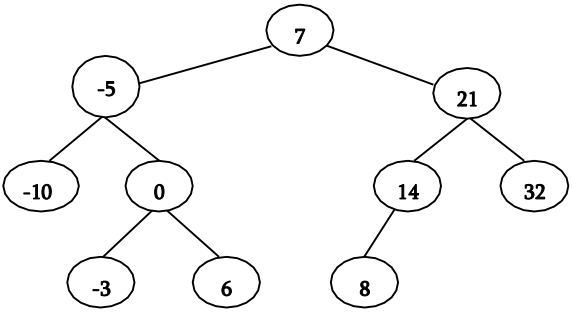
2) ծառի **ընթացիկ** գագաթի հայտանիշի արժեքը (K) համեմատում ենք **հերթական** R_i տարրի(հանգույցի) հայտանիշի արժեքի հետ ($R_i.K$).

ա) եթե $K>R_i.K$ և ծառի ընթացիկ գագաթը ձախ որդի չունի, ապա այդ գագաթին ձախ կողմից կախում ենք նոր գագաթ, որում տեղադրում ենք R_i տարրը, իսկ եթե ձախ որդի ունի, ապա դիտարկման համար վերցնում ենք այդ ձախ որդին և կրկնում ենք 2-րդ կետի ալգորիթմը:

բ) եթե $K<R_i.K$ և ծառի ընթացիկ գագաթը աջ որդի չունի, ապա այդ գագաթին աջ կողմից կախում ենք նոր գագաթ, որում տեղադրում ենք R_i տարրը, իսկ եթե աջ որդի գոյություն ունի, ապա դիտարկման համար վերցնում ենք այդ աջ որդի հանդիսացող գագաթը և կրկնում ենք 2-րդ կետի ալգորիթմը:

3) 2-րդ կետի ալգորիթմը կրկնում ենք այնքան, մինչև հաջորդականության բոլոր n տարրերը տեղադրվեն բինար ծառի գագաթներում:

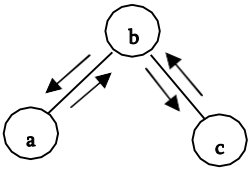
7, -5, 0, -10, 21, 14, 6, 32, 8, -3 հաջորդականության համար կառուցենք նրա բինար ծառը.



Ինչպես տեսնում ենք ծառի յուրաքանչյուր գագաթ գտնվում է որևէ k մակարդակում, հետևաբար այդ գագաթին հասնելու համար կկատարվի k համեմատում, ինչը ապահովվում է հայտանիշների արժեքների միջև մտցված կարգի հաշվին:

Ինֆորմացիոն մասսիվը բինար ծառի տեսքով ներկայացնելուց հետո դիտարկենք փնտրման և կարգավորման խնդիրները:

Կարգավորումը բինար ծառում: Բինար ծառերի համար կարգավորման խնդիրը առանձնապես հետաքրքրություն չի ներկայացնում, ծառը կառուցվում է տվյալների ներմուծման ժամանակ՝ արդեն կարգավորված: Քանի որ նպատակը կարգավորված հաջորդականության արտածումն է, իսկ ծառում կարգավորվածությունը բացահայտ չի երևում, հետևաբար բինար ծառերի համար ուսումնասիրման տեսանկյունից իրենից հետաքրքրություն է ներկայացնում ***<u>ծառի շրջանցումը</u>***, որը թույլ է տալիս ծառի գագաթներից նորից ստանալ հաջորդականություն, արդեն կարգավորված ըստ հայտանիշի:



Ներկայացնենք բինար ծառի շրջանցման մի քանի հայտնի ալգորիթմներից մեկը՝ բինար ծառի սիմետրիկ շրջանցման ալգորիթմը:

Այն ռեկուրսիվ ալգորիթմ է: Ըստ ծառի գագաթների միջև գոյություն ունեցող կարգի դուրս գրելով սկզբից ընթացիկ գագաթի ձախ որդի հանդիսացող գագաթի արժեքը, ապա իրեն արժեքը, իսկ հետո աջ որդի հանդիսացող գա-

գաթի արժեքը կստանանք կարգավորված տարրեր, օրինակ՝ a b c:

Շրջանցումն սկսվում է ծառի արմատից, ապա ծառի յուրաքանչյուր ընթացիկ գագաթի համար կատարվում է հետևյալը.

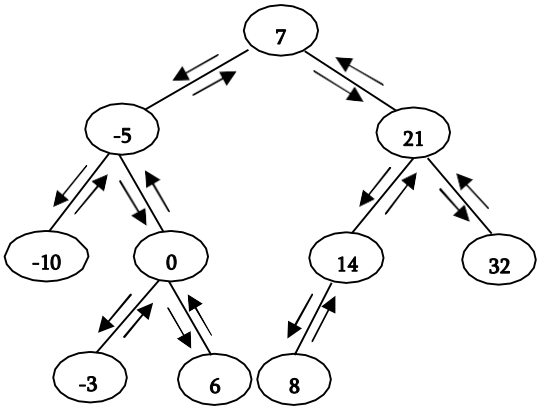
ա) եթե ընթացիկ գագաթը ձախ որդի ունի, ապա դիտարկման համար վերցնում ենք ձախ որդի հանդիսացող գագաթը և այսպես շարունակում ենք այնքան, մինչև հանդիպենք գագաթի որն այլևս ձախ որդի չունի և այդ գագաթի հայտանիշի արժեքն արտածում ենք որպես կարգավորված հաջորդականության հերթական տարր,

բ) դիտարկման համար վերցնում ենք այդ գագաթի աջ որդի հանդիսացող գագաթը, եթե այն գոյություն ունի նրա վրա կիրառում ենք ա) կետի քայլը,

գ) եթե ընթացիկ գագաթն աջ որդի չունի, ապա իրականացնում ենք վերադարձ հայր գագաթին,

դ) վերադարձի ժամանակ, եթե ընթացիկ գագաթն իր հայր գագաթի ձախ որդին է, ապա հայր գագաթի հայտանիշի արժեքն արտածում ենք որպես կարգավորված հաջորդականության հերթական տարր և դիտարկման համար վերցնելով այդ հայր գագաթն անցնում ենք բ) կետի քայլի իրականացմանը,

ե) եթե վերադարձի ժամանակ ընթացիկ գագաթն իր հայր գագաթի աջ որդին է, ապա դա նշանակում է, որ հայր գագաթի հայտանիշի արժեքն արդեն արտածված է և եթե այդ գագաթը ծառի արմատն է, ապա ծառի շրջանցումն ավարտված է, իսկ եթե ծառի արմատը չէ, ապա իրականացնում



ենք վերադարձ նրա հայր գագաթին և անցնում դ) կետի քայլի իրականացմանը:

Այս բինար ծառի շրջանցման արդյունքում կստանանք հետևյալ հաջորդականությունը.
-10, -5, -3, 0, 6, 7, 8, 14, 21, 32 և ինչպես տեսնում ենք այն կարգավորված է:

Այսինքն կարգավորված հաջորդականության ստացումը ապահովվում է ծառի կառուցման ժամանակ:

Փաստորեն կարգավորման խնդիրը տրոհվում է 2 մասի, որի արդյունքում համեմատությունների քանակը պայմանավորված է.

1) համեմատություններով, որոնք կատարվում են ծառի կառուցման ժամանակ՝ $C_կ$,

2) համեմատություններով, որոնք կատարվում են ծառի շրջանցման ժամանակ՝ C_2:

Հետևաբար՝ $C(n) = C_կ + C_2$:

Ծառի կառուցման ժամանակ համեմատությունների քանակը պայմանավորված է հերթական գագաթի ավելացման պահին ծառի մակարդակների քանակով. վատագույն դեպքում հաջորդականության k-րդ տարրի ավելացման ժամանակ եթե արդեն ծառն ունի (k-1) մակարդակ, ապա կկատարվի k-1-ից ոչ ավել համեմատությունն: Հետևաբար, n տարր ունեցող հաջորդականության դեպքում համեմատությունների քանակը ծառի կառուցման ժամանակ կլինի՝

$$C_կ \le 1+2+\ldots+(n-1) = \frac{n(n-1)}{2} \approx \frac{n^2}{2}$$

Շրջանցման ժամանակ կատարվում են ոչ թե հայտանիշների համեմատման գործողություններ, այլ միայն աջ կամ ձախ որդի ունենալու, հայր գագաթ լինելու ստուգման գործողություններ, որոնց մենք հաշվի չենք առել նաև կառուցման ժամանակ համեմատությունների քանակը հաշվելուց, հետևաբար $C_2=0$;

Ուրեմն՝ $C(n) \le \frac{n^2}{2}$;

Փնտրման խնդիրը բինար ծառում: Ամենաշատը քանի համեմատում կկատարվի բինար ծառում հայտանիշի տրված արժեքն ունեցող գագաթի փնտրման համար:

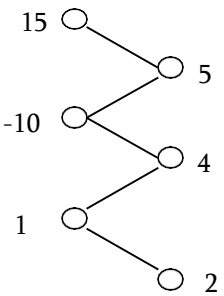
Դիցուք հաջորդականությունը ներկայացված է բինար ծառի տեսքով, որն ունի k մակարդակ (k բարձրություն), հետևաբար k-րդ մակարդակի յուրաքանչյուր գագաթին հասնելու համար կկատարվի k համեմատում: Այսինքն, ծառի վերջին մակարդակի համարը որոշում է համեմատությունների մաքսիմալ քանակը: Եթե n գագաթանի սովորական բինար ծառն ունի k հատ մակարդակ, ապա փնտրման խնդրի համար համեմատությունների քանակը կլինի՝

$$C(n) \le k$$

Այստեղից հետևում է, որ ինչքան ծառի վերջին մակարդակի համարը փոքր լինի, այնքան փնտրման ժամանակ կկատարվեն քիչ համեմատություններ:

Նկատենք, որ սովորական բինար ծառը փնտրման խնդրի համար լավագույն արդյունք չի տալիս, որովհետև կլինեն հաջորդականություններ, որոնց համապատասխան բինար ծառը կունենա ***n գագաթ և n մակարդակ***:

Օրինակ՝ -15, 5, -10, 4, 1, 2 հաջորդականության համար կառուցենք նրան համապատասխան բինար ծառը:



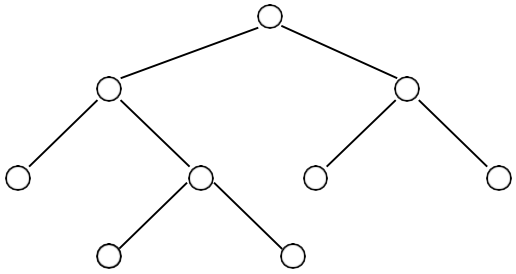
Տեսնում ենք, որ այս բինար ծառն ունի 6 գագաթ և 6 մակարդակ, հետևաբար հայտանիշի “2” արժեքն ունեցող գագաթի փնտրման համար կկատարվի 6 համեմատում:

Սովորական բինար ծառի դիտարկման արդյունքում գալիս ենք հետևյալ եզրահանգման. <u>համեմատությունների քանակը խիստ կախված է ծառի մակարդակների քանակից:</u>

Խնդիր: *Ինչպիսի՞ բինար ծառեր կարելի է առաջարկել, որոնցում ծառի մակարդակների քանակը փոքր է, և ինչպես է մակարդակների քանակը կախված ծառի գագաթների քանակից:*

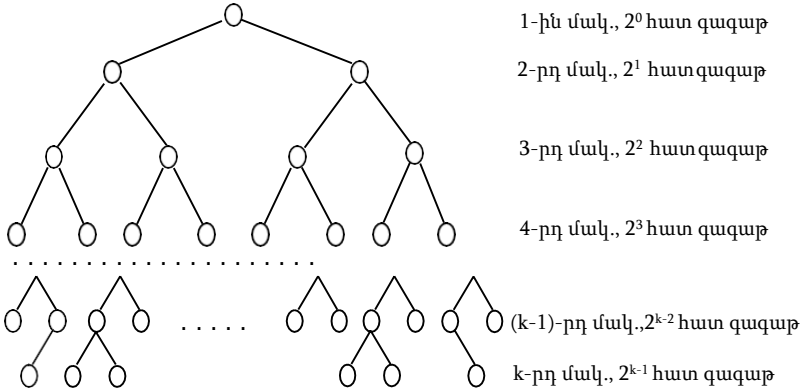
6.2. Հավասարեցված ծառեր

Բինար ծառը կոչվում է հավասարեցված. եթե նրա ածի գագաթները գտնվում են միայն վերջին և նախավերջին մակարդակների վրա:



Նշված ծառում ածի գագաթները գտնվում են 3-րդ և 4-րդ մակարդակների վրա:

Խնդիր: Դիցուք ունենք բինար հավասարեցված ծառ, որը պարունակում է n գագաթ: Որ՞ն է նրա վերջին մակարդակի համարը, այն արդյոք կախված է n-ից, թե՞ ոչ:



Ապացույց: Դիցուք վերջին մակարդակի համարը **k**-ն է: Գտնենք կապը n-ի և k-ի միջև:

Ածի գագաթները գտնվում են (k-1) և k մակարդակների վրա, իսկ գագաթների n քանակը բավարարում է հետևյալ անհավասարմանը.

$$\underbrace{2^0+2^1+\cdots+2^{k-2}}_{I} < n \le \underbrace{2^0+2^1+\cdots+2^{k-1}}_{II}$$

I գումարը որոշում է բոլոր այն գագաթների քանակը , որոնք գտնվում են մինչև (k-1)–րդ մակարդակի վրա, ընդ որում այդ բոլոր գագաթները կան, քանի որ ծառը հավասարեցված է, իսկ II գումարը որոշում է բոլոր այն գագաթների քանակը, որոնք գտնվում են և կարող էին գտնվել k-րդ մակարդակի վրա:

Ստանում ենք.

$$2^{k-1}-1 < n \le 2^k-1,$$

$$2^{k-1} \le n < 2^k,$$

որտեղից ստանում ենք՝

$$\log_2 n < k \le \log_2 n+1$$

Հետևաբար, հավասարեցված ծառի ամենավերջին մակարդակի համարը (k-ն), որը որոշում է նաև ամենաշատ համեմատությունների քանակը հայտանիշի տրված արժեքն ունեցող գագաթի փնտրման ժամանակ, կախված է ծառի n գագաթների քանակից հետևյալ առնչությամբ՝

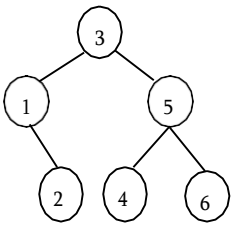
$$\log_2 n < k \le \log_2 n+1$$

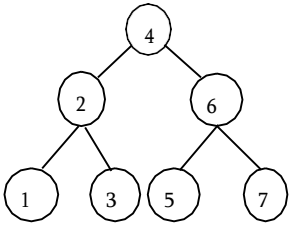
Այսպիսով, $k = \lceil \log_2 n \rceil$, վատագույն դեպքում կպահանջվի k հատ համեմատություն, և համեմատությունների քանակի համար ստացել ենք փնտրման լավագույն ալգորիթմի գնահատականը: Հավասարեցված ծառի համար փնտրումը փաստորեն իրականացվում է հնարավոր ամենաքիչ քայլերում:

Սակայն ծառերը ստատիկ օբյեկտներ չեն: Նկատենք, որ հավասարեցված ծառի ածի գագաթներից նոր գագաթներ կախելիս կարող է խախտվել նրա հավասարեցվածությունը, որի արդյունքում անհրաժեշտ է լինում ծառը վերակառուցել՝ նորից հավասարեցված դարձնելու համար: Իսկ հավասարեցված ծառի վերա-

կառուցումը բավականին աշխատատար է: Հավասարեցված ծառի վերակառուցման ալգորիթմ գոյություն չունի:

Օրինակ՝ տրված հավասարեցված ծառին ավելացնել հայտանիշի 7 արժեքն ունեցող գագաթ: Նոր գագաթը պետք է ավելացվի 6 արժեքն ունեցող գագաթից, որն ակնհայտորեն խախտում է ծառի հավասարեցվածությունը, հետևաբար այն պետք է վերակառուցել, որի արդյունքում ստանում ենք հավասարեցված ծառ, որի ոչ մի գագաթ, ինչպես տեսնում ենք, իր նախկին տեղում չի մնացել:





6.3. Բալանսավորված (հավասարակշռված) ծառեր

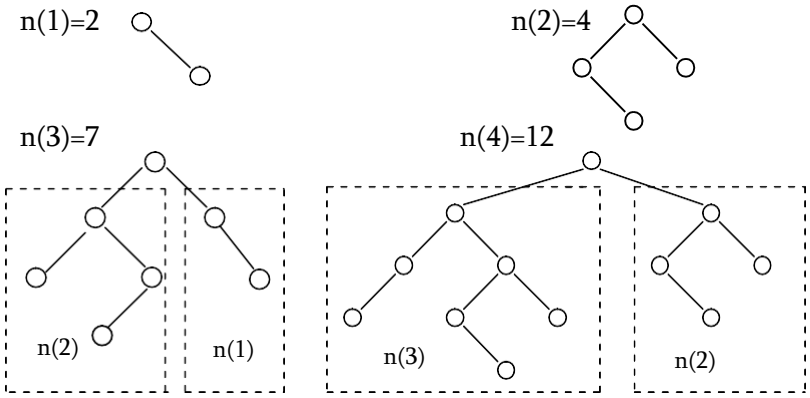
Որևէ գագաթից դուրս եկող *ճյուղ* ասելով կհասկանանք այդ գագաթը որպես արմատ դիտարկելով ստացվող ենթածառը:

ճյուղի երկարություն ասելով կհասկանանք այդ ճյուղի վերջին մակարդակի և ճյուղի սկզբնական գագաթի մակարդակի միջև եղած տարբերությունը, այլ կերպ ասած. ճյուղի երկարությունն ասելով կհասկանանք այդ ճյուղի ամենաերկար ճանապարհի երկարությունը:

Բինար ծառը կանվանենք *բալանսավորված*, եթե այդ ծառի յուրաքանչյուր գագաթից դուրս եկող աջ և ձախ ճյուղերի երկարությունների տարբերությունը մոդուլով չի գերազանցում մեկին:

$n(\ell)$-ով նշանակենք ℓ երկարությամբ ճյուղ պարունակող բալանսավորված ծառի մինիմալ գագաթների թիվը:

Օրինակ՝ $n(0)=1$, այս ծառը պարունակում է միայն արմատը:



Ստացվում է (ℓ -1) երկարությամբ ճյուղը ℓ երկարության ճյուղ դարձնելու համար վերևից ավելացվում է 1 գագաթ (հայրը), որին ավելացվում է ևս մեկ որդի՝ (ℓ -2) երկարությամբ ճյուղով:

Այսինքն տեղի ունի հետևյալը՝

$$n(\ell)= n(\ell-1) + n(\ell-2) + 1, \text{ երբ } \ell \geq 3:$$

$\square$ -ով նշանակենք $x^2=x+1$ քառակուսի հավասարման դրական արմատը՝ $\square = \frac{\sqrt{5}+1}{2} \approx 1,65$

Ապացուցենք, որ $n(\ell) \geq \square^{\ell+1}$, ℓ =3,4,... համար:

Ապացույց. Ապացուցենք ինդուկցիայի մեթոդով:

ℓ =3 դեպքում n(3)=7 , $\square^4 = \left(\frac{\sqrt{5}+1}{2}\right)^4 = \frac{7+3\sqrt{5}}{2} = 3,5+1,5\sqrt{5} \leq 7$

ℓ =4 դեպքում n(4)=12 , $\square^5 = \left(\frac{\sqrt{5}+1}{2}\right)^5 = \frac{11+5\sqrt{5}}{2} \leq 12$

Դիցուք $n(\ell) \geq \square^{\ell+1}$ անհավասարությունը ճիշտ է բոլոր ℓ -երի համար: Ապացուցենք, որ այն ճիշտ է նաև (ℓ +1)-ի համար, այսինքն տեղի ունի $n(\ell+1) \geq \square^{\ell+2}$:

Ունենք $n(\ell+1) = n(\ell) + n(\ell-1)+ 1$,

$n(\ell+1) \geq \square^{\ell+1}+\square^{\ell}+1=\square^{\ell}(\square+1)+1=\square^{\ell}.\square^2+1=\square^{\ell+2}+1 \geq \square^{\ell+2}$

Օգտվեցինք այն հանգամանքից , որ եթե $\square$ -ն $x^2=x+1$ հավասարման արմատն է, ապա $\square^2=\square+1$:

Ստացանք , որ $n(\ell+1) \geq \square^{\ell+2}$: Ինչը պետք էր ապացուցել:

Մեր ծառը պարունակում է n գագաթ, իսկ $n(\ell)$-ը ℓ երկարության ճյուղ ունեցող ծառի մինիմալ գագաթների թիվն է, հետևաբար

$$n \geq n(\ell) \geq \square^{\ell+1},$$

$$n \geq \square^{\ell+1}$$

$$\ell+1 \leq \log_{\square} n = \frac{\log_2 n}{\log_2 \square} = \log n \times \log 2$$

$$\log_{\square} 2 \approx 1{,}44$$

ստանում ենք՝ $\ell+1 \leq 1{,}44 \times \log_2 n$:

Եթե բալանսավորված ծառի ամենաերկար ճյուղի երկարությունը ℓ է, ապա այդ ծառի k վերջին մակարդակի համարը՝ k = ℓ +1 և ստանում ենք, որ հայտանիշի տրված արժեքն ունեցող գագաթի փնտրման համար n գագաթ ունեցող բալանսավորված ծառում համեմատությունների քանակը կլինի՝

$$C(n) \leq 1{,}44 \times \log_2 n$$

Այսպիսով, այս արդյունքը զիջում է հավասարեցված ծառի համար ստացած արդյունքին: Սակայն բալանսավորված ծառը հետաքրքրում է նրանով, որ նոր գագաթ ավելացնելուց հետո, եթե խախտվում է ծառի բալանսավորվածությունը, ապա նրա վերակառուցումը դյուրին է և գոյություն ունի բալանսավորված ծառի կառուցման պարզ ալգորիթմ:

Բալանսավորված ծառի կառուցման ալգորիթմը: Բալանսավորված ծառի գագաթները բնութագրենք 00 , 01 կամ 10 զույգերով, որոնք որոշում են տվյալ գագաթի համապատասխանաբար ձախ և աջ ճյուղերի երկարությունների տարբերությունը, այսինքն 00-ն ցույց է տալիս, որ աջ և ձախ ճյուղերի երկարությունները հավասար են, 01-ը ցույց է տալիս, որ աջ ճյուղը երկար է ձախ ճյուղից, իսկ 10-ն՝ հակառակը: Ընդ որում ընդունենք, որ եթե գագաթը չունի աջ կամ ձախ որդի, ապա համապատասխան ճյուղի երկարությունը հավասար է 0:

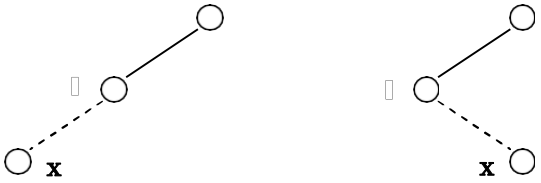
Բալանսավորված ծառում նոր գագաթի ավելացումը կատարվում է հետևյալ կերպ.

1) Նոր գագաթը կախում ենք այնպես, որ պահպանվի ծառի բինարությունը և նրան վերագրում ենք 00 բնութագիր:

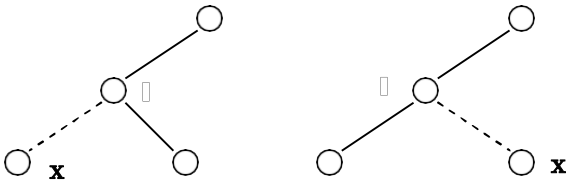
2) Կատարում ենք ծառի գագաթների բնութագրերի փոփոխում: Փոփոխվում են միայն արմատից նոր գագաթին բերող ճանապարհի վրա ընկած գագաթների բնութագրերը: Այդ ճանապարհն անվանենք «նշված» ճանապարհ: Բնութագրերի փոփոխման արդյունքում էլ պարզվում է ծառի վերակառուցման անհրաժեշտությունը:

Բնութագրերի փոփոխումը կատարվում է այսպես. նախ ուղղվում է նոր x գագաթի □ հայր գագաթի բնութագիրը: □ -ի համար հնարավոր են հետևյալ բնութագրերը՝ ա) 00 բ) 01 կամ 10:

Առաջին դեպքում □ գագաթի բնութագիրը փոխվում է և դառնում 10 (կամ 01)



Երկրորդ դեպքում նոր x գագաթը □ -ից կախվել է ազատ կողմից, որի արդյունքում □ գագաթի աջ և ձախ ճյուղերի երկարությունները հավասարվել են (բնութագիրը դառնում է հավասար 00), որով էլ նոր x գագաթի ավելացումն ավարտվում է:



Դիտարկենք «նշված» ճանապարհի կառուցվածքը, երբ x գագաթի □ հայր գագաթն ունի 00 բնութագիր:

Լեմմա: P «նշված» ճանապարհի Q ենթաճանապարհի, որի բոլոր գագաթներն ունեն 00 բնութագիր, իսկ նրա առաջին գագաթի (արմատին ամենամոտ) բնութագիրը 00-ից տարբեր է, երկարությունը հավասար է այն ենթածառի բարձրությանը (մակարդակների քանակին), որի արմատն է Q ենթաճանապարհի առաջին գագաթը:

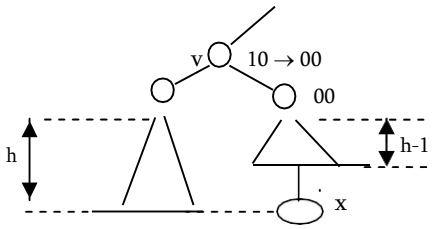
Ապացույց: Ենթադրենք մեր պնդումը ճիշտ չէ: Q ենթաճանապարհի առաջին գագաթը (որում բնութագիրը հավասար չէ 00) նշանակենք v-ով: v-ից դուրս եկող և Q ճանապարհից երկար բոլոր ճանապարհներից ընտրենք այն R ճանապարհը, որը Q-ի հետ ունի ամենաշատ ընդհանուր գագաթները: Դիցուք՝ t –ն Q և R ճանապարհների ամենավերջին ընդհանուր գագաթն է: Այդ դեպքում t գագաթի ճյուղերն ունեն տարբեր երկարություններ, հետևաբար t գագաթի բնութագիրը հնարավոր չէ որ հավասար լինի 00:

Լեմման ապացուցված է:

Այս լեմմայից բխում է, որ բնութագրերի փոփոխությունը բերում է Q ենթաճանապարհի բնութագրերի փոփոխմանը 00-ից 01 կամ 10, ընդ որում յուրաքանչյուր ենթածառ մնում է բալանսավորված:

Հնարավոր է երեք դեպք.

1. Q = P , այս դեպքում ծառը մնում է բալանսավորված:
2. Q ենթաճանապարհին ընկած է T(v) ենթածառի կարճ ճյուղի վրա: Այդ դեպքում նոր x գագաթի ավելացումից հետո ծառը նորից մնում է բալանսավորված և v գագաթի բնութագիրը դառնում է 00:



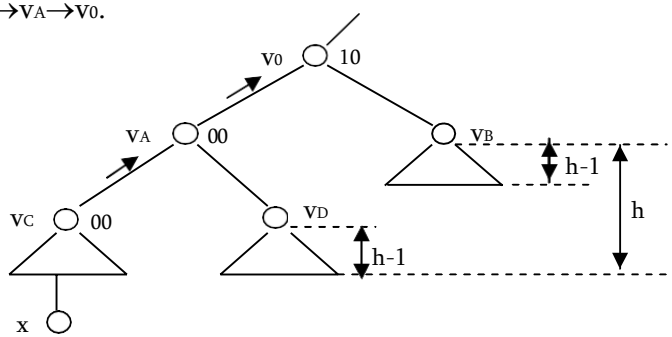
3. Q ենթաճանապարհին ընկած է T(v) ենթածառի երկար ճյուղի վրա:

Հնարավոր է երկու դեպք:

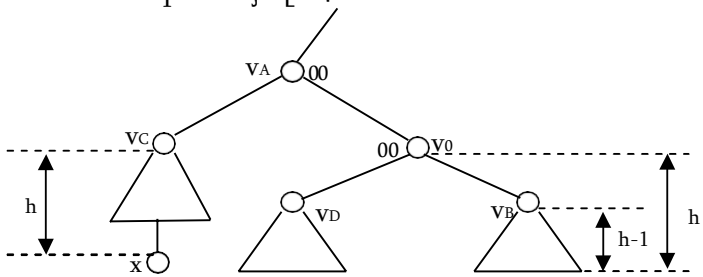
ա) Դեպի v գագաթը շարժվելիս վերջին երկու քայլերը կատարվում են նույն ուղղությամբ, այսինքն երկու անցումն էլ կա-

տարվում են ձախ որդուց հորը ($v_C \to v_A \to v_0$) կամ աջ որդուց հորը ($v_D \to v_B \to v_0$), այդ դեպքում ծառի վերակառուցման համար կատարվում է պարզ պտտում:

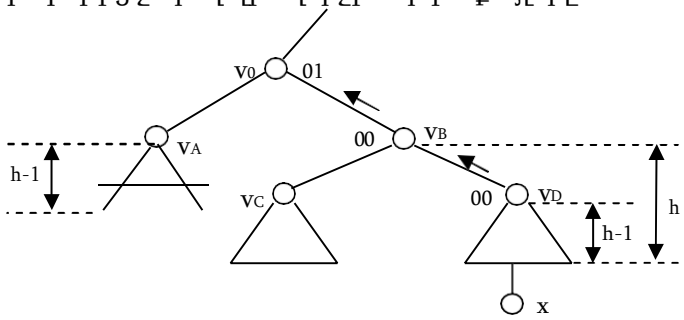
Ներկայացնենք v գագաթից շարժվելիս վերջին երկու քայլերը՝ $v_C \to v_A \to v_0$.



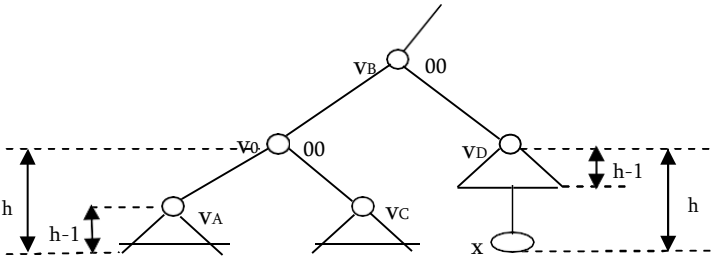
Ծառի վերակառուցման սխեման $v_C \to v_A \to v_0$ (ձախից ձախ) անցման ժամանակ հետևյալն է.



v գագաթից շարժվելիս վերջին երկու քայլերը՝ $v_D \to v_B \to v_0$

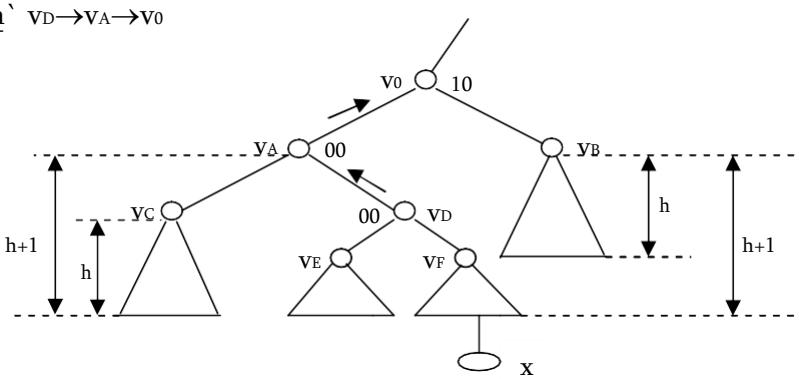


Ծառի վերակառուցման սխեման $v_D \to v_B \to v_0$ (աջից աջ) անցման ժամանակ հետևյալն է՝

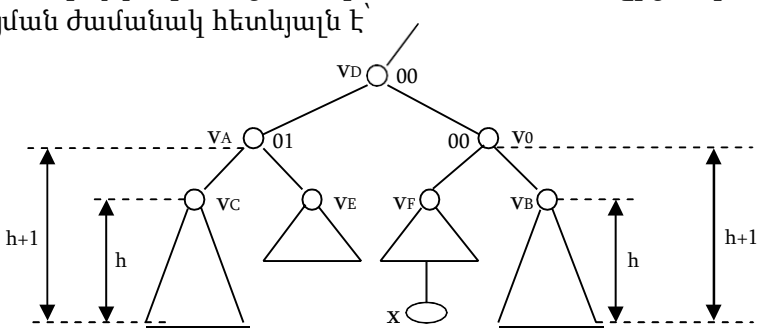


բ)Դեպի v գագաթը շարժվելիս վերջին երկու քայլերը կատարվում են տարբեր ուղղությամբ, այսինքն աջ որդուց հորը, որն իր հոր ձախ որդին է ($v_D \to v_A \to v_0$), կամ ձախ որդուց հորը, որն իր հոր աջ որդին է ($v_C \to v_B \to v_0$): Այս դեպքում ծառի վերակառուցման համար կատարվում է կրկնակի պտտում:

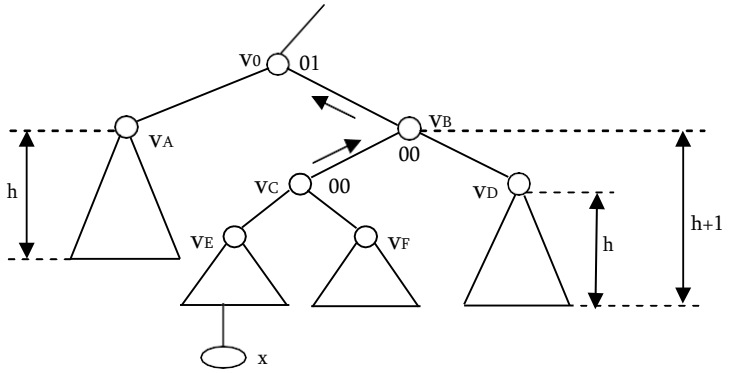
Ներկայացնենք v գագաթից շարժվելիս վերջին երկու քայլերը՝ $v_D \to v_A \to v_0$



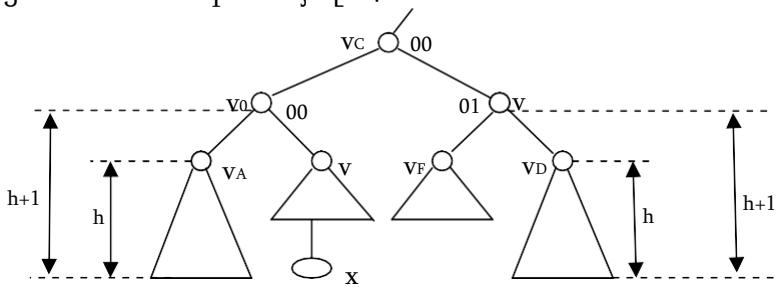
Ծառի վերակառուցման սխեման $v_D \to v_A \to v_0$ (աջից ձախ) անցման ժամանակ հետևյալն է՝



v գագաթից շարժվելիս վերջին երկու քայլերը՝ $v_C \to v_B \to v_0$.



Ծառի վերակառուցման սխեման $v_C \to v_B \to v_0$ (ձախից աջ) անցման ժամանակ հետևյալն է՝



Այս նկարներում բալանսի խախտում կատարվում է v_0 գագաթում, այսինքն Q ենթաճանապարհի առաջին գագաթում: Կարելի է ստուգել, որ վերակառուցումից առաջ և հետո ծառի բալանսավորվածությունը պահպանված է:

6.4. B-ծառեր

6.4.1. Ուժեղ ճյուղավորվող ծառեր

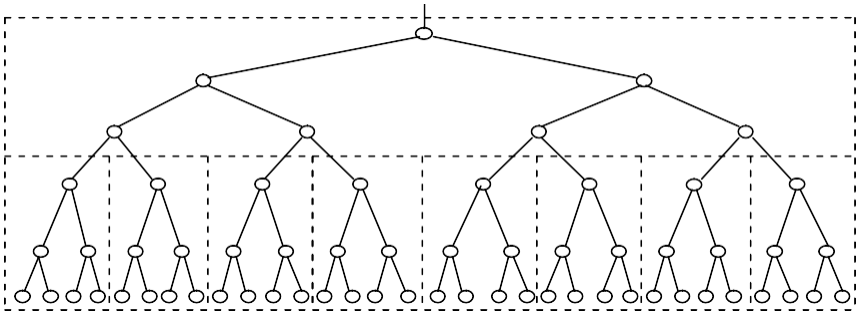
Տվյալների պահպանման B-ծառ կառույցն իր հատկությունների պատճառով ունի բավականին լայն կիրառություններ: Ծառերի հետ աշխատանքի ժամանակ ավելի հաճախակի դիտարկ-

վում են այն ծառերը, որոնց յուրաքանչյուր գագաթ ունի ամենաշատը երկու ժառանգ, այսինքն բինար ծառեր: Սա լիովին բավարար է, եթե օրինակ ցանկանում ենք ներկայացնել մարդկանց բարեկամական հարաբերությունները “ներքևից վերև գծի” առաջնայնությամբ, այսինքն երբ յուրաքանչյուր մարդու համար ներկայացվում են նրա ծնողները, ի վերջո յուրաքանչյուր մարդ երկուսից ավել ծնող չունի: Իսկ ինչպես վարվել, երբ անհրաժեշտ է ներկայացնել "վերևից ներքև գիծը" (օրինակ՝ տոհմածառը նկարելիս): Ակնհայտ է, հանգում ենք փաստին, որ որոշ մարդիկ ունեն 2-ից ավել զավակներ, հետևաբար այդ ծառը կպարունակի 2-ից ավել ճյուղերով գագաթներ: Այդպիսի ծառերը կոչվում են ***ուժեղ ճյուղավորվող ծառեր***:

Գոյություն ունի ուժեղ ճյուղավորվող ծառերի կիրառության ընդհանուր հետաքրքրություն ներկայացնող կարևոր տիրույթ: Դա մեծամասշտաբ ***փնտրման ծառերի ձևավորումն ու օգտագործումն է*** (օրինակ տվյալների բազաներում): Ենթադրենք դիտարկվում է որևէ քաղաքի բնակիչների հաշվառման բազան, որում յուրաքանչյուր բնակչի համար դիտարկված են նրա անձնական տվյալները, այդ թվում *"սոցիալական քարտի համարը"*: Բազաներում անհրաժեշտ են ինչպես **<u>ավելացման</u>** (օրինակ՝ յուրաքանչյուր նոր բնակիչ բազայում ավելացվում է ըստ իրեն սոցիալական քարտի համարի), **<u>փնտրման</u>** (օրինակ, որևէ բնակչի տվյալները փնտրվում են ըստ իրեն սոցիալական քարտի համարի), այնպես էլ **<u>հեռացման</u>** (օրինակ՝ տրված բնակիչը բազայից հեռացվում է ըստ իրեն սոցիալական քարտի համարի) ***գործողություններ***, որոնց իրականացման համար մեծաքանակ բնակչություն ունեցող քաղաքի դեպքում /օրինակ 11.000.000 բնակիչ/, ակնհայտ է կպահանջվի բավականին ժամանակ, բացի դրանից մեքենայի ներքին հիշողությունը բավականաչափ մեծ չէ, կամ բավականին թանկարժեք է, որպեսզի այն օգտագործվի տվյալների երկարժամանակյա պահպանման համար: Այդ գործողությունների իրականացման համար բազաներում ավելի հաճախ կիրառվում են ըստ որևէ բանալու ինդեքսավորված ֆայլերը, սակայն ուժեղ ճյուղա-

վորվող ծառերի կիրառությունը կբերի տվյալների պահպանման և փնտրման լավ արդյունքների:

Ենթադրենք ծառի հանգույցները պետք է պահվեն արտաքին հիշողությունում, ինչպիսին սկավառակն է: Եթե տվյալների համախումբը, որը բաղկացած է ենթադրենք միլիոն տարրերից, պահված է բինար ծառի տեսքով, ապա անհրաժեշտ տարրի ***փնտրման ժամանակ կպահանջվի փնտրման $\log_2 10^6 \approx 20$ քայլ:*** Քանի որ յուրաքանչյուր քայլը պահանջում է դիմում սկավառակին, ավելի ցանկալի կլինի հիշողության ավելի քիչ դիմում պահանջող կազմակերպում: Ուժեղ ճյուղավորվող ծառերը հանդիսանում են այդ խնդրի լուծման լավագույն միջոցը: Եթե դիմում է կատարվում ծառի միայն մեկ հանգույցին, որը գտնվում է արտաքին կրողի վրա, ապա նույն հաջողությամբ կարելի է դիմել նաև հանգույցների խմբին: Այստեղից հետևում է, որ ծառը պետք է տրոհել ենթածառերի, հաշվի առնելով, որ բոլոր այդ ենթածառերը միաժամանակ ամբողջությամբ հասանելի են: Այդպիսի ենթածառերին անվանենք ***էջեր***: Նկ.1-ում ներկայացված է էջերի տրոհված բինար ծառ, որի յուրաքանչյուր էջը պարունակում է 7 հանգույց:



նկ.1. Էջերի բաժանված բինար ծառ

Ենթադրենք, որոշել ենք յուրաքանչյուր էջում տեղադրել 100 հատ հանգույց (սա ընդունելի տարբերակ է), այդ դեպքում միլիոն տարր պարունակող փնտրման ծառը կպահանջի միջին հաշվով $\log_{100} 10^6 \approx 3$ դիմում էջերին 20-ի փոխարեն: Իհարկե, եթե ծառը ա-

ճում է “կամայականորեն”, ապա ամենավատագույն դեպքը կարող է պահանջել նույնիսկ 10^4 դիմում: Հասկանալի է, որ ուժեղ ճյուղավորվող ծառերի դեպքում անպայման անհրաժեշտ է նրանց աճը (բարձրությունը) ղեկավարող սխեմա:

6.4.2. n-րդ կարգի B-ծառեր

Աճը ղեկավարող հայտանիշի փնտրման ժամանակ պետք է միանգամից բացառել իդեալական հավասարակշռությունը, քանի որ այդ հավասարակշռության համար պահանջվում է մեծ ծախսեր(ժամանակ): Բավականին տրամաբանական հայտանիշ ձևակերպվել է Ռ. Բեյերի կողմից. **տրված n հաստատունի դեպքում յուրաքանչյուր էջ (բացի մեկից) պարունակում է n-ից 2n հանգույց:** Հետևաբար, N տարրով և էջի 2n առավելագույն չափով ծառում վատագույն դեպքը կպահանջի $\log_n N$ հղում էջերին (այդ գնահատականի ստացումը 3-րդ գլխում), իսկ հղումները, ինչպես հայտնի է, կազմում են փնտրման վրա ծախսված ժամանակի հիմնական մասը: Բացի այդ, հիշողության օգտագործման կարևոր գործակիցը կազմում է 50%-ից ոչ քիչ, քանի որ էջերը լցված են գոնե կիսով չափ: Այս բոլոր առավելություններով հանդերձ տրված կառույցը պահանջում է նաև տարրի <u>փնտրման, ավելացման և հեռացման</u> համեմատաբար պարզ ալգորիթմներ:

Դիտարկվող տվյալների կառույցները կոչվում են **n-րդ կարգի B-ծառեր** և ունեն հետևյալ հատկությունները. (n-ը կոչվում է B-ծառի կարգ).

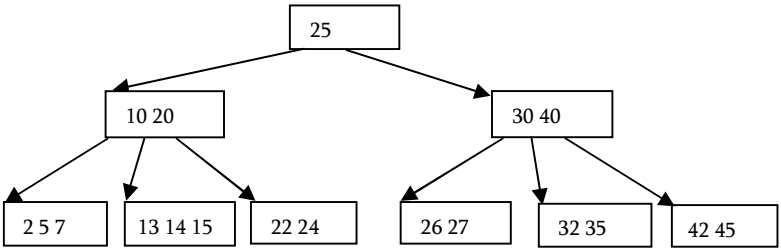
1. Յուրաքանչյուր էջ պարունակում է 2n-ից ոչ ավել տարր (բանալիներ),

2. Յուրաքանչյուր էջ բացի արմատից պարունակում է n-ից ոչ քիչ տարր,

3. Յուրաքանչյուր էջ համարվում է կամ տերև, այսինքն՝ ժառանգներ չունի, կամ ունի m+1 ժառանգներ, որտեղ m-ը նրանում գտնվող բանալիների թիվն է,

4. Բոլոր տերևները գտնվում են նույն մակարդակի վրա:

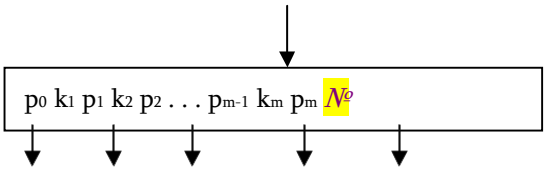
Նկ.2-ում ներկայացված է 3 մակարդակով 2-րդ կարգի (n=2) B-ծառ:



նկ.2. 2-րդ կարգի B-ծառ

Բոլոր էջերը պարունակում են 2, 3 կամ 4 տարր ($n \le m \le 2n$): Բացառություն է կազմում արմատը, որին թույլատրվում է պարունակել նաև մեկ տարր: Բոլոր տերևները գտնվում են 3-րդ մակարդակում: Բանալիները տեղադրված են ձախից աջ աճման կարգով: Եթե ծառը պրոյեկտենք մեկ մակարդակի վրա՝ տեղադրելով ժառանգներին իրենց հայր-էջի բանալիների միջև, կստանանք բոլոր բանալիները՝ դասավորված աճման կարգով: Այսպիսի դասավորությունը ներկայացնում է փնտրման բինար ծառերի բնական զարգացումը և սահմանում է տրված բանալով տարրի փնտրման մեթոդը:

Դիտարկենք նկ.3-ում ներկայացված տեսքն ունեցող էջը, որտեղ $\forall p_i$, $i=0,1,...,m$, հղում է անհրաժեշտ էջին, $\forall k_i$, $i=1,2,..,m$, բանալի է:



նկ.3. m բանալիներ պարունակող B–ծառի էջը

Ենթադրելով, որ էջը տեղադրված է օպերատիվ հիշողությունում, կարող ենք կիրառել $k_1,\ldots,k_m$ բանալիների միջև փնտրման հայտնի մեթոդներ (կախված հիշողության մեջ էջի ներկայացման կառույցից): Եթե m-ը բավականաչափ մեծ է և էջը ներկայացված է մասսիվի տեսքով, ապա կարելի է կիրառել բինար փնտրում (կիսման մեթոդ), իսկ եթե այն համեմատաբար մեծ չէ, ապա հար-

մար է պարզ հաջորդական փնտրումը: Այստեղ կարելի է նկատել, որ ***օպերատիվ հիշողությունում փնտրման վրա ծախսված ժամանակը բավականաչափ փոքր է այն ժամանակից, որն անհրաժեշտ է էջը արտաքին կրողի վրայից օպերատիվ հիշողություն տեղափոխելու համար:***

Դիցուք տրված է ***փնտրման x արգումենտը***: Եթե փնտրումը տվյալ էջում ***<u>հաջող է</u>*** ավարտվել, ուրեմն էջում գոյություն է ունեցել բանալի՝ $k_i=x$, $i=1,2,...,m$: Եթե փնտրումն ***<u>անհաջող է</u>***, ապա ուրեմն տեղի ունի հետևյալ իրավիճակներից որևէ մեկը.

1. կամ $\mathbf{k_i<x<k_{i+1}}$, որևէ $1\le i<m$–ի համար, այդ դեպքում փնտրումը շարունակվում է $P_i\uparrow$ (P_i-ում գրված հասցեով) էջում;
2. կամ $\mathbf{x>k_m}$, այդ դեպքում փնտրումը շարունակվում է $P_m\uparrow$ էջում;
3. կամ $\mathbf{x<k_1}$, այդ դեպքում փնտրումը շարունակվում է $P_0\uparrow$ էջում:

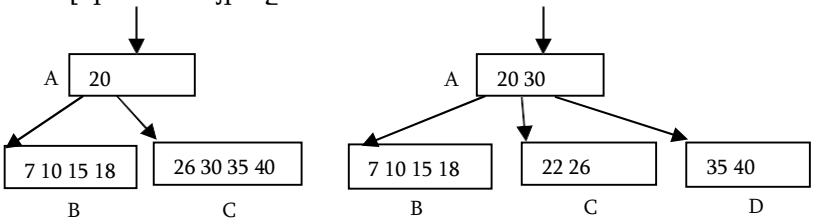
Եթե որևէ P հղումը հավասար է **nill,** այսինքն համապատասխան ժառանգ չկա, ապա ***<u>x բանալիով տարր չկա ամբողջ ծառում և փնտրումն ավարտվում է:</u>***

Զարմանալի է, սակայն B-ծառում ***նոր տարրի ավելացումը*** նույնպես կատարվում է համեմատաբար պարզ ձևով: Եթե տարրը տեղադրվում է էջում, որը պարունակում է $\mathbf{m<2n}$ տարր, ապա ավելացման գործողությունը սահմանափակվում է այդ էջով, իսկ էջում տարրի տեղադրման ալգորիթմի ընտրությունը պայմանավորված է դարձյալ հիշողության մեջ էջի ներկայացման կառույցից: Ավելացումն արդեն ***<u>լցված էջում</u>*** ազդում է ծառի կառուցվածքի վրա և այն կարող է առաջացնել նոր էջերի առաջացում: Որպեսզի պարզ դառնա, թե ինչ է տեղի ունենում այս դեպքում, դիտարկենք նկ.4-ը, որում ներկայացված է 22 արժեքն ունեցող բանալու ավելացումը 2-րդ կարգի B-ծառում: Այն բաղկացած է հետևյալ փուլերից.

1. Նախ ստուգվում և պարզվում է, որ 22 բանալին բացակայում է: C էջում նոր բանալու ավելացումը հնարավոր չէ, քանի որ C-ն արդեն լցված է:

2. C էջը բանալու 22 արժեքի հետ միասին տրոհվում է 2 էջերի, այսինքն տեղադրվում է նոր D էջը:

3. m+1 հատ բանալիները հավասարապես բաժանվում է C-ի և D-ի միջև, իսկ միջին բանալին (մեջտեղի) մեկ կարգ բարձրանում է վերև՝ A հայր-էջ:



նկ.4. 22 բանալու ավելացումը B-ծառին

Այս բավականին գեղեցիկ ալգորիթմը պահպանում է B-ծառերի բոլոր հիմնական հատկությունները: Մասնավորապես, տրոհման ժամանակ ստացվում են ճիշտ n տարր պարունակող էջեր: Ակնհայտ է, որ տարրի ավելացումը հայր-էջին իր հերթին կարող է առաջացնել էջի գերլցում, ինչը բերում է ***տրոհման տարածման***: Ծայրահեղ դեպքում այն կարող է **տարածվել մինչև արմատ**: Սա հենց B-ծառի բարձրության ավելացման միակ դեպքն է: Հետևաբար, B-ծառն աճում է տերևներից արմատ:

<u>Ուշադրություն:</u> *Նշենք, որ, եթե ծառի էջերը պահվելու են արտաքին կրողի վրա, ապա նկ.3-ում Pi հղումներն իրենցից ներկայացնելու են* ***ուղղակի էջերի համարներ****, որոնց մասին ինֆորմացիան կմտցվի էջի կառուցվածքի մեջ նոր «№» դաշտի ավելացմամբ, մասնավորապես, օրինակ արմատի համարը կլինի 0, իսկ էջի յուրաքանչյուր տրոհման ժամանակ էջի.*

– 1-ին կեսը կմնա նույն համարով, իսկ 2-րդ կեսը կստանա հերթական ազատ համարը, եթե տրոհվող էջը արմատը չէ,

– 2 կեսն էլ կստանան հերթական ազատ համարներ, եթե տրոհվող էջը արմատն է, նոր ստեղծվող արմատի համարը դնելով 0:

n-րդ կարգի B-ծառի գնահատականները

Դիտարկելով n-րդ կարգի B-ծառի կառուցումը, նկատում ենք, որ այն իրականացվում է հետևյալ 3 տիպի գործողությունների միջոցով.

– էջի փնտրում (հղում էջին),

– էջում տարրի փնտրում (կամ ավելացում),

– ելքի/մուտքի գործողություններ, եթե էջերը կազմակերպված են արտաքին կրողի վրա:

Հղումները կազմում են փնտրման վրա ծախսված ժամանակի հիմնական մասը: Բացի այդ, կախված B-ծառի էջերի կազմակերպումից (այսինքն էջերը պահված են ներքին, թե արտաքին հիշողությունում) ավելանում են նաև ելքի/մուտքի գործողություններ, որոնց վրա ծախսվող ժամանակը բավականաչափ մեծ է օպերատիվ հիշողությունում փնտրման վրա ծախսված ժամանակից: Ստանանք B-ծառի տարբեր իրականացումների դեպքում *տարրի փնտրման և ավելացման գործողությունների վրա ծախսված ժամանակի գնահատականներն* ըստ վերևում նշված գործողությունների:

Դիցուք ինֆորմացիոն մասսիվը բաղկացած է N տարրից և դիտարկվում է n-րդ կարգի B-ծառ, այսինքն էջում բանալիների քանակը՝ m-ը, բավարարում է $n \le m \le 2n$ պայմանին:

Ապացուցենք, ***որ N տարրով և էջի 2n առավելագույն չափով ծառում վատագույն դեպքը կպահանջի $\log_n N$ հղում էջերին:***

k-ով նշանակենք B-ծառի մակարդակների քանակը: Տարրի փնտրման ժամանակ k մակարդակ ունեցող B-ծառում էջերին հղումների քանակը չի գերազանցի k-ին:

Դիտարկենք նկ.3-ում ներկայացված տեսքն ունեցող էջը, որում տարրերի քանակը հավասար է m ($n \le m \le 2n$), հետևաբար հղումների քանակը հավասար է m+1 (ամենաշատը՝ 2n+1): Էջերի քանակը վատագույն դեպքում (երբ յուրաքանչյուր էջում կա ճիշտ n հատ տարր, բացի արմատից) չի գերազանցում **N/n**: Հաշվենք էջերի հնարավոր մինիմալ քանակը, այսինքն երբ յուրաքանչյուր էջը կպարունակի 2n+1 բանալի: Այդ դեպքում 1-ին մա-

կարդակում կունենանք 1 էջ, 2-րդ մակարդակում՝ (2n+1) էջ, k-րդ մակարդակում՝ $(2n+1)^{k-1}$ էջ, հետևաբար էջերի ընդհանուր թիվը կլինի այդ թվերի գումարը՝

$$\sum_{i=1}^{k}(2n+1)^{i-1}=\frac{(2n+1)^k-1}{2n}:$$

Ունենալով էջերի հնարավոր ամենաքիչ և ամենաշատ քանակները կստանանք.

$$\frac{(2n+1)^k-1}{2n}\le\frac{N}{n},\ n\neq 0$$

$$\begin{cases}(2n+1)^k\le 2N+1\\(2n+1)^k>(2n)^k+1\end{cases}\Rightarrow(2n)^k+1<2N+1\Rightarrow(2n)^k<2N$$

Հաշվի առնելով, որ $\log_n(2n)>1$, $0<\log_n 2\le 1$ ($n\ge 2$), $N>>n$, այստեղից կստանանք.

$$k<\log_{2n}2N=\frac{\log_n(2N)}{\log_n(2n)}<\log_n(2N)=\log_n 2+\log_n N\approx\log_n N$$

$$k<\log_n N$$

Ստացանք՝ $k<\log_n N$, որը նշանակում է, որ վատագույն դեպքում ***էջերին հղումների քանակը չի գերազանցում*** $\log_n N$:

Էջում ***<u>տարրի փնտրման, ապա տարրի ավելացման</u>*** համար կատարվող գործողությունների քանակի գնահատականը պայմանավորված է ներքին հիշողությունում էջի կազմակերպումից (կառույցից).

1. Եթե էջը կազմակերպված է որպես ***<u>միաչափ մասիվ</u>***, ապա

- տարրի <u>փնտրման</u> համար կարելի է կիրառել հայտնի որևէ լավ մեթոդ՝ օրինակ կիսման, որում համեմատությունների քանակը չի գերազանցում $\log_2 2n=\log_2 n+1$,

- տարրի <u>ավելացման</u> համար կկատարվի վատագույն դեպքում $\log_2 n+1$ **համեմատում՝** տարրի տեղը գտնելու համար, և **2n տեղափոխություն:**

2. Եթե էջը կազմակերպված է որպես ***<u>գծային կարգավորված ցուցակ</u>***, ապա

- տարրի *<u>փնտրումը</u>* կարելի է իրականացնել որպես հաջորդական փնտրում կարգավորված մասսիվում, որի ժամանակ վատագույն դեպքում կկատարվի ***2n համեմատում***,

- տարրի *<u>ավելացման համար</u>* կկատարվի վատագույն դեպքում **2n համեմատում՝** տեղը գտնելու համար, և ***ոչ մի տեղափոխություն,*** ուղղակի ավելացում:

Եթե ծառի էջերը պահվում են արտաքին կրողի վրա, կկատարվեն ***ելքի/մուտքի*** գործողություններ, որոնց քանակը յուրաքանչյուր տարրի փնտրման կամ ավելացման ժամանակ պայմանավորված է ծառի մակարդակների քանակով, որը ստացել ենք **k≈log₂N**, հետևաբար ***ե/մ գործողությունների քանակը***, մասնավորապես մեկ տարրի փնտրման (ավելացման) համար, չի գերազանցում **2*(k-1)**, իսկ N հատ տարրերի փնտրման(ավելացման) համար վատագույն դեպքում՝ **≈N*2*(k-1)≈N*2(log₂N-1)**, ինչը մեծ N-ի և բավականաչափ փոքր n-ի դեպքում կկազմի մեծ թիվ, հետևաբար բավականին շատ ժամանակ:

Այս դիտարկումների արդյունքում կարելի է կատարել հետևյալ եզրակացությունը. ***ինչքան մեծ է n-ը, այնքան քիչ են ե/մ գործողությունները, հետևաբար եթե N-ը այնքան մեծ թիվ է, որ նրա B-ծառն ամբողջությամբ չի տեղավորվում ներքին հիշողությունում, այսինքն օգտվելու ենք արտաքին հիշողությունից, ապա n-ը կարելի է ընտրել այնպես, որ ներքին հիշողության օգտագործվող տիրույթը բավարարի ծառի 2 էջ պահելու համար՝ արմատը և ընթացիկ էջը, հաշվի առնելով մեկ էջի կառուցվածքը (նկ.3):***

ԽՆԴԻՐՆԵՐ

1. Կարգավորման և փնտրման խնդիրներ հաջորդականությունների համար

1. Գրել ծրագիր, որը կարգավորում է հետևյալ պայմանին բավարարող տրված $\{a_n\}$ հաջորդականությունը.

$$a_1,a_2,\ldots,a_k,\ a_{k+1}<\ldots<a_n,\ k>1:$$

Կարգավորման ի՞նչ ալգորիթմով է հարմար այն իրականացնել, գնահատել ալգորիթմի ժամանակն ըստ համեմատությունների և տեղափոխությունների քանակի:

2. Գրել ծրագիր, որն ստանում է n տարր պարունակող հաջորդականության k հատ ամենափոքր տարրերը: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի: k-ի ինչպիսի՞ արժեքների համար է արդյունավետ նախապես կարգավորել հաջորդականությունը, ապա նոր ստանալ անհրաժեշտ տարրերը:

3. Գրել ծրագիր, որն ստանում է հաջորդականության ամենամեծ և ամենափոքր տարրերը: Կարո՞ղ է այդ ծրագիրը տրված փնտրումն իրականացնել 2n-3 համեմատություններից քիչ համեմատում կատարելով:

4. Դիցուք ունենք n տարրերից կազմված $S_1,S_2,\ldots,S_k$ հաջորդականությունները: Գրել ծրագիր, որն ստանում է հետևյալ գումարը.

$$\min\{s_{i1}+s_{i2}+\ldots+s_{ik}\},$$

որտեղ $\forall\ s_{ij}\in S_j$, $j=1,2,\ldots,k$, $k>1$: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունները կարգավորված են, և երբ կարգավորված չեն:

5. Գրել ծրագիր, որն ստանում է n տարրից կազմված հաջորդականության <u>ահտ</u>–ն (ահտ - ամենաշատ հանդիպող տարրը): Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի:

6. Գրել ծրագիր, որը ներմուծում է 2 հաջորդականություններ և ստուգում՝ նրանք համընկնում են, թե՞ ոչ (2 հաջորդականու-

թյուններ անվանենքն համընկնող հաջորդականություններ, եթե նրանցից յուրաքանչյուրի բոլոր տարրերը մյուսում կան): Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունները կարգավորված են, և երբ կարգավորված չեն:

7. Գրել ծրագիր, որը կարգավորում է տրված $\{a_n\}$ հաջորդականությունը, որի համար տեղի ունի հետևյալ պայմանը.

$$a_1 < a_2 < \ldots < a_k, a_{k+1}, \ldots, a_n,\ k>1:$$

Կարգավորման ո՞ր ալգորիթմով է հարմար այն իրականացնել, գնահատել ալգորիթմի ժամանակն ըստ համեմատությունների և տեղափոխությունների քանակի:

8. Գրել ծրագիր, որը ստանում է հաջորդականության ամենամեծ բացասական և ամենափոքր դրական տարրերը: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունները կարգավորված են, և երբ կարգավորված չեն:

9. Գրել ծրագիր, որն ստանում է n տարր պարունակող հաջորդականության k հատ ամենամեծ տարրերը: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի: k-ի ինչպիսի՞ արժեքների համար է արդյունավետ նախապես կարգավորել հաջորդականությունը, ապա նոր ստանալ անհրաժեշտ տարրերը:

10. Գրել ծրագիր, որը ներմուծում է 2 հաջորդականություններ, և ստուգում՝ նրանք տարբերվու՞մ են արդյոք միայն մեկ տարրով: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունները կարգավորված են, և երբ կարգավորված չեն:

11. Դիցուք ունենք n տարրերից կազմված $S_1, S_2, \ldots, S_k$ հաջորդականությունները: Գրել ծրագիր, որը ստանում է.

$$\min \{s_{i1} * s_{i2} * \ldots * s_{ik}\},$$

որտեղ $\forall\ s_{ij} \in S_j$, $j=1,2,\ldots,k$, $k>1$: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունները կարգավորված են, և երբ կարգավորված չեն:

12. Գրել ծրագիր, որը ստանում է n տարրից կազմված հաջորդականության աքտ–ն (աքտ - ամենաքիչ հանդիպող տարրը): Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունը կարգավորված է, և երբ կարգավորված չէ:

13. Գրել ծրագիր, որը ներմուծում է 3 հաջորդականություններ և ստուգում՝ նրանք համընկնում են, թե՞ ոչ: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունները կարգավորված են, և երբ կարգավորված չեն:

14. Գրել ծրագիր, որը ստանում է հաջորդականության ամենամեծ և ամենափոքր տարրերի մտնումների քանակը: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունները կարգավորված են, և երբ կարգավորված չեն:

15. Գրել ծրագիր, որն ստանում է n տարր պարունակող հաջորդականության k ($k<n/2$) հատ ամենափոքր և k հատ ամենամեծ տարրերը: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի: k-ի ինչպիսի՞ արժեքների համար է արդյունավետ նախապես կարգավորել հաջորդականությունը, ապա նոր ստանալ անհրաժեշտ տարրերը:

16. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը, և ստուգում՝ այդ հաջորդականությունում կրկնվող տարր կա, թե՞ ոչ: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունները կարգավորված են, և երբ կարգավորված չեն:

17. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը և ստուգում՝ այդ հաջորդականության զույգ թվերը կազմում են թվաբանական պրոգրեսիա, թե՞ ոչ: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի:

18. Գրել ծրագիր, որն ստանում է n տարր պարունակող հաջորդականության k հատ այն տարրերը, որոնց միավորի կարգում գրված թվանշանները ամենափոքրն են: Ինչպիսի՞ն է այդ

ծրագրի գնահատականն ըստ համեմատությունների քանակի: k-ի ինչպիսի՞ արժեքների համար է արդյունավետ նախապես կարգավորել հաջորդականությունը, ապա նոր ստանալ անհրաժեշտ տարրերը:

19. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը և պարզում՝ այդ հաջորդականության տարրերով ստացվում է թվաբանական պրոգրեսիա, թե՞ ոչ: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի:

20. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը և պարզում՝ այդ հաջորդականության տարրերով ստացվում է երկրաչափական պրոգրեսիա, թե՞ ոչ: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի:

2. Խնդիրներ դինամիկ հաջորդականությունների համար

1. Գրել ծրագիր, որը ներմուծում է n տարրից բաղկացած $S_1,S_2,...,S_k$ $(k>1)$ կարգավորված հաջորդականությունները և նրանցից ստանում մեկ կարգավորված հաջորդականություն: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի:

2. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը և ստանում նրա բնական թվի քառակուսի հանդիսացող թվերից կազմված b հաջորդականությունը: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունը կարգավորված է, և երբ կարգավորված չէ:

3. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը և ստանում նրա Ֆիբոնաչիի թվերից կազմված b հաջորդականությունը: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունները կարգավորված են, և երբ կարգավորված չեն: Ֆիբոնաչիի թվերը

ստացվում են հետևյալ անդրադարձ առնչությամբ. F(1)=1, F(2)=1, F(n)=F(n-1)+F(n-2), n>2:

4. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը և ստանում նրա պարզ թվերից կազմված b հաջորդականությունը: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունը կարգավորված է, և երբ կարգավորված չէ:

5. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը և ստանում նրա 2-ի աստիճան հանդիսացող թվերից կազմված b հաջորդականությունը: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունը կարգավորված է, և երբ կարգավորված չէ:

6. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը և ստանում նրա սիմետրիկ թվերից կազմված b հաջորդականությունը: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունը կարգավորված է, և երբ կարգավորված չէ:

7. Գրել ծրագիր, որը ներմուծում է n տարր պարունակող $S_1,S_2,...,S_k$ (k>1) նվազման կարգով կարգավորված հաջորդականությունները և նրանցից ստանում մեկ աճման կարգով կարգավորված հաջորդականություն: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի:

8. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը և ստանում նրա զույգ թվերի քառակուսի հանդիսացող թվերից կազմված b հաջորդականությունը: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունը կարգավորված է, և երբ կարգավորված չէ:

9. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը և ստանում նրա կատարյալ թվերից կազմված b հաջորդականությունը: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի, երբ այդ հաջորդականությունը կարգավորված է, և երբ կարգավորված չէ:

10. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը և ստանում b_n հաջորդականությունը, որի տարրերը a_n հաջորդականության թվերն են՝ վերադասավորված այդ թվերի թվանշանների գումարի աճման կարգով: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների և տեղափոխությունների քանակի:

11. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը և այն վերադասավորում նրա տարրերի ավագ կարգերի աճման կարգով: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի:

12. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը և ստանում b_{10} հաջորդականությունը, որի յուրաքանչյուր տարրը a_n հաջորդականության տարրերում 0-ից 9-ը թվանշանների մտնումների քանակն է: Ինչպիսի՞ն է այդ ծրագրի գնահատականն ըստ համեմատությունների քանակի:

3. Խնդիրներ ցուցակային և ծառային կառույցների համար

1. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը, կառուցում այդ հաջորդականության պարզ թվերից կազմված կարգավորված ցուցակ, արտածում, ապա ստացված ցուցակից հեռացնում է բանալու 100-ից փոքր պարզ թիվ արժեքով հանգույցները: Ստանալ այդ գործողության գնահատականն ըստ համեմատությունների քանակի: Ինչպիսի՞ն կլինի այդ գնահատականը չկարգավորված ցուցակի դեպքում:

2. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը, կառուցում այդ հաջորդականության սիմետրիկ (պոլինդրոմ) թվերից կազմված երկկողմանի կարգավորված ցուցակ, արտածում, ապա ստացված ցուցակից հեռացնում է բանալու 4-ին պատիկ թիվ արժեքով հանգույցները: Ստանալ այդ գործողության գնահատականն ըստ համեմատությունների քանակի: Ինչպիսի՞ն

կլինի այդ գնահատականը չկարգավորված երկկողմանի ցուցակի դեպքում:

3. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը, կառուցում այդ հաջորդականության քառանիշ երջանիկ թվերից կազմված կարգավորված ցիկլիկ ցուցակ, ապա ստացված ցուցակից հեռացնում է բանալու 3-ին պատիկ թիվ արժեքով հանգույցները: Ստանալ այդ գործողության գնահատականն ըստ համեմատությունների քանակի: Ինչպիսի՞ն կլինի այդ գնահատականը չկարգավորված ցիկլիկ ցուցակի դեպքում:

4. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը, կառուցում այդ հաջորդականության Ֆիբոնաչիի թվերից կազմված ստեկ և արտածում: Ստանալ ստեկի զույգ թվերի քանակը:

5. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը, կառուցում այդ հաջորդականության տարրերից կազմված հերթ և արտածում: Ստանալ հերթի կատարյալ թվերի քանակը:

6. Գրել ծրագիր, որը ներմուծում է a_n հաջորդականությունը, կառուցում այդ հաջորդականության բնական թվի քառակուսի հանդիսացող թվերից կազմված սովորական բինար ծառ և արտածում: Ստանալ ծառում 100-ից փոքր կենտ թվերի քանակը:

ԳՐԱԿԱՆՈՒԹՅՈՒՆ

1. Д. Кнут - *Искусство программирования для ЭВМ*, Сортировка и поиск , Мир 1976г.
2. С.С. Лавров, Л.И. Гончарова - *Автоматическая обработка данных. Хранение информиции в памяти ЭВМ*, Наука 1971г.
3. В.А. Евстигнеев - *Применение теории графов в программировании*, М. Наука 1985г.
4. Х.М. Дейтел, П.Дж. Дейтел – *Как программировать на C++*: М. 2000.
5. У. Топп, У.Форд – *Структуры данных в C++*: М. Бином, 2006г.
6. А.В. Ахо, Дж.Э. Хопкрофт, Дж.Д. Ульман - *Структуры данных и алгоритмы*. М.,С-П., К., 2003г.

ԲՈՎԱՆԴԱԿՈՒԹՅՈՒՆ