Analiza comparativă a complexității

Universitatea de Vest

Bulevardul Vasile Pârvan 4, Timișoara, România

Facultatea de matematică și informatică

Lenuț Răzvan-Ionuț

Aprilie 2019

Cuprins

1 Rezumat

Sortarea reprezintă un element important în cadrul proceselor de prelucare a datelor și presupune o aranjare a datelor în baza unor criterii de ordonare prestabiliti. Prezentul material detaliază importanța criteriilor de sortare, atât din punct de vedere teorietic, cât și aplicativ, fiind concluzionate cele mai relevante rezultatele, obinute în urma simulării unor cazuri similar cu cele reale.

2 Introducere

Procesul de sortare este prezentat în mod diferențiat în funcție de domenul în care acesta este aplicat. Deși în mod curent sortarea este prezentată ca fiind un proces de organizare prin separatori care prezintă anumite caracteristici de grup, pentru programatori acest proces prespune reorganizarea unui ansamblu de date și informații înmagazinate în memoria sistemului ascendent sau descendent. Lucrarea îşi propune să demonstreze efciența algoritmior de sortare, precum și prezentarea eventualelor diferențe sau asemănări rezultate în urma aplicării practice a fiecărui algoritm. Procesul de sortare este analizat atât din perspectivă teoretică, cât și practică, urmărindu-se demostrarea aplicabilității și veridicității noțiunilor teoretice, precum și reliefarea complexității diferitelor metode de sortare, sens în care a fost utilizat limbajul de programare C++.

3 Procesul de sortare prin comparație

Algoritmii folosiți pentru sortarea prin comparație se împart, în funcție de scop în: - Interschimbare, care presupune reorganizarea valorilor a[i] și a[j] într-o anumita ordine; În acest sens, prezintă interes deosebit metoda bulelor – "Bubble Sort".

3.1 Metoda Bubble Sort

Această modalitate de sortare nu este cea mai performantă, fiind eficinet în situația în care există elemente „aproape ordonate", la baza algoritmului fiind compararea între două elemente alăturate, la fiecare etapă. Ceea ce definește acest model este faptul că elemenetele se schimbă între ele pe baza variabilei aux, iar variabila booleană OK vizează modul de control pentru acțiunea repetitivă. În vederea analizării intevalul de timp necesar implementării acestei metode, se impune luarea în considerare a: numărului de comparații, cel de interschimbări, precum și numărul de treceri. Astfel, se poate ajunge la cea mai favorabilă situație, cea în care elementele să fie aranjate crescător, și care ar presupune parcurgerea informațiilor doar o singură dată, respectiv cea mai nefavorabilă situație, în care datele apar în ordine descrescătoare, ceea ce va impune parcurgerea a n-1, n-2, etc. etape. Inserție, prin introducerea unei valori în cadrul unui subsecvențe aflță deja într-o anumită ordine;

```
 Subalgoritm Metoda_bulelor(v,n)
1: int sortat;
2:  do{
3:          sortat = 1;
4:          for (i = 1; i <= N-1; i++){
5:                  if (v[i] > v[i+1]){
6:                          int aux = v[i];
7:                           v[i] = v[i+1];
8:                           v[i+1] = aux;
9:                          sortat = 0;
10:                 }
11:          }
12:  }while(sortat == 0);
13:
14:  for (i = 1; i <= N; i++)
15:          cout<<v[i]<<" ";
```

3.1.1 Partea experimentala a complexității

În urma aplicării practice a noțiunilor teoretice, cel mai bun timpul de execuție se obține în situația în care datele sunt sortate în ordinea dorită. Este de pecizat faptul că, cu cât crește număr de date, se majorează timpul de execuție, fiind foarte greu de determinat. În această situație, aplicarea terorectică confirmă ipotezele teoretice.

Timpul de execuție pentru un tablou ordonat descrescator:

Numărul de elemente	Timpul de execuție
n=1000	0,186 secunde
n=3000	1,689 secunde
n=5000	4,607 secunde
n=7500	10,605 secunde
n=10000	19,375 secunde
n=100000	NA

3.2 Sortare prin inserție

Sortarea prin inserţie este folosită cu precădere în cazul sortării unui număr relativ redus de elemete, fiind dependentă de ordonarea deja realizată în cazul elementelor supuse procesului de sortare. Pentru a facilita înțelegerea acestei metode, este utilă analogia cu un pachet de cărți așezat pe un suport, a cărui aranjare se face într-o mână, prin ridicarea câte unei cărți, pe rând, și verificarea acesteia cu restul cărților avute în mână, în vederea stabilirii poziției corecte a fiecărei cărți. În cazul acestei metode, intervelul de timp necesar aplicării unui algoritm depinde de volumul datelor introduse, sens în care timpul de execuție depinde de numărul operațiilor primate spre a fi executate. În acest caz, situația

cea mai favorabilă este cea în care vectorul de intrare era deja sortat, respectiv cel mai defavorabil, în care vectorul de intrare era sortat în ordine inversa. Chiar dacă sunt realizate îmbunătățiri pentru metoda buleleor, acestea este mai lentă decât sortarea prin inserție, cea dintâi fiind însă mai accesibilă, datorită modul simplu de implementare.

```
    Sort-Inser¸tie(array x):
1:  for i = 2 · · · n
2:     aux  x[i] // fixarea elementului i
3:     j  i  1
4:  while (j >= 1) and (aux < x[j]) // cautarea pozi¸tiei
5:     x[j + 1]  x[j] // deplasare element x[j] pe poziția j + 1
6:     j  j  1
7:  endwhile
8:  x[j + 1]  aux // deplasare element x[i] pe pozi¸tia cautat  a
9:  endfor
10: return x
```

3.2.1 Partea experimentala a complexității

Timpul de execuție pentru un tablou ordonat descrescator:

Numărul de elemente	Timpul de execuție
n=1000	0.076 secunde
n=3000	0.734 secunde
n=5000	2.203 secunde
n=7500	4.858 secunde
n=10000	8.268 secunde
n=100000	NA

Timpul de execuție pentru un tablou ordonat crescator:

Numărul de elemente	Timpul de execuție
n=1000	0.076 secunde
n=3000	0.734 secunde
n=5000	2.203 secunde
n=7500	4.858 secunde
n=10000	8.268 secunde
n=100000	NA

În urma aplicării practice, se observă că, elementele de teorie se confirmă. Astfel, se observă că, o crește a dimeninsiunii problemei determină o majorare a timpului de execuție, respectiv de afișare a rezultatului. Precizăm că, în comparație cu metoda bulelor, aceasta metodă se comportă mai bine în situația cazului defavorabil.

4 Procesul de sortare prin metoda "Divide et impera"

Divide et impera reprezintă fundamentul de construire a unor algoritmi eficienți pentru sortări, înmulțiri de numere mari, analize sintetice sau programarea în paralel, cu ajutorul mai multor procesoare. Acest algoritm presupune parcurgea a trei etape: divide, stăpânește și combină.

4.1 Sortarea rapidă (quick-sort)

În vederea implementării se impune existența unei proceduri care trasează o parte de vector situate între indicii p și q. Rolul acesteia vizează plasarea primei componente a[p] pe o poziție k, aflată între p și q, cu condiția ca elementele vectorului aflate între p și k-1 să fie mai mici sau egale decat a[k], respective elementele vectorului situate între k+1 și q să fie mai mari sau egale cu a[k]

Acest algoritm este considerat a fi cel mai eficient și presupune parcurgerea a trei etape: Divide: Şirul a[p..q] este împărţit în doua subşiruri nevide a[p..k] şi a[k + 1..q], astfel încât fiecare element al subşirului A[p..k] să fie mai mic sau egal cu orice element al subşirului a[k + 1..q]. Stăpâneşte: Cele două subşiruri a[p..k] şi a[k + 1..q] sunt sortate prin apeluri recursive ale algoritmului de sortare rapidă. Combină: Datorită faptului că cele două subșiruri sunt sortate pe loc, nu se impune nici o combinare. Într-o abordare simplistă, acest algoritm presupune stabilirea unui element din întreg,care trebuie sortat. Întregul este împărțit în două divizii, aplasate de o parte și de alta a elementului astfel: toate componentele mai mari decât elementul sunt plsate în dreapta și cele mai mici în stânga. Timpul de execuţie al acestui algoritm se stabilește în funcție de felul de partiţionare – dacă acesta este echilibrat sau nu -, sens în care, în cazul unei partiționări echilibrate algoritmul poate genera rezultate în același interval de timp ca sortarea prin interclasare. Într-o situație de partiționare dezechilibrată algoritmul necesită un timp de executiție la fel de încet ca sortarea prin inserare. Rezultatul cel mai defavorabil al algoritmului de sortare rapidă se înregistrează în situația în care procedura de partiţionare produce un vector de n-1 elemente şi unul de 1 element, iar daca algoritmul produce doi vectori de n/2 elemente atunci algoritmul de sortare rapidă lucrează mai eficient. În vederea clarificării unei comportări medii a acestui algoritm se formulează propuneri privind frecvența unor intrări, dintre care cea mai evidentă este că toate permutările elementelor de intrare sunt la fel de probabile. În contexul în care se lucrează pe o intrare aleatoare, există probabilitatea ca partiţionarea să nu mai fie șa același nivel, fiind aşteptat ca unele partiţionări să fie echilibrate, iar altele nu.

```
1:  void pozitie(int p,int u,int &k)
2:  {int i,j,di,dj,aux;
3:  i=p;
4:  j=u;di=0;dj=-1;
5:  while(i<j)
```

```
6:  {if(v[i]>v[j])
7:   {aux=v[i];
8:     v[i]=v[j];
9:     v[j]=aux;
10:     aux=di;
11:     di=-dj;
12:     dj=-aux;
13:     }
14: i=i+di;
15: j=j+dj;
16: }
17: k=i;
18: }
19: void quick(int p,int u)
20: {int k;
21: if(p<u)
22: {pozitie(p,u,k);
23: quick(p,k-1);
24: quick(k+1,u);
25: }
26: }
```

4.1.1 Partea experimentala a complexității

În urma testării celor trei cazuri prezentate a nivel teoretic, s-a constatat o creștere a timpului de execuție pentru cazul defavorabil, comparativ cu cel favorabil și cel mediu, în cazul celor două fiind obținute valori apropiate.

Timpul de execuție pentru un tablou ordonat descrescator:

Numărul de elemente	Timpul de execuție
n=1000	0.015 secunde
n=3000	0.020 secunde
n=5000	0.036 secunde
n=7500	0.049 secunde
n=10000	2.240 secunde
n=100000	4.56 secunde
n=700000	7.511 secunde
n=900000	8.624 secunde
n=1000000	15.309 secunde

Timpul de execuție pentru un tablou ordonat crescator:

Numărul de elemente	Timpul de execuție
n=1000	0.002 secunde
n=3000	0.003 secunde
n=5000	0.004 secunde
n=7500	0.006 secunde
n=10000	0.505 secunde
n=100000	1.231 secunde
n=700000	2.256 secunde
n=900000	3.078 secunde
n=1000000	16.009 secunde

Timpul de execuție pentru un tablou aproape sortat:

Numărul de elemente	Timpul de execuție
n=1000	0.003 secunde
n=3000	0.007 secunde
n=5000	0.014 secunde
n=7500	0.022 secunde
n=10000	0.033 secunde
n=100000	0.475 secunde
n=700000	3.606 secunde
n=900000	5.578 secunde
n=1000000	7.301 secunde

4.2 Sortarea prin interclasare

Acest tip de algoritm utilizează o abordare de tipul divide şi stăpâneşte, și presupune împărțirea problemei inițiale în mai multe probleme de mici dimensiuni, similare celei inițiale, a căror rezolvare este combinată pentru a determina soluția problemei inițiale. Aferent acestei metode, există următorul mod de abordare: Divide: Împarte şirul de n elemente care urmează a fi sortat în două subşiruri de câte n/2 elemente. Stăpâneşte: Sortează recursiv cele două subşiruri utilizând sortarea prin interclasare. Combină: Interclasează cele două subşiruri sortate pentru a produce rezultatul final. Acest tip de sortare are ca principal avantaj parcurgerea secvențială a elementelor, metoda fiind recomandat a fi utilizată în cazul sortărilor de fișiere și liste. Principiul de funcționare presupune împărțirea, în mai mulți pași, a unei liste în două părți egale și soratea ulterioară a fiecări părți prin interclasare. În vederea determinării elementului plasat la mijloc este necesară parcurgerea întregii liste, ulterior având loc izolarea fiecări părți în câte o listă.

```
1: int i = 0, j = 0;
2:     while(i < n && j < m)
3:     {if(A[i] < B[j])
```

```
4:          { C[k] = A[i];
5:              k++;
6:              i++;}
7:         else
8:          {C[k] = B[j];
9:              k++;
10:             j++;}
11:    }
12:
13:    if(i <= n)
14:    {for(int p = i; p < n; p++)
15:        {C[k] = A[p];
16:            k++;}
17:    }
18:    if(j <= m)
19:    {for(int p = j; p < m; p++)
20:        {C[k] = B[p];
21:            k++;}
22:    }
23:
24:    for(int p = 0; p < k; p++)
25:        cout << C[p] << " ";
```

4.2.1 Partea experimentala a complexității

Folosind acest criteriu de sortare, eficiența în execție este vizibilă doar în cazul existenței unor numere mari.

Timpul de execuție pentru un tablou ordonat crescator:

Numărul de elemente	Timpul de execuție
n=1000	0.003 secunde
n=3000	0.019 secunde
n=5000	0.035 secunde
n=7500	0.059 secunde
n=10000	0.071 secunde
n=100000	1.321 secunde
n=700000	4.414 secunde
n=900000	8.706 secunde
n=1000000	10.015 secunde

Timpul de execuție pentru un tablou ordonat descrescator:

Numărul de elemente	Timpul de execuție
n=1000	0.007 secunde
n=3000	0.020 secunde
n=5000	0.032 secunde
n=7500	0.0602 secunde
n=10000	0.073 secunde
n=100000	0.864 secunde
n=700000	4.456 secunde
n=900000	8.709 secunde
n=1000000	9.934 secunde

Bibliografie

Cormen, Thomas Leiserson, Charkes Rivest, Ronald (1990) - Introducere in algoritmi București: Editura Aurora

Livovschi, L. Georgescu - Sinteza si analiza algoritmilor Editura Științifiicaa si Enciclopedică, București 1986 Knuth, D.E. - Tratat de programarea calculatoarelor. Sortare si căutare, București 1976

Brassard G., Bratley P., Algorithmics - Theory and Practice, Prentice Hall, Englewood Cliffs, 1988

Lică, Dana Pașoi - Fundamentele programării (2005), București Editura LS info-mat