Proiectarea algoritmilor

Sortare - bubleSort, naivSort, InsertionSort

Lucrare de laborator nr. 5

Cuprins

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

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

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

- O pereche (i,j), cu $i < j$, formează o *inversiune (inversare)*, dacă $a[i] > a[j]$.
- Pe baza definiției de mai sus vom spune că tabloul `a` este sortat dacă și numai dacă nu există nici o inversiune $(i, i+1)$.
- Metoda *bubble-sort* propune parcurgerea iterativă a tabloului `a` și, la fiecare parcurgere, ori de câte ori se întâlnește o inversiune $(i, i+1)$ se procedează la interschimbarea $\mathtt{a[i]} \leftrightarrow \mathtt{a[i+1]}$.

Sortarea prin interschimbarea elementelor vecine - continuare

- La prima parcurgere, elementul cel mai mare din secvență formează inversiuni cu toate elementele aflate după el și, în urma interschimbărilor realizate, acesta va fi deplasat pe ultimul loc care este și locul său final.
- În iterația următoare, se va întâmpla la fel cu cel de-al doilea element cel mai mare.
- În general, dacă subsecvența $\mathtt{a}[\mathtt{r}+1..\mathtt{n}-1]$ nu are nici o inversiune la iterația curentă, atunci ea nu va avea inversiuni la nici una din iterațiile următoare.
- Aceasta permite ca la iterația următoare să fie verificată numai subsecvența $\mathtt{a}[0..\mathtt{r}]$.
- Terminarea algoritmului este dată de faptul că la fiecare iterație numărul de interschimbari este micșorat cu cel puțin 1.

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
```

Evaluarea algoritmului

- Cazul cel mai favorabil este întâlnit atunci când secvența de intrare este deja sortată, caz în care algoritmul `bubbleSort` execută $O(n)$ operații.
- Cazul cel mai nefavorabil este obținut când secvența de intrare este ordonată descrescător și, în această situație, procedura execută $O(n^2)$ operații.

Sortare prin inserție directă

- Algoritmul sortării prin inserție 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 secvența `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 poziție la dreapta (aceasta presupune o parcurgere de la dreapta la stânga);
  3. plasarea lui `a[k]` în locul ultimului element deplasat.

Algoritmul de sortare prin inserție 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
```

Evaluarea algoritmului

- Căutarea poziției i în subsecvența `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 execuție este $O(n)$.

Sortare prin selecție naivă

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

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

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

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

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

Algoritmul de sortare prin selecție 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
```

Evaluarea algoritmului

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

Sarcini de lucru și barem de notare

Sarcini de lucru:

1. Scrieți o funcție C/C++ care implementează algoritmul bubbleSort.
2. Scrieți o funcție C/C++ care implementează algoritmul insertionSort.
3. Scrieți o funcție C/C++ care implementează algoritmul naivSort.
4. Măsurați timpii de execuție pentru n numere, unde $10.000 \leq n \leq 10.000.000$. Comparați rezultatele.

Barem de notare:

1. Funcția bubbleSort: 2p
2. Funcția insertionSort: 2p
3. Funcția naivSort: 2p
4. Măsurarea și compararea timpilor de execuție: 3p
5. Baza: 1p

Bibliografie

Lucanu, D. și Craus, M., *Proiectarea algoritmilor*, Editura Polirom, 2008.