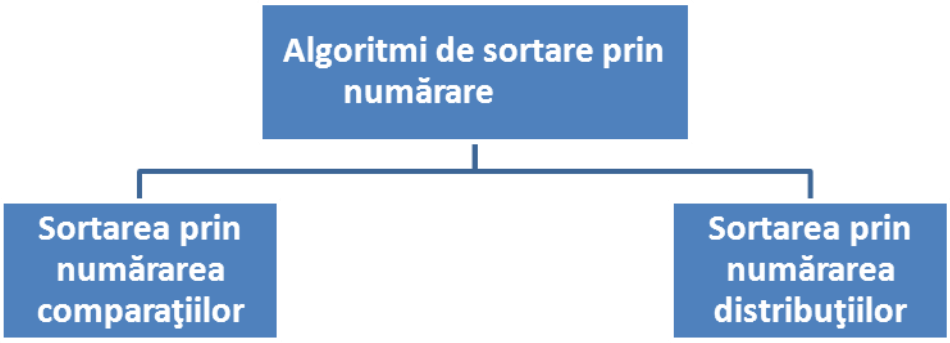
ALGORITMI DE SORTARE PRIN NUMĂRARE
- STUDIU -

Autor: Secită Neli

Ideea algoritmilor de sortare prin numărare este de a număra pentru fiecare element, v_i al unui vector, câte elemente sunt strict mai mici decât el. Numerele obţinute sunt memorate într-un vector COUNT. Pe baza acestuia se vor rearanja elementele vectorului v într-un alt vector X.



SORTAREA PRIN NUMĂRAREA COMPARAŢIILOR

PREZENTAREA METODEI

Sortarea prin numărare este o metodă simplă, care se bazează pe ideea că valoarea elementului aflat pe poziţia *j* din secvenţa finală sortată este mai mare decât exact *(j-1)* dintre celelalte valori ale elementelor. Cu alte cuvinte, dacă ştim că o anumită valoare a unui element depăşeşte exact 23 de alte valori şi dacă nu există două valori egale, elementul corespunzător trebuie să ajungă după sortare pe poziţia 24.

Metoda constă în compararea fiecărei perechi de valori, numărând câte vor fi mai mici decât fiecare valoare particulară.

Comparaţiile se fac în modul următor:

((se compară v_j, cu v_i) pentru $1 \le j \le n$) pentru $1 \le i \le n$.

Se observă că jumătate din aceste comparaţii sunt redundante, deoarece nu este necesar să se compare v_a cu v_a, pentru $1 \le a \le n$, şi nici v_a cu v_b şi apoi v_b cu v_a, pentru $1 \le a, b \le n$. În aceste condiţii avem nevoie de

((să se compare v_j *cu* v_i) pentru $1 \le j \le i$) pentru $1 \le i \le n$.

De remarcat că acest algoritm nu presupune deplasări de elemente. El este similar unei sortări cu tabel de adrese, pentru că tabelul ***COUNT*** indică aranjamentul final al elementelor. Dar este şi diferit, într-o oarecare măsură, deoarece ***COUNT[j]*** ne spune unde să-l mutăm pe v_j în loc să ne arate ce element trebuie mutată în locul v_j.

PREZENTAREA ALGORITMULUI

Algoritmul de comparare prin numărare va sorta elementele vectorului $v_1, v_2, ..., v_n$, după valorile sale prin intermediul unui tabel auxiliar ***COUNT[1], COUNT[2], ..., COUNT[n]*** pentru a număra câte elemente sunt mai mici decât un element dat. După terminarea algoritmului, ***COUNT[j]+1*** va specifica poziţia finală a elementului v_j. La terminarea algoritmului, elementele sunt deplasate într-o zonă de ieşire $X_1, X_2, ..., X_n$.

Algoritmul sortării prin numărarea comparaţiilor este următorul:

- Date de intrare: vectorul ***v[n]***
- Date de ieşire: vectorul ***X[n]*** care conţine elementele sortate.
- Se folosite vectorul ***COUNT[n]*** care conţine poziţiile în vectorul sortat ***X[n]***, adică ***COUNT[j]+l*** va specifica poziţia finală a înregistrării v_j.

Algoritmul în pseudocod este următorul:

```
Subalgoritm numarare_prin_comparatie (n,v,X)
  pentru  i=1,n execută
      COUNT[i] ← 0
  sfârşit pentru
  pentru  i=1,n execută
      pentru  j=1,i execută
            daca v[j]>v[i]atunci
                  COUNT[j] ← COUNT[j]+1
            altfel
                  COUNT[i] ← COUNT[i]+1
            sfârşit dacă
      sfârşit pentru
  sfârşit pentru
  pentru  i=1,n execută
      X[COUNT[i]] ← v[i]
  sfârşit pentru
sfârşit subalgoritm
```

Tabelul următor ilustrează comportarea tipică a numărării prin comparare, aplicată la 13 numere:

CHEIE	**512**	**210**	**315**	**511**	**456**	**859**	**231**	**411**	**502**	**990**	**41**	**310**	**45**
COUNT(iniţial)	0	0	0	0	0	0	0	0	0	0	0	0	0
COUNT(i=1)	1	0	0	0	0	0	0	0	0	0	0	0	0
COUNT(i=2)	2	1	0	0	0	0	0	0	0	0	0	0	0
COUNT(i=3)	3	1	2	0	0	0	0	0	0	0	0	0	0
COUNT(i=4)	4	1	2	3	0	0	0	0	0	0	0	0	0
COUNT(i=5)	5	1	2	4	3	0	0	0	0	0	0	0	0
COUNT(i=6)	5	1	2	4	3	6	0	0	0	0	0	0	0
COUNT(i=7)	6	1	3	5	4	7	2	0	0	0	0	0	0
COUNT(i=8)	7	1	3	6	5	8	2	4	0	0	0	0	0
COUNT(i=9)	8	1	3	7	5	9	2	4	6	0	0	0	0
COUNT(i=10)	8	1	3	7	5	9	2	4	6	10	0	0	0
COUNT(i=11)	9	2	4	8	6	10	3	5	7	11	1	0	0
COUNT(i=12)	10	2	5	9	7	11	3	6	8	12	1	4	0
COUNT(i=13)	**11**	**3**	**6**	**10**	**8**	**12**	**4**	**7**	**9**	**13**	**1**	**5**	**2**

Dacă există elemente care au valori egale, acestora le vor corespunde COUNT-uri egale, rearanjarea finală fiind greoaie, dar algoritmul dă rezultatul corect indiferent de numărul de valori egale.

Algoritmul în C++ este următorul:

```
void numarare_prin_comparatie(int n, int v[20], int (&X)[20])
{ int i,COUNT[20],j;
  for(i=1;i<=n;i++)
      COUNT[i]=0;
  for(i=1;i<=n;i++)
      for(j=1;j<=i;j++)
            if(v[j]>v[i])
                  COUNT[j]=COUNT[j]+1;
            else
                  COUNT[i]=COUNT[i]+1;
  for(i=1;i<=n;i++)
      X[COUNT[i]]=v[i];
}
```

COMPLEXITATEA ALGORITMULUI

Numărul de comparaţii din acest algoritm este egal cu $C_n^2 = \frac{n^2-n}{2}$, al cărui factor dominant $\boldsymbol{n^2}$ reprezintă timpul de rulare a acestui algoritm. Factorul n^2 care domină timpul de rulare arată că algoritmul nu este o cale eficientă de sortare în cazul în care *n* este mare. Dublarea numărului de elemente duce la creşterea timpului de rulare de patru ori. Deoarece metoda solicită compararea tuturor perechilor de chei distincte (v_i, v_j.), nu există o posibilitate de a scăpa de dependenţa de $\boldsymbol{n^2}$. Timpul mediu de rulare poate fi redus la **n log n** folosind un alt algoritm. Acest algoritm este interesant pentru simplitate şi nu pentru eficienţă.

SORTAREA PRIN NUMĂRAREA DISTRIBUŢIILOR

Din punct de vedere al eficienţei există o altă posibilitate de sortare prin numărare mai importantă. Acesta se aplică mai ales în cazurile în care există multe elemente care au valori egale şi când toate valorile elementelor sunt cuprinse în intervalul ***[u, t],*** cu ***t-u*** mic, adică $\boldsymbol{u \le v_j \le t}$, pentru $\boldsymbol{1 \le j \le n}$.

PREZENTAREA METODEI

Presupunem că toate valorile elementelor unui vector sunt cuprinse între 1 şi 110. Într-o primă trecere prin vector numărăm câţi de 1, de 2, ..., de 110 sunt prezente. La a doua trecere putem deplasa elementele vectorului în locurile corespunzătoare în zona de ieşire, etc.

PREZENTAREA ALGORITMULUI

Considerăm că toate valorile elementelor unui vector sunt întregi în domeniul $\boldsymbol{u \le v_j \le t}$ pentru $\boldsymbol{1 \le j \le n}$. Acest algoritm va sorta elementele vectorului $\boldsymbol{v_1, v_2 ..., v_n}$ utilizând un tabel auxiliar ***COUNT[u] COUNT[u+1], ..., COUNT[t]***. La terminarea algoritmului, elementele vectorului sunt deplasate într-o zonă de ieşire $\boldsymbol{X_1, X_2, ..., X_n}$.

Algoritmul în pseudocod este următorul:

```
Subalgoritm numarare_prin_distributie (n,v,X)
  u = v[1]
  t = v[1]
  pentru i =2,n executa
      daca u>=v[i] atunci
            u = v[i]
      altfel
            daca t<v[i] atunci
                  t = v[i]
  sfârşit pentru
  pentru i =u,t execută
      COUNT[i]:=0
  sfârşit pentru
  pentru j =1,n executa
      COUNT[v[j]] = COUNT[v[j]] +1
  sfârşit pentru
  pentru i = u+1, t executa
      COUNT[i] = COUNT[i] + COUNT [i-1]
  sfârşit pentru
  pentru j = n,1 executa
      i = COUNT[v[j]]
      X[i] =v[j]
      COUNT[v[j]] = i-1
  sfârşit pentru
sfârşit subalgoritm
```

Algoritmul în C++ este următorul:

```
void distributie(int n, int v[20], int (&X)[20])
{ int COUNT[20],i,j,u,t;
    u=v[1];
    t=v[1];
 for(i=2;i<=n;i++)
     if(u>=v[i])
          u=v[i];
    else
       if (t<v[i])
            t=v[i];
 for(i=u;i<=t;i++)
      COUNT[i]=0;
 for(j=1;j<=n;j++)
      COUNT[v[j]]=COUNT[v[j]]+1;
 for(i=u+1;i<=t;i++)
      COUNT[i]=COUNT[i]+COUNT[i-1];
 for(j=n;j>=1;j--)
      {i=COUNT[v[j]];
      X[i]=v[j];
      COUNT[v[j]]=i-1;}
}
```