Algoritmi şi structuri de date (I). Seminar 8: Algoritmi de generare a permutărilor. Aplicaţii ale tehnicii reducerii. Analiza complexităţii algoritmilor recursivi.

Problema 1 Să se genereze toate permutările de ordin n în ordine lexicografică (în ordinea crescătoare a valorii asociate permutării). Pentru $n = 3$ aceasta înseamnă generarea valorilor în ordinea: $(1, 2, 3)$, $(1, 3, 2)$, $(2, 1, 3)$, $(2, 3, 1)$, $(3, 1, 2)$, $(3, 2, 1)$.

Indicaţie. Se porneşte de la permutarea identică $p[1..n]$, $p[i] = i, i = \overline{1, n}$ (care pentru $n = 3$ are asociată valoarea $123 \ldots n$) şi se generează succesiv valoarea imediat următoare constituită din aceleaşi cifre. Pentru fiecare nouă permutare generată se parcurg etapele:

- identifică cel mai mare indice $i \in \{2, 3, \ldots, n\}$ cu proprietatea $p[i] > p[i-1]$;
- determină poziţia, k, a celei mai mici valori din $p[i..n]$ cu proprietatea că $p[k] > p[i-1]$;
- interschimbă $p[k]$ cu $p[i-1]$;
- inversează ordinea elementelor din subtabloul $p[i..n]$.

Problema 2 *Algoritmul Johnson-Trotter.* Să se genereze permutările de ordin n astfel încât fiecare permutare să fie obţinută din cea imediat anterioară prin interschimbarea a exact două elemente. Acest lucru se poate realiza prin algoritmul Johnson-Trotter ale cărui etape principale sunt:

- Se porneşte de la permutarea identică; asociază fiecărui element un sens de parcurgere (iniţial toate elementele au asociat un sens de parcurgere înspre stânga).
- Determină elementul mobil maximal (un element este considerat mobil dacă sensul ataşat lui indică către o valoare adiacentă mai mică).
- Interschimbă elementul mobil cu elementul mai mic către care indică.
- Modifică direcţia tuturor elementelor mai mari decât elementul mobil.

Ultimele trei etape de mai sus se repetă până nu mai poate fi găsit un element mobil. În cazul permutărilor de ordin 3 algoritmul conduce la următoarea secvenţă de rezultate:

$\overleftarrow{1}\,\overleftarrow{2}\,\overleftarrow{3}$
$\overleftarrow{1}\,\overleftarrow{3}\,\overleftarrow{2}$
$\overleftarrow{3}\,\overleftarrow{1}\,\overleftarrow{2}$
$\overrightarrow{3}\,\overleftarrow{2}\,\overleftarrow{1}$
$\overleftarrow{2}\,\overrightarrow{3}\,\overleftarrow{1}$
$\overleftarrow{2}\,\overleftarrow{1}\,\overrightarrow{3}$

Indicaţie. Sensurile de parcurgere pot fi uşor modelate printr-un tablou $d[1..n]$ în care valoarea -1 corespunde orientării înspre stânga iar valoarea $+1$ corespunde orientării înspre dreapta.

Problema 3 Să se genereze toate şirurile cu n elemente din $\{0, 1\}$. De exemplu, pentru $n = 3$ sunt opt şiruri binare: $(0, 0, 0)$, $(0, 0, 1)$, $(0, 1, 0)$, $(0, 1, 1)$, $(1, 0, 0)$, $(1, 0, 1)$, $(1, 1, 0)$, $(1, 1, 1)$. Stabiliţi ordinul de complexitate al algoritmului propus.

Indicaţie. Sunt mai multe variante de rezolvare:

(i) Se parcurg valorile de la 0 la $2^n - 1$ şi pentru fiecare dintre ele se construieşte reprezentarea în baza 2 pe n poziţii.

(ii) Se "numără" direct în baza 2 pornind de la şirul valorilor egale cu 0 şi "incrementând" cu 1 la fiecare etapă (incrementarea se realizează prin adnarea lui 1 în baza 2 (se parcurge tabloul de la dreapta la stânga, toate valorile egale cu 1 sunt transformate în 0, iar prima valoare egală cu 0 este transformată în 1; dacă nu există valoare egală cu 0 înseamnă că s-a ajuns la $2^n - 1$).

(iii) Se aplică tehnica reducerii:

```
generare(k)
if k == 1 then
    p[1] = 0
    write (p[1..n])
    p[1] = 1
    write (p[1..n])
else
    p[k] = 0
    generare(k − 1)
    p[k] = 1
    generare(k − 1)
endif
```

Algoritmul de mai sus se apelează pentru $k = n$ (`generare`(n)) presupunând că $p[1..n]$ este o variabilă globală. La fiecare apel al funcţiei se completează poziţia corespunzătoare lui k cu 0, respectiv 1 după care se apelează recursiv algoritmul pentru a genera toate subşirurile binare cu $k - 1$ elemente.

Problema 4 Să se calculeze A^p unde A este o matrice $n \times n$ şi p este o valoare naturală mai mare decât 1. Să se analizeze complexitatea algoritmului propus.

Indicaţie. Se presupune că `produs`$(A[1..n, 1..n], B[1..n, 1..n])$ este un algoritm care returnează produsul matricilor A şi B specificate ca parametri de intrare. Problema poate fi rezolvată folosind tehnica forţei brute sau tehnica reducerii (folosind ideea de la calculul puterii naturale a unui număr).

Probleme suplimentare

1. Să se rescrie algoritmul de incrementare cu 1 a unui număr reprezentat în baza 2 fără a folosi operaţii de împărţire.
   Indicaţie. Se parcurge tabloul cu cifrele binare de la cifra cea mai puţin semnificativă către cea mai semnificativă şi se pune pe 1 prima cifră egală cu 0 întâlnită iar cifrele egale cu 1 întâlnite până la primul 0 se înlocuiesc cu 0.

2. Să se genereze toate cele 2^n submulţimi ale unei mulţimi cu n elemente.
   Indicaţie. O submulţime poate fi reprezentată prin tabelul indicatorilor de prezenţă care este un şir binar cu n elemente, prin urmare generarea tuturor submulţimilor unei mulţimi cu n elemente este echivalentă cu generarea tuturor şirurilor binare cu n elemente.

3. Folosind tehnica divizării algoritmul de înmulţire a două matrici poate fi reorganizat astfel încât ordinul de complexitate să fie redus. Un exemplu în acest sens este algoritmul lui Strassen al cărui ordin de complexitate este $\mathcal{O}(n^{2.7})$ în cazul înmulţirii a două matrici pătratice de dimensiune $n \times n$. Consultaţi resurse web (folosind "Strassen algorithm" drept cheie de căutare) unde este descris algoritmul şi încercaţi să îl implementaţi.

4. Stabiliţi ce afişează algoritmul de mai jos atunci când este apelat pentru $k = n$ (în ipoteza că lucrează asupra unui tablou global $a[1..n]$ iniţializat astfel încât $a[i] = i$) şi stabiliţi ordinul de complexitate al algoritmului.

```
alg(integer k)
if k == 1 then write a[1..n]
else for i = 1, k do
    alg(k − 1)
    if kMOD2 == 1 then a[1] ↔ a[k]
        else a[i] ↔ a[k]
    endif
endfor
endif
```