Algoritmi de tip succesor

- Se pot utiliza în cazul problemelor în care soluția este reprezentată sub forma unui şir (vector, cu componente alese din mulțimi finite, nevide şi ordonate crescător) şi este necesară determinarea şirului următor sau precedent în ordine lexicografică[1]

- Poate fi generalizat pentru generarea tuturor şirurilor de acest tip, ordonate lexicografic

- Algoritmi fundamentali: generare submulțimi, produs cartezian, combinări, aranjamente, permutări

Algoritm general pentru determinarea soluției următoare în ordine lexicografică:

Pas 1. Se inițializează şirul cu configurația dată $x=(x_1, x_2,...x_n)$

Pas 2. Se determină, dacă există, cel mai mare indice `k`, $1 \le k \le n$, cu proprietatea că elementul `x[k]` are succesor în mulțimea din care ia valori

Pas 2.1. dacă nu există `k`, soluția dată este cea mai mare în ordine lexicografică

Pas 2.2. dacă există `k`, `x[k]` se actualizează cu elementul următor lexicografic din mulțimea corespunzătoare şi se completează soluția cu `x[k+1], x[k+2]… x[n]` alese astfel încît să reprezinte secvența minimă corectă conform ordinii lexicografice.

Generare submulțimi

Fie `n` un număr natural nenul. Să se genereze toate submulțimile mulțimii {`0`, `1`, …, `n-1`}.

Soluție

Vom reprezenta submulțimile prin vector caracteristic (un vector `x` cu `n` componente `0` sau `1`, având semnificația `x[i]=1`, dacă `i` aparține submulțimii şi `0` în caz contrar). Problema se reduce astfel la generarea tuturor secvențelor binare de lungime `n`. Pentru a rezolva această problemă vom utiliza un algoritm de tip succesor astfel:
Pas 1. Se inițializează soluția cu mulțimea vidă (configurație minimă în ordine lexicografică – un şir de n valori 0)
Pas 2. Cât timp este există succesor
- se afişează configurația curentă
- se generează configurația următoare lexicografic: determinăm cel mai mare k astfel încât x[k] admite succesor (identificăm prima valoare `0` de la dreapta la stânga); `x[k]=1` şi toate valorile `x[k+1], x[k+2]… x[n]` devin `0` (secvența minimă în ordine lexicografică)
- când `k<1` (toate valorile sunt egale cu 1 în soluție), nu mai există successor şi algoritmul se opreşte

Generare elemente produs cartezian

Fie `n` un număr natural (`n>1`) şi $M=(m_1, m_2, ..., m_n)$ `n` numere naturale nenule. Să se determine în ordine lexicografică toate elementele produsului cartezian
$\{1, 2, ..., m_1\} x \{1, 2, ..., m_2\} x ... x \{1, 2, ..., m_n\}$.

[1] Fie $x=(x_1, x_2, ..., x_n)$ şi $y=(y_1, y_2, ..., y_m)$. Spunem că `x` este mai mic lexicografic decât `y` dacă există `k` astfel încât $x_k<y_k$ şi $x_i=y_i$, orice $i<k$ sau $x_i=y_i$ orice $i \le n \le m$.

De exemplu, pentru n=3 şi L=(2, 3, 2), elementele produsului cartezian sunt:
(1,1,1), (1,1,2), (1,2,1), (1,2,2), (1,3,1), (1,3,2), (2,1,1), (2,1,2), (2,2,1), (2,2,2), (2,3,1), (2,3,2).

Produsul cartezian are $m_1*m_2*...*m_n$ elemente.

Soluție

Reprezentăm soluția ca un vector x cu n elemente, unde $x[i] \in \{1, 2, ..., m_i\}$.
Generăm elementele produsului cartezian în ordine lexicografică cu un algoritm de tip succesor astfel:

Pas 1. Se inițializează vectorul x cu 1 (cel mai mic element al produsului cartezian, din punct de vedere lexicografic).

Pas 2. Cât timp există succesor
- se afişează elementul curent;
- se generează configurația următoare lexicografic: determinăm cel mai mare k astfel încât x[k] poate fi mărită (adică x[k]<m[k]); dacă găsim o astfel de componentă o mărim cu 1 şi resetăm cu 1 toate componentele următoare; dacă nu găsim o astfel de componentă deducem că generarea s-a încheiat, acesta a fost cel mai mare şirs din punct de vedere lexicografic.

Generare combinări

Fie n şi m două numere naturale, $1 \le m \le n \le 20$.

Să se genereze în ordine lexicografică toate submulțimile formate din m elemente ale mulțimii {1, 2, ..., n} (ordinea elementelor în submulțime nu este semnificativă).

Soluție

Reprezentăm soluția ca un vector x cu m elemente distincte, alese din mulțimea

$\{1, 2, ..., n\}$, $1 \le x[i] \le n$, $1 \le i \le m$.

Deoarece ordinea elementelor în submulțime nu contează, pentru a nu obține de mai multe ori aceeaşi soluție, impunem ca elementele submulțimii să fie generate în ordine crescătoare: x[i]<x[i+1], pentru orice $1 \le i < m$.

Cea mai mică submulțime conform ordinii lexicografice este {1, 2, ..., m} iar cea mai mare submulțime este {n-m+1,..., n-1, n}

Observăm că atunci când poziția scade cu 1, valoarea maximă scade cu 1, deci diferența dintre orice indice k şi maxim ce poate fi ales pe poziția k este constantă:

maxim-k = n-m

Deducem că valoarea maximă care poate fi plasată pe poziția k este n-m+k.

Generăm submuțimile printr-un algoritm de tip successor astfel:

Pas 1. Se inițializează vectorul x cu cea mai mică submulțime în ordine lexicografică (x[i]=i, pentru orice $1 \le i \le m$)

Pas 2. Cât timp există succesor
- se afişează submulțimea curentă;
- se generează submulțimea următoare: căutăm prima componentă (începând din dreapta către stânga) care poate fi mărită (adică x[i]<n-m+i); dacă găsim o astfel de componentă o mărim, şi repunem pe cea mai mică valoare posibilă toate componentele următoare ($x[j]=x[j-1]+1$, $i<j \le m$); dacă nu găsim o astfel de componentă deducem că generarea s-a încheiat, acesta a fost cea mai mare submulțime din punct de vedere lexicografic.

Aplicații

1. Cod de identificare

Pentru a concura cu numărul de serie de la procesoarele Intel Pentium III, Advanced Micro Devices a stabilit un sistem de identificare pentru noile procesoare cu numele de cod Thunderbird. Fiecare firmă distribuitoare primeşte o mulțime de litere (de exemplu {a, m, x}) din care va trebui să-şi formeze codurile proprii de identificare. Firmelor li se impune exact de câte ori trebuie să apară fiecare literă în aceste coduri. De exemplu, o firmă trebuie să formeze identificatori care să conțină exact 3 litere a, 2 litere m şi 1 literă x. Scrieți un program care, cunoscând un anumit cod dat, determină următorul cod corect în ordine lexicografică, dacă există un astfel de cod următor. (ONI Constanța 2000, IX)

Date de intrare

Fişierul de intrare COD.IN conține un cod. Codurile sunt formate din cel mult 100 de litere mici ale alfabetului latin.

Date de ieşire

Fişierul de ieşire COD.OUT va conține o singură linie pe care se va afla codul următor; dacă nu există un astfel de cod, atunci în fişier se va scrie 'Este ultimul cod.'

Exemple

```
COD.IN              COD.OUT
amaaxm              amamax

COD.IN              COD.OUT
xmmaaa              Este ultimul cod.
```

2. Paranteze

Fie n un număr natural (1<n<10000) şi p1p2p3...pn o succesiune de paranteze rotunde care se închid corect. Determinați parantezarea imediat următoare din punct de vedere lexicografic, dacă există. Dacă nu există, se va scrie „eroare".

Exemplu

parant.in

```
8
(()())()
```

parant.out

```
(())(())
```

3. Campion.edu.ro/arhiva

- adunscad
- pluricex
- grupuri

Bibliografie

Cerchez, Emanuela, Şerban, Marinel, *Programarea în Limbajul C/C++ pentru liceu, Ed. Polirom, Iaşi, 2005*

http://www.campion.edu.ro/arhiva