

# Proiectarea algoritmilor

## Lucrare de laborator nr. 6

### Sortarea prin metoda distribuirii - Algoritmul radixSort

## Cuprins

1	Sortarea prin distribuire	1
2	Sortarea cuvintelor	1
3	radixSort - descriere, pseudocod	2
4	Evaluarea algoritmului	3
5	Sarcini de lucru și barem de notare	3

## 1 Sortarea prin distribuire

Algoritmii de sortare prin distribuire presupun cunoașterea de informații privind distribuția acestor elemente.

Aceste informații sunt utilizate pentru a distribui elementele secvenței de sortat în „pachete” care vor fi sortate în același mod sau prin altă metodă, după care pachetele se combină pentru a obține lista finală sortată.

## 2 Sortarea cuvintelor

Presupunem că avem  $n$  fișe, iar fiecare fișă conține un nume ce identifică în mod unic fișa (cheia). Se pune problema sortării manuale a fișelor. Pe baza experienței câștigate se procedează astfel:

Se împart fișele în pachete, fiecare pachet conținând fișele ale căror cheie începe cu aceeași literă. Apoi se sortează fiecare pachet în aceeași manieră după a doua literă, apoi etc. După sortarea tuturor pachetelor, acestea se concatenează rezultând o listă liniară sortată.

Vom încerca să formalizăm metoda de mai sus într-un algoritm de sortare a șirurilor de caractere (cuvinte). Presupunem că elementele secvenței de sortat sunt șiruri de lungime fixată  $m$  definite peste un alfabet cu  $k$  litere. Echivalent, se poate presupune că elementele de sortat sunt numere reprezentate în baza  $k$ . Din acest motiv, sortarea cuvintelor este denumită în engleză *radix-sort* (cuvântul *radix* traducându-se prin *bază*).

Dacă urmărim ideea din exemplul cu fișele, atunci algoritmul ar putea fi descris recursiv astfel:

1. Se împart cele  $n$  cuvinte în  $k$  pachete, cuvintele din același pachet având aceeași literă pe poziția  $i$  (numărând de la stânga la dreapta).
2. Apoi, fiecare pachet este sortat în aceeași manieră după literele de pe pozițiile  $i + 1, \dots, m - 1$ .

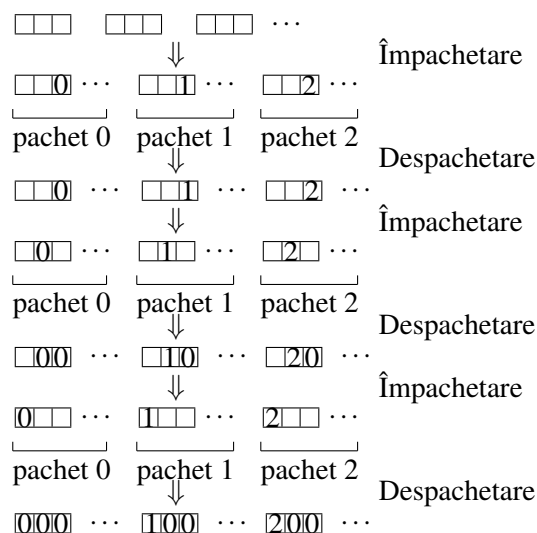


Figura 1: Sortare prin distribuire

3. Se concatenează cele  $k$  pachete în ordinea dată de literele de pe poziția  $i$ . Lista obținută este sortată după subcuvintele formate din literele de pe pozițiile  $i, i + 1, \dots, m - 1$ .

Inițial se consideră  $i = 0$ . Apare următoarea problemă:

Un grup de  $k$  pachete nu va putea fi combinat într-o listă sortată decât dacă cele  $k$  pachete au fost sortate complet pentru subcuvintele corespunzătoare. Este deci necesară ținerea unei evidențe a pachetelor, fapt care conduce la utilizarea de memorie suplimentară și creșterea gradului de complexitate a metodei.

O simplificare majoră apare dacă împărțirea cuvintelor în pachete se face parcurgând literele acestora de la dreapta la stânga. Procedând așa, observăm următorul fapt surprinzător: după ce cuvintele au fost distribuite în  $k$  pachete după litera de pe poziția  $i$ , cele  $k$  pachete pot fi combinate înainte de a le distribui după litera de pe poziția  $i - 1$ .

Exemplu: Presupunem că alfabetul este  $\{0 < 1 < 2\}$  și  $m = 3$ . Cele trei faze care cuprind distribuirea elementelor listei în pachete și apoi concatenarea acestora într-o singură listă sunt sugerate grafic în figura 1.

### 3 radixSort - descriere, pseudocod

Pentru gestionarea pachetelor vom utiliza un tablou de structuri de pointeri numit `pachet`, cu semnificația următoare:

`pachet[i]` este structura de pointeri `pachet[i].prim` și `pachet[i].ultim`,

`pachet[i].prim` face referire la primul element din lista ce reprezintă pachetul  $i$  și

`pachet[i].ultim` face referire la ultimul element din lista corespunzătoare pachetului  $i$ .

Etapă de distribuire este realizată în modul următor:

1. Inițial, se consideră listele `pachet[i]` vide.
2. Apoi se parcurge secvențial lista supusă distribuirii și fiecare element al acesteia este distribuit în pachetul corespunzător.

Etapă de combinare a pachetelor constă în concatenarea celor  $k$  liste `pachet[i]`,  $i = 0, \dots, k - 1$ .

```

procedure radixSort(L,m)
for i ← m-1 downto 0 do
    for j ← 0 to k-1 do
        pachet[j] ← listaVida()
    while (not esteVida(L)) do /* împachetare */
        w ← citește(L, 0)
        elimina(L, 0)
        insereaza(pachet[w[i]], w)
    for j ← 0 to k-1 do /* despachetare */
        concateneaza(L, pachet[j])
end

```

## 4 Evaluarea algoritmului

Distribuirea în pachete presupune parcurgerea completă a listei de intrare, iar procesarea fiecărui element al listei necesită  $O(1)$  operații.

- Faza de distribuire se face în timpul  $O(n)$ , unde  $n$  este numărul de elemente din listă.
- Combinarea pachetelor presupune o parcurgere a tabloului pachet, iar adăugarea unui pachet se face cu  $O(1)$  operații, cu ajutorul tabloului `ultim`.
- Faza de combinare a pachetelor necesită  $O(k)$  timp.

Algoritmul `radixSort` are un timp de execuție de  $O(m \cdot n)$ .

## 5 Sarcini de lucru și barem de notare

### Sarcini de lucru:

1. Scrieți o funcție C/C++ care implementează algoritmul `radixSort`. Se presupune că secvența de sortat este formată din  $n$  numere întregi, cu cifre din baza 10:  $s = (a_0, a_1, \dots, a_{n-1})$ . Fiecare număr  $a_i$ ,  $i = 0, 1, \dots, n-1$ , din secvența de sortat are maxim  $k$  cifre ( $a_i = c_1 c_2 \dots c_k$ ).
2. Măsurați timpul de execuție pentru  $n$  numere, unde  $10.000 \leq n \leq 10.000.000$ .

### Barem de notare:

1. Funcția `radixSort`: 7p
2. Măsurarea timpului de execuție: 2p
3. Baza: 1p

## Bibliografie

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