Proiectarea algoritmilor

Paradigma *Divide_et_Impera*

Lucrare de laborator nr. 7

Cuprins

Sortare rapidă (Quick Sort) - descriere

- Ca și în cazul algoritmului *Merge Sort*, vom presupune că trebuie sortată o secvență memorată într-un tablou a[p..q].
- Divizarea problemei constă în alegerea unei valori x din $a[p..q]$ și determinarea prin interschimbări a unui indice k cu proprietățile:
  - $p \leq k \leq q$ și $a[k] = x$;
  - $\forall i : p \leq i \leq k \Rightarrow a[i] \leq a[k]$;
  - $\forall j : k < j \leq q \Rightarrow a[k] \leq a[j]$;
- Elementul x este numit *pivot*. În general, se alege pivotul $x = a[p]$, dar nu este obligatoriu.
- Partiționarea tabloului se face prin interschimbări care mențin invariante proprietăți asemănătoare cu cele de mai sus.
- Se consideră două variabile index: `i` cu care se parcurge tabloul de la stânga la dreapta și `j` cu care se parcurge tabloul de la dreapta la stânga. Inițial se ia $i = p + 1$ și $j = q$.
- Proprietățile menținute invariante în timpul procesului de partiționare sunt:

$$\forall i' : p \leq i' < i \Rightarrow a[i'] \leq x \tag{1}$$

și

$$\forall j' : j < j' \leq q \Rightarrow a[j'] \geq x \tag{2}$$

Sortare rapidă (Quick Sort) - descriere (continuare)

- Presupunem că la momentul curent sunt comparate elementele $\texttt{a[i]}$ și $\texttt{a[j]}$ cu $i < j$.
- Distingem următoarele cazuri:
  1. $a[i] \le x$. Transformarea $\texttt{i} \leftarrow \texttt{i} + 1$ păstrează proprietatea (1).
  2. $a[j] \ge x$. Transformarea $\texttt{j} \leftarrow \texttt{j} - 1$ păstrează proprietatea (2).
  3. $a[i] > x > a[j]$. Dacă se realizează interschimbarea $\texttt{a[i]} \leftrightarrow \texttt{a[j]}$ și se face $\texttt{i} \leftarrow \texttt{i} + 1$ și $\texttt{j} \leftarrow \texttt{j} - 1$, atunci sunt păstrate ambele predicate (1) și (2).
- Operațiile de mai sus sunt repetate până când i devine mai mare decât j.
- Considerând $k = i - 1$ și interschimbând $\texttt{a[p]}$ cu $\texttt{a[k]}$ obținem partiționarea dorită a tabloului.
- După sortarea recursivă a subtablourilor $\texttt{a[p..k} - \texttt{1]}$ și $\texttt{a[k} + \texttt{1..q]}$ se observă că tabloul este sortat deja. Astfel partea de asamblare a soluțiilor este vidă.

Quick Sort - pseudocod

```
procedure quickSort1(a, p, q)
   if (p < q)
      then / * determină prin interschimbări indicele k pentru care:
                 p ≤ k ≤ q
                 (∀)i : p ≤ i ≤ k ⇒ a[i] ≤ a[k]
                 (∀)j : k < j ≤ q ⇒ a[k] ≥ a[j]  */
           partitioneaza(a, p, q, k)
           quickSort(a, p, k-1)
           quickSort(a, k+1, q)
end
```

Quick Sort - pseudocod

```
procedure partitioneaza1(a, p, q, k)
   x ← a[p]
   i ← p + 1
   j ← q
   while (i ≤ j) do
      if (a[i] ≤ x then i ← i + 1
      if (a[j] ≥ x then j ← j - 1
      if (i < j)
         then if ((a[i] > x) and (x > a[j]))
                 then interschimba(a[i],a[j])
                      i ← i + 1
                      j ← j - 1
   k ← i-1
   a[p] ← a[k]
   a[k] ← x
end
```

Quick Sort - pseudocod (continuare)

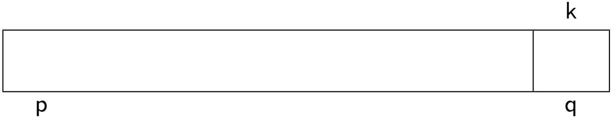
```
procedure partitioneaza2(a, p, q, k)
   x ← a[p]
   i ← p
   j ← q
   while (i < j) do
      while (a[i] ≤ x and i ≤ q) do i ← i + 1
      while (a[j] > x and j ≥ p) do j ← j - 1
      if (i < j)
         then interschimba(a[i],a[j])
   k ← j
   a[p] ← a[k]
   a[k] ← x
end
```

Complexitatea timp a algoritmului Quick Sort

- Cazul cel mai nefavorabil se obține atunci când la fiecare partiționare se obține una din subprobleme cu dimensiunea 1.
- Deoarece operația de partiționare necesită $O(q-p)$ comparații, rezultă că pentru acest caz numărul de comparații este $O(n^2)$.
- Acest rezultat este oarecum surprinzător, având în vedere că numele algoritmului este „sortare rapidă".
- Numele algoritmului se justifică prin faptul că îtr-o distribuție normală, cazurile pentru care quickSort execută n^2 comparații sunt rare, fapt care conduce la o complexitate medie foarte bună a algoritmului.
- Complexitatea medie a algoritmului QuickSort este $O(n \log_2 n)$.

Consumul de memorie

- La execuția algoritmilor recursivi, importantă este și spațiul de memorie ocupat de stivă.
- Considerăm spațiul de memorie ocupat de stivă în cazul cel mai nefavorabil, $k = q$:



- În acest caz, spațiul de memorie ocupat de stivă este $M(n) = c + M(n-1)$, ce implică $M(n) = O(n)$.
- În general, pivotul împarte secvența de sortat în două subsecvențe.
- Dacă subsecvența mică este rezolvată recursiv, iar subsecvența mare este rezolvată iterativ, atunci consumul de memorie se reduce.

Quick Sort - pseudocod îmbunătățit

```
procedure quickSort2(a, p, q)
   while (p < q) do
      partitioneaza(a, p, q, k)
      if (k-p > q-k)
         then quickSort(a, k+1, q)
              q ← k-1
         else quickSort(a, p, k-1)
              p ← k+1
end
```

Consumul de memorie în cazul algoritmului îmbunătățit

- Spațiul de memorie ocupat de stivă pentru algoritmul îmbunătățit satisface relația $M(n) \leq c + M(n/2)$, de unde rezultă $M(n) = O(\log n)$.
- Dacă tabloul a conține multe elemente egale, atunci algoritmul `partitioneaza` realizează multe interschimbări inutile (de elemente egale).
  Exercițiu: Să se modifice algoritmul astfel încât noua versiune să elimine acest inconvenient.

Sarcini de lucru și barem de notare

Sarcini de lucru:

1. Scrieți o funcție C/C++ care implementează o algoritmul quickSort1;
2. Scrieți o funcție C/C++ care implementează o algoritmul quickSort2;
3. Comparați funcțiile quickSort1, quickSort1 și qSort (din biblioteca STL). Pentru aceasta, măsurați timpii de execuție pentru n chei de sortare ($10.000 \leq n \leq 10.000.000$).

Barem de notare:

1. Funcția quickSort1: 4p
2. Funcția quickSort2: 3p
3. Compararea funcțiilor quickSort1, quickSort1 și qSort: 2p
4. Baza: 1p

Bibliografie

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