Proiectarea algoritmilor
Lucrare de laborator nr. 5

Sortare - bubleSort, naivSort, InsertionSort

Cuprins

1 Sortarea prin interschimbarea elementelor vecine (*bubble-sort*)

1.1 Descriere

Notăm cu $SORT(\texttt{a})$ predicatul care ia valoarea *true* dacă şi numai dacă tabloul `a` este sortat. Metoda *bubble-sort* se bazează pe următoarea definiţie a predicatului $SORT(\texttt{a})$:

$$SORT(\texttt{a}) \iff (\forall i)(0 \leq i < n-1) \Rightarrow a[i] \leq a[i+1]$$

O pereche (i, j), cu $i < j$, formează o *inversiune (inversare)*, dacă $a[i] > a[j]$. Pe baza definiţiei de mai sus vom spune că tabloul `a` este sortat dacă şi numai dacă nu există nici o inversiune $(i, i+1)$.

Metoda *bubble-sort* propune parcurgerea iterativă a tabloului `a` şi, la fiecare parcurgere, ori de câte ori se întâlneşte o inversiune $(i, i+1)$ se procedează la interschimbarea $\texttt{a}[\texttt{i}] \leftrightarrow \texttt{a}[\texttt{i}+1]$. La prima parcurgere, elementul cel mai mare din secvenţă formează inversiuni cu toate elementele aflate după el şi, în urma interschimbărilor realizate, acesta va fi deplasat pe ultimul loc care este şi locul său final. În iteraţia următoare, se va întâmpla la fel cu cel de-al doilea element cel mai mare. În general, dacă subsecvenţa $\texttt{a}[\texttt{r}+1..\texttt{n}-1]$ nu are nici o inversiune la iteraţia curentă, atunci ea nu va avea inversiuni la nici una din iteraţiile următoare. Aceasta permite ca la iteraţia următoare să fie verificată numai subsecvenţa $\texttt{a}[0..\texttt{r}]$. Terminarea algoritmului este dată de faptul că la fiecare iteraţie numărul de interschimbari este micşorat cu cel puţin 1.

1.2 Algoritmul *bubbleSort* - pseudocod

```
procedure bubbleSort(a, n)
   ultim ← n-1
   while (ultim > 0) do
      n1 ← ultim - 1
      ultim ← 0
      for i ← 0 to n1 do
          if (a[i] > a[i+1])
             then interschimba(a[i], a[i+1])
                  ultim ← i
end
```

1.3 Evaluarea algoritmului

- Cazul cel mai favorabil este întâlnit atunci când secvenţa de intrare este deja sortată, caz în care algoritmul `bubbleSort` execută $O(n)$ operaţii.
- Cazul cel mai nefavorabil este obţinut când secvenţa de intrare este ordonată descrescător şi, în această situaţie, procedura execută $O(n^2)$ operaţii.

2 Sortare prin inserţie directă

2.1 Descriere

Algoritmul sortării prin inserţie directă consideră că în pasul k, elementele `a[0..k-1]` sunt sortate crescător, iar elementul `a[k]` va fi inserat, astfel încât, după această inserare, primele elemente `a[0..k]` să fie sortate crescător.

Inserarea elementului `a[k]` în secvenţa `a[0..k-1]` presupune:

1. memorarea elementului într-o variabilă temporară;
2. deplasarea tuturor elementelor din vectorul `a[0..k-1]` care sunt mai mari decât `a[k]`, cu o poziţie la dreapta (aceasta presupune o parcurgere de la dreapta la stânga);
3. plasarea lui `a[k]` în locul ultimului element deplasat.

2.2 Algoritmul de sortare prin inserţie directă - pseudocod

```
procedure insertionSort(a, n)
   for k ← 1 to n-1 do
       i ← k-1
       temp ← a[k]
       while ((i ≥ 0) and (temp < a[i])) do
          a[i+1] ← a[i]
          i ← i-1
       if (i ≠ k-1) then a[i+1] ← temp
end
```

2.3 Evaluarea algoritmului

- Căutarea poziţiei i în subsecvenţa `a[0..k-1]` necesită $O(k-1)$ timp.
- Timpul total în cazul cel mai nefavorabil este $O(1+\cdots+n-1) = O(n^2)$.
- Pentru cazul cel mai favorabil, când valoarea tabloului la intrare este deja în ordine crescătoare, timpul de execuţie este $O(n)$.

3 Sortare prin selecţie naivă

3.1 Descriere

Este o metodă mai puţin eficientă, dar foarte simplă în prezentare. Se bazează pe următoarea caracterizare a predicatului $SORT(\mathtt{a})$:

$$SORT(\mathtt{a}) \iff (\forall i)(0 \le i < n) \Rightarrow a[i] = \max\{a[0], \ldots, a[i]\}$$

Ordinea în care sunt aşezate elementele pe poziţiile lor finale este $n-1, n-2, \ldots, 0$. O formulare echivalentă este:

$$SORT(\mathtt{a}) \iff (\forall i)(0 \le i < n) : a[i] = \min\{a[i], \ldots, a[n]\},$$

caz în care ordinea de aşezare este $0, 1, \ldots, n-1$.

3.2 Algoritmul de sortare prin selecţie naivă - pseudocod

```
procedure naivSort(a, n)
for i ← n-1 downto 1 do
    locmax ← 0
    maxtemp ← a[0]
    for j ← 1 to i do
        if (a[j] > maxtemp)
           then locmax ← j
                maxtemp ← a[j]
    a[locmax] ← a[i]
    a[i] ← maxtemp
end
```

3.3 Evaluarea algoritmului

Timpul de execuţie este $O(n^2)$ pentru toate cazurile, adică algoritmul `NaivSort` are timpul de execuţie $\Theta(n^2)$.

4 Sarcini de lucru şi barem de notare

Sarcini de lucru:

1. Scrieţi o funcţie C/C++ care implementează algoritmul bubbleSort.
2. Scrieţi o funcţie C/C++ care implementează algoritmul insertionSort.

3. Scrieţi o funcţie C/C++ care implementează algoritmul naivSort.
4. Măsuraţi timpii de execuţie pentru n numere, unde $10.000 \leq n \leq 10.000.000$. Comparaţi rezultatele.

Barem de notare:

1. Funcţia bubbleSort: 2p
2. Funcţia insertionSort: 2p
3. Funcţia naivSort: 2p
4. Măsurarea şi compararea timpilor de execuţie: 3p
5. Baza: 1p

Bibliografie

[1] Lucanu, D. şi Craus, M., *Proiectarea algoritmilor*, Editura Polirom, 2008.