ШАГ компьютерная академия

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С

Dərs №10

C

Dilində proqramlaşdırma

Mündəricat

1. Xətti axtarış

Siz artıq massivinizi sıralamaq və ya onda hansısa bir veriləni tez tapmaq ehtiyacı duymuşsunuz. Bu dərsdə axtarış və çeşidləmə alqoritmləri şərh ediləcək. Bu gün biz sizə bu prosesləri avtomatlaşdırmaqda kömək edəcəyik.

Başlanğıc olaraq verilənlərin dahə sadə axtarılmasına – xətti axtarışa baxaq.

Bu alqoritm massivin hər bir elementini verilmiş açara görə müqayisə edir. Bizim nümunə kimi verilmiş massivimiz çeşidlənməmişdir və elə hal yaranar ki, axtarılan qiymət massivin ilk elementi ola bilər. Lakin ümumilikdə xətti axtarışı reallaşdıran proqram massivin elementlərinin yarısını açarla müqayisə edir.

```
#include <iostream>

using namespace std;

int LinearSearch (int array[], int size, int key){
    for(int i=0;i<size;i++)
        if(array[i] == key)
            return i;
    return -1;
}


void main()
{
    const int arraySize=100;
    int a[arraySize], searchKey, element;
    for(int x=0;x<arraySize;x++)
            a[x]=2*x;


     //Növbəti sətir ekrana ismarış çıxarır
     //Axtarış açarını daxil edin:
     cout<<"Please, enter the key: ";
```

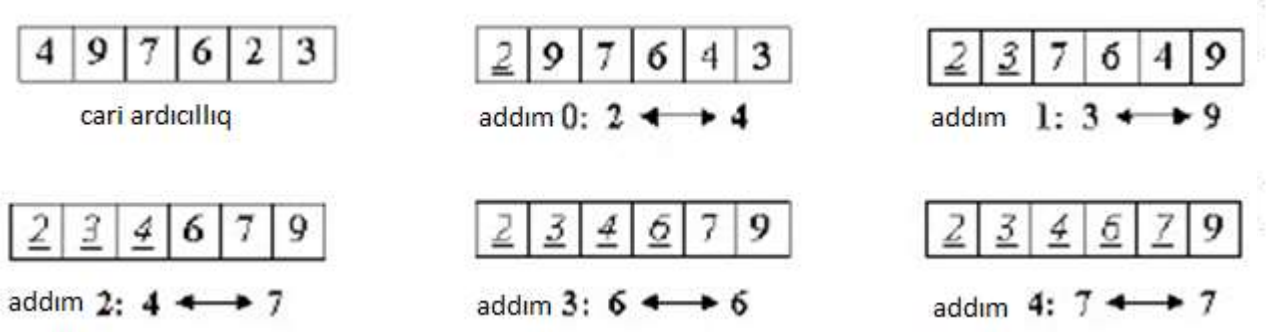
```
    cin>>searchKey;
    element=LinearSearch(a, arraySize, searchKey);

    if(element!=-1)
      //Növbəti sətir ekrana ismarış çıxarır
      //Massivdə axtarılan qiymət tapıldı
      cout<<"\nThe key was found in element "<<element<<'\n';

    //Növbəti sətir ekrana ismarış çıxarır
    //Qiymət tapılmadı
    else
      cout<<"\nValue not found ";

}
```

Qeyd edək ki, xətti axtarış alqoritmi yalnız böyük olmayan və ya çeşidlənməmiş massivlər üçün müvəffəqiyyətlə işləyir və tamamilə etibarlıdır.

2. Seçməklə çeşidləmə

Bu metodun idayası elementləri ardıcıl düzməklə sıralanmış ardıcıllıq əldə etməkdən ibarətdir.

İndi biz sizinlə masivin sol tərəfindən başlayaraq ardıcıllığı qurmağa çalışacağıq. Alqoritm 0-dan başlayıb (n-1)-də bitən n sayda ardıcıl addımlardan ibarətdir. i addımında a[i] ... a[n] elementləri arasında ən böyüyünü seçirik və a[i] elementi ilə onun yerini dəyişdiririk. n=5 olduğunda addımlar ardıcıllığı aşağıdakı sətirdə verilmişdir.



Növbəti addımın i nömrəsindən asılı olmayaq a[0]...a[i] ardıcıllığı çeşidlənmişdir. Beləliklə, (n-1) addımında a[n]-dən başqa bütün ardıcıllıq çeşidlənmiş olur, a[n] isə ən sağda qalmış olur, bütün kiçik elementlər isə sola keçmişdir. Bu metodu reallaşdıran misala baxaq:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
template <class T>
void selectSort(T a[], long size) {
      long i, j, k;
      T x;


       for(i=0;i<size;i++) {// i – cari addımın nömrəsi
           k=i;
           x=a[i];
           // ən kiçik elementin seçilməsi dövrü
           for(j=i+1;j<size;j++)
               if(a[j]<x){
                  k=j;
                  x=a[j];
                  // k - ən kiçik elementin indeksi
               }
           a[k]=a[i];
           a[i]=x;// ən kiçik a[i] ilə yerini dəyişdiririk
     }

}

void main(){
           srand(time(NULL));
           const long SIZE=10;
           int ar[SIZE];


              // çeşidlənmədən əvvəl
              for(int i=0;i<SIZE;i++){
                  ar[i]=rand()%100;
                  cout<<ar[i]<<"\t";
              }
              cout<<"\n\n";
              selectSort(ar,SIZE);


              // çeşidlənmədən sonra
              for(int i=0;i<SIZE;i++){
                  cout<<ar[i]<<"\t";
              }
              cout<<"\n\n";

}
```

Metodun əsas xüsusiyyətləri

1. n+1sayda baxılanlar arasında ən kiçik elementin qiymətini tapmaq üçün alqoritm n müqayisə aparır. Beləliklə, yerdəyişmələrin sayı müqayisələrin sayından həmişə az olacağı üçün çeşidləmə müddəti elementlərin sayından asılı olaraq artır.

2. Alqoritm əlavə yaddaş istifadə etmir: bütün əməliyyatlar “yerində” icra olunur.

Gəlin bu metodun nə qədər dayanıqlı olduğunu təyin edək? Hər biri iki sahə ehtiva edən (çeşidləmə onlardan birində yerinə yetirilir) üç elemetdən ibarət ardıcıllığa baxaq. Ardıcıllığın çeşidlənməsinin nəticəsini artıq 0-cı addımdan sonra görmək olar, belə ki, bundan başqa yerdəyişmə baş verməyəcək.

2a, 2b açarlar ardıcıllığı 2b, 2a olaraq dəyişmişdir.

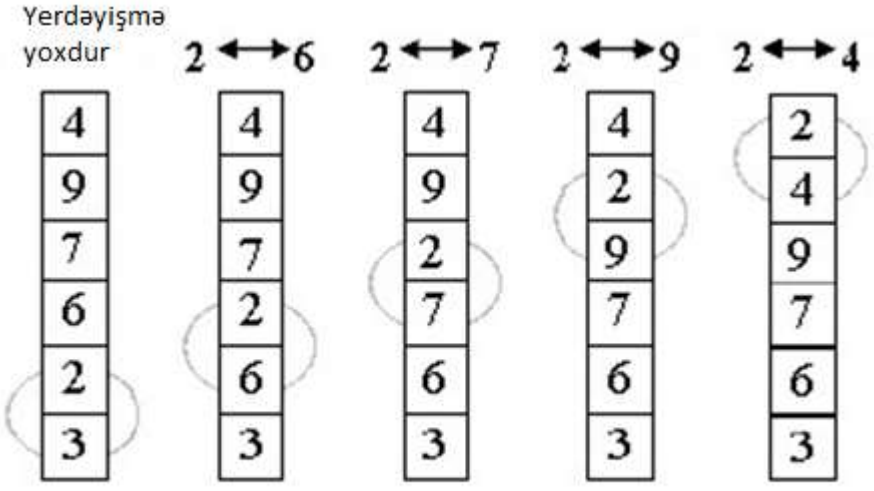
Beləliklə, giriş ardıcıllığı demək olar ki, sıralanmışdır, müqayisə də o qədər olacaq, deməli alqoritm optimal deyil. Lakin, bu çeşidləməni ölçüsü az olan massivlərə tətbiq etmək olar.

3. «Qabarcıqlı» çeşidləmə

Metodun ideyası növbəti şəkildədir: çeşidləmə addımı massiv boyunca aşağıdan yuxarı doğru icra edilir. Yol boyunca iki qonşu elementə baxılır. Əgər hər hansı bir qonşu elementlər cütlüyü düzgün ardıcıllıqda deyillərsə, onda onların yerlərini dəyişdiririk. Reallaşdırmaq üçün massivi yuxarıdan aşağıya, sıfırıncı elementdən sonuncu elementə doğru yerləşdirək.

Massivdə sıfırıncı keçiddən sonra «üstdə» ən «kiçik» element olacaq – qabarcıq anlayışı da burdan gəlir.

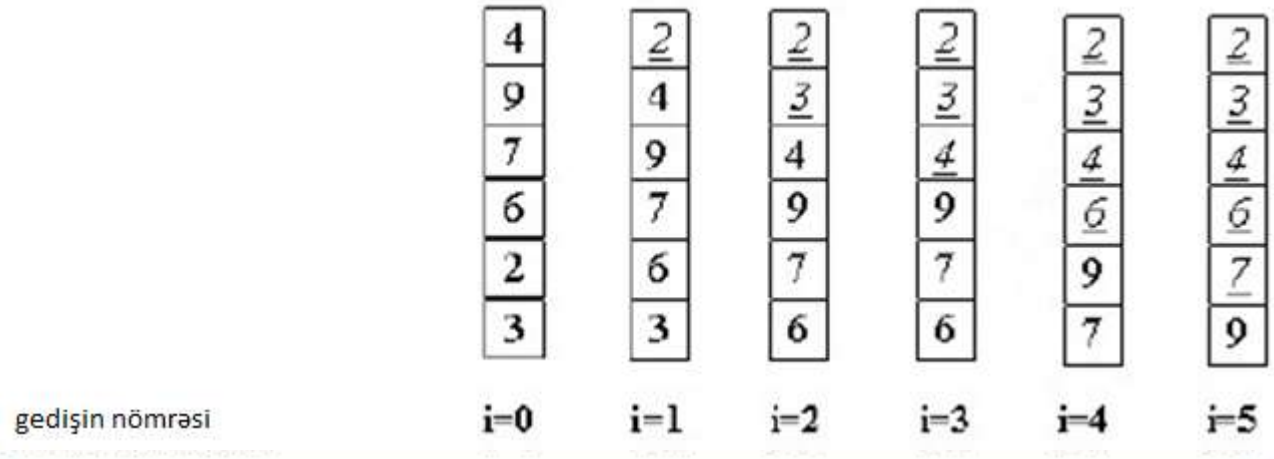
Növbəti gediş ikinci üstəki elementə qədər yerinə yetirilir, beləliklə qiymətcə ikinci element düzgün mövqeyə qalxır.



Müqayisə olunan sıfır gedişi qeyd edilmişdir

Massivin bütün azalan aşağı hissəsində bir element qalana qədər gediş edirik. Bununla da çeşidləmə tamamlanmış olur, belə ki, ardıcıllıq artan sıra ilə çeşidlənmişdir.

Kod nümunəsi:




```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
template <class T>
void bubbleSort(T a[], long size){
      long i, j;
      T x;
      for(i=0;i<size;i++){// i – gedişin nömrəsi
          for(j=size-1;j>i;j--){// gedişin daxili dövrü
              if(a[j-1]>a[j]){
                 x=a[j-1];
                 a[j-1]=a[j];
                 a[j]=x;
              }
          }
      }
}



void main(){
      srand(time(NULL));
      const long SIZE=10;
      int ar[SIZE];


       // çeşidləmədən əvvəl
       for(int i=0;i<SIZE;i++){
           ar[i]=rand()%100;
           cout<<ar[i]<<"\t";
       }
       cout<<"\n\n";
       bubbleSort(ar,SIZE);


       // çeşidləmədən sonra
       for(int i=0;i<SIZE;i++){
           cout<<ar[i]<<"\t";
       }
       cout<<"\n\n";

}
```

Metodun əsas xüsusiyyətləri.

Müqayisə və yerdəyişmələrin sayı kvadratik artır, burdan da o nəticəyə gəlmək olar ki, qabarcıqlı çeşidləmə alqoritmi astadır və azsəmərəlidir. Buna baxmayaraq onun kifayət qədər müsbət cəhətləri də vardır: o sadədir və onu müxtəlif formada inkişaf etdirmək olar. Biz indi bu hallara baxacağıq.

Hansısa bir gedişdə yerdəyişmənin olmadığı hala baxaq. Bu o deməkdir ki, bütün cütlüklər düzgün mövqedədirlər, yəni massiv artıq çeşidlənmişdir və prosesi davam etdirmək mənasızdır. Beləliklə, optimallaşdırmanın ilk addımı bu gedişdə hər hansı bir yerdəyişmənin olmasını yadda saxlamaqdan ibarətdir. Əgər yoxdursa, alqoritm işini tamamlayır.

Optimallaşdırma prosesini, yalnız yerdəyişmə faktını deyil, sonuncu yerdəyişmənin k indeksini yadda saxlamaqla da davam etdirmək olar. Həqiqətən də, bütün qonşu elementlərin k-dan kiçik olan indeksləri artıq lazım olan ardıcıllıqda yerləşmişdirlər. Növbəti gedişləri, əvvəlcədən təyin olunmuş üst sərhəddə görə deyil, k indeksində bitirmək olar.

Digər təkmilləşdirilmiş alqoritmi növbəti şəkildə əldə etmək olar. Belə ki, «yüngül» qabarcıq aşağıdan yuxarı bir gedişə qalxır, «ağır» qabarcıqlar isə aşağı sürətlə düşür: hər iterasiyada bir addım. Ona görə də 2 3 4 5 6 1 massivi 1 gedişə çeşidlənmiş olacaq, 6 1 2 3 4 5 ardıcıllığının çeşidlənməsi üçün isə 5 gediş lazımdır.

Bu cür səmərəlilikdən qaçmaq üçün ardıcıl gedişlərin istiqamətini dəyişmək olar. Alınan alqoritmi bəzən «titrək-çeşitləmə» adlandırırlar.

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
template <class T>
void shakerSort(T a[], long size) {
   long j, k=size-1;
 long lb=1, ub=size-1; // çeşidlənməmiş massivin sərhədləri
   T x;


    do{
         // aşağıdan yuxarıya gediş
         for(j=ub;j>0;j--){
             if(a[j-1]>a[j]){
                x=a[j-1];
                a[j-1]=a[j];
                a[j]=x;
                k=j;
             }
         }
         lb = k+1;
         // yuxarıdan aşağıya gediş
         for(j=1;j<=ub;j++){
             if(a[j-1]>a[j]){
                x=a[j-1];
                a[j-1]=a[j];
                a[j]=x;
                k=j;
             }
          }
         ub=k-1;
    }while (lb<ub);


}
void main(){
     srand(time(NULL));
     const long SIZE=10;
     int ar[SIZE];


      // çeşidlənməzdən öncə
      for(int i=0;i<SIZE;i++){
          ar[i]=rand()%100;
          cout<<ar[i]<<"\t";
      }
      cout<<"\n\n";
      shakerSort(ar,SIZE);
      // çeşidlənmədən sonra
      for(int i=0;i<SIZE;i++){
          cout<<ar[i]<<"\t";
      }
      cout<<"\n\n";

}
```

4. Daxiletməklə çeşidləmə

Sadə yerləşdirməklə çeşidləmə əvvəlki bölmələrdə şərh etdiyimizlərlə müəyyən qədər oxşardır. O məntiqi olara massivin hissəsi boyunca keçidlər edir və analoji olaraq onun əvvəlində çeşidlənmiş ardıcıllıq artır.

Lakin qabarcıqlı və seçməklə çeşidləmədə i-ci addımda a[0]...a[i] elementləri düzgün mövqedə olurlar və bir daha yerlərini dəyişməyəcəkləri dəqiqdir. Burda isə eyni mühakimə daha zəif olacaq: a[0]...a[i] ardıcıllığı çeşidlənmişdir. Bu zaman alqoritmin gedişatında ona bütun yeni elementlər daxil ediləcəkdir.

i-ci addımda alqoritmin necə icra olunacağına baxmaqla araşdıracağıq.

Yuxarıda deyildiyi kimi, ardıcıllıq bu hala qədər iki hissəyə ayrılmışdır: sıralanmış a[0]...a[i] və sıralanmamış a[i+1]...a[n]. Alqoritmin hər bir növbəti (i+1)-ci addımında a[i+1]-i götürüb massıvın artıq sıralanmış hissəsində lazım olan yerə yerləşdiririk.

Növbəti element üçün uyğun yerin tapılması ondan əvvəlki elementlərlə ardıcıl müqayisə ilə həyata keçirilir. Müqayisənin nəticəsindən asılı olaraq element ya cari yerində (daxil etmə tamamlanmışdır) qalır, ya da onar yerlərini dəyişdirirlər və proses təkrarlanır.

Beləliklə, yerləşdirmə prosesində biz x elementini massivin əvvəlinə doğru ondan kiçik element rast gələnə qədər və ya massivin əvvəlinə çatana qədər «ələkdən keçiririk».

Metodun reallaşdırılması

```
#include <iostream>
#include <stdlib.h>
#include <time.h>

using namespace std;

template <class T>
void insertSort(T a[], long size) {
        T x;
        long i, j;
        // kecidlər dövrü, i – keçidin nömrəsi
        for(i=0;i<size;i++){
            x=a[i];


              // hazır ardıcıllıqda elementin yerinin axtarılması
              for(j=i-1;j>=0&&a[j]>x;j--)
                // elementi yerinə çatana qədər sağa sürüşdürürük
                a[j+1]=a[j];
                // yer tapıldı, elementi daxil edin
                a[j+1] = x;

          }
}

void main(){
        srand(time(NULL));
        const long SIZE=10;
        int ar[SIZE];
```

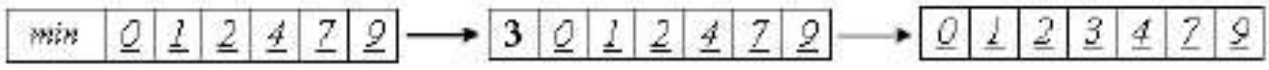
```
        // çüşidlənmədən əvvəl
        for(int i=0;i<SIZE;i++){
                ar[i]=rand()%100;
                cout<<ar[i]<<"\t";
        }
        cout<<"\n\n";
        shakerSort(ar,SIZE);


        // çeşidlənmədən sonra
        for(int i=0;i<SIZE;i++){
                cout<<ar[i]<<"\t";
        }
        cout<<"\n\n";
}
```

Metodun xüsusiyyəti

Bu üsulla çeşidləmənin əsas göstəricisi təbii halın olmasıdır: demək olar ki, artıq çeşidlənmiş massiv daha tez çeşidlənmiş olacaq. Alqoritmin bu dayanıqlığı uyğun hallarda metodun seçilməsinə şərait yaradır. Lakin, alqoritmi asanlıqla yaxşılaşdırmaq olar. Qeyd edək ki, daxili dövrün hər bir addımında 2 şərt yoxlanılır. Massivin əvvəlinə «siqnal element» yerləşdirməklə bu dövrləri birləşdirmək olar. Bu element şübhəsiz, massivin bütün elementlərindən kiçik olmalıdır.

Onda j=0 olduqda, a[0] <=x olacaq. Dövr 0-cı elementdə dayanacaq ki, bu da j>=0 şərtinə uyğundur. Beləliklə, Çeşidləmə düzgün aparılacaq, daxili dövrdə isə müqayisə bir vahid az olacaq. Lakin, çeşidlənmiş massiv tam olmayacaq, belə ki, onda ilk ədəd olmayacaq.

Çeşidləməni tamamlamaq üçün bu ədədi geri qaytarmaq, sonra isə çeşidlənmiş a[1]...a[n] ardıcıllığına əlavə etmək lazımdır.




```
#include <iostream>
#include <stdlib.h>
#include <time.h>

using namespace std;

template <class T>
void setMin(T a[],long size){
        T min=a[0];
        for(int i=1;i<size;i++)
                if(a[i]<min)
                        min=a[i];
        a[0]=min;
}


template <class T>
void insertSortGuarded(T a[], long size) {
        T x;
        long i, j;
        // könə birinci elementi saxlamaq
        T backup = a[0];
        // ən kiçiklə dəyişdirmək
        setMin(a,size);


          // çeşidlənmiş massiv
          for(i=1;i<size;i++){
              x = a[i];


               for(j=i-1;a[j]>x;j--)
                   a[j+1]=a[j];


               a[j+1] = x;
          }

          // backup-ı düzgün yerə yürləşdirmək
          for(j=1;j<size&&a[j]<backup;j++)
              a[j-1]=a[j];

          // elementi daxil etmək
          a[j-1] = backup;
}
```

```
void main(){
	srand(time(NULL));
	const long SIZE=10;
	int ar[SIZE];


	  // çeşidlənmədən əvvəl
	  for(int i=0;i<SIZE;i++){
		ar[i]=rand()%100;
		cout<<ar[i]<<"\t";
	  }
	  cout<<"\n\n";
	  insertSortGuarded(ar,SIZE);


	  // çeşidlənmədən sonra
	  for(int i=0;i<SIZE;i++){
		cout<<ar[i]<<"\t";
	  }
	  cout<<"\n\n";

}
```

5. Ev tapşırığı

1. 10 element ölçüsündə massiv verilir. Massivi üçüncü parametrdən asılı olaraq artan və ya azalan çeşidləyən funksiya yazın. Əgər bu parametr 1-ə bərabərdirsə, çeşidləmə azalan olmalı, 0-dırsa, artan olmalıdır. Funksiyanın ilk iki parametri bu massiv və onun ölçüsü, üçüncü parametri isə susmaya görə 1-ə bərabərdir.

2. –20-dən 20 diapazonunda təsadüfi ədədlər verilir. Kənar mənfi elementlərin (ən sol və ən sağ mənfi elementlərin) mövqeyini tapmalı və onlar arasındaki elementləri çeşidləməli.

3. 1-dən 20-ə qədər diapazonda qiymətlər alan 20 tam ədəd ehtiva edən massiv verilib.

Növbəti əməliyyatları yerinə yetirməli:

- massivin elementlərini ixtiyari şəkildə yerləşdirən funksiya yazmalı;
- bu diapazonda olan ixtiyari ədəd təyin etməli və massivdə onun mövqeyini tapmalı;
- massivin tapılmış mövqedən solda olan azalan sıra ilə, sağdakıları isə artan sıra ilə çeşidləməli.