

Magic

Prof. Zoltan Szabo, Gr. Șc. „P. Maior”, Reghin

	2	5	4
3			
1			
6			

fig. 1

	2	5	4
3	1	9	9
1	1	1	1
6	9	9	1

fig. 2

$o_2=1$	1	1	1
$v_1=2$	1	1	9
$o_1=3$	1	9	9
$v_3=4$	9	1	1
$v_2=5$	9	1	9
$o_3=6$	9	9	1

fig. 3

	v_1	v_2	v_3
o_1			
o_2			
o_3			

fig. 4

Cum se observa din figura 3, cele doua siruri o si v formează o permutare a numerelor $1...2n$, iar conținutul liniilor este echivalent cu numerele naturale $0..2^n-1$.

Intr-adevăr dacă înlocuim 1 cu 0 și 9 cu 1 și toate cifrele de pe o linie le scriem grupate, pentru $n=3$ vom avea următoarele 8 cazuri:

Linia matricei	schimbat in 0-1	echivalentul in baza 10
111	000	0
119	001	1
191	010	2
199	011	3
911	100	4
919	101	5
991	110	6
999	111	7

Problema se va rezolva cu tehnica backtracking, generand toate combinările numerelor $0,1,...,2^n-1$, luate câte $2n$, fiecare linie sau coloana a matricei va corespunde unui număr care se va introduce bit cu bit în matrice pe linie sau coloană după cum se cere în enunț.

Dacă un element al matricei completat orizontal intră în contradicție cu un element completat vertical atunci e caz de revenire, Dacă un câmp nu era completat sau era completat cu număr identic, atunci putem continua backtrackingul pentru nivelul următor.

Cu aceasta metodă se memorează prima soluție. celelalte soluții se vor număra, astfel vom avea răspuns la ambele cerințe ale problemei.

Reacții

Prof. Marinel Șerban, Liceul de Informatică „Grigore Moisil” Iași

Soluția 1 – utilizarea unei stive

Se utilizează o stivă S , ale cărei elemente sunt subsecvențe $1, 1+1, \dots, 1+k$ ale mulțimii $\{1, 2, \dots, n\}$. De fapt, un element al stivei are doar două componente, elementul minim și respectiv elementul maxim din subsecvență. În general, pentru a adăuga o substanță s la stiva S , se verifică dacă substanța din vârful stivei S formează o nouă substanță împreună cu s , cu alte cuvinte dacă reuniunea celor două seturi de valori este o substanță (conform regulii). În caz afirmativ, se extrage substanța din vârful stivei S , se formează noua substanță, apoi se repetă pașii anteriori (verificare, extragere, formare nouă substanță). Dacă reuniunea a două seturi de valori (substanțe) nu formează o nouă substanță, atunci ultima nouă substanță obținută se pune în vârful stivei S .

Astfel, setul de substanțe este parcurs o singură dată; se consideră inițial fiecare element ca fiind o substanță separată și se adaugă fiecare dintre aceste substanțe la S , în modul descris mai sus.

Dacă la terminarea parcurgerii setului de valori, stiva S conține un singur element (o singură substanță), adică valorile 1 și n , substanța finală este stabilă, în caz contrar nu.

Pentru valorile $2, 1, 4, 5, 3$ și notațiile tradiționale

- | | |
|----------|--|
| Push e | - pune în vârful stivei elementul e |
| Pop e | - scoate din vârful stivei elementul e |
| Add e | - adaugă elementul e la stivă conform regulilor descrise |

procesul arată astfel:

Add 2 : Push 2	Stack : (2,2)
Add 1 : Pop 2, [Add (2,1) : Push (2,1)]	Stack : (2,1)
Add 4 : Push 4	Stack : (4) (2,1)
Add 5 : Pop 4, [Add (4,5) : Push (4,5)]	Stack : (4,5) (2,1)
Add 3 : Pop (4,5), [Add (4,5,3) : Pop (2,1), Add (2,1,4,5,3) : Push (2,1,4,5,3)]	Stack : (1,5)

Modalitatea de verificare a obținerii unei substanțe stabile poate să nu fie unică. De exemplu pentru substanțele $4, 5, 3, 1, 2$ pot exista două modalități de combinare:

$(((4, 5), 3), (1, 2))$

a) $(((4, 5), (3, (1, 2)))$

Soluția indicată mai sus validează varianta a).

Observație

Funcție de implementare algoritmul poate utiliza mai multă sau mai puțină memorie. Dacă verificarea posibilității de combinare se face utilizând vector caracteristic și în stivă se reține întreaga substanță, se utilizează memorie multă și timpul este mai mare. Dacă fiecare element este reținut pe stivă doar prin două valori (min și max numerelor de ordine a substanțelor care o compun), memoria utilizată este mult mai mică și, de asemenea, timpul de verificare a posibilității de combinare este foarte mic.

Soluția2 – divide & impera

Se consideră că substanța finală este stabilă și se încearcă (utilizând divide & impera) să se detecteze cele două substanțe care o compun. Apoi, recursiv, pentru fiecare substanță detectată se repetă procedeul. Dacă în final s-au obținut n substanțe distincte, substanța inițială a fost stabilă, iar dacă, la un anumit pas, nu s-au putut separa, pentru o anumită substanță, cele două componente, substanța inițială nu a fost stabilă. Funcție de implementări și de optimizări de pot obține între 40-60% din punctaj.

Soluția3 - brut

Se caută două substanțe care pot reacționa, se reunesc, apoi se reia procesul de la început. Funcție de implementare se pot obține până la 50 % din punctaj.

TEXT - Descrierea soluției

Prof. Emanuela Cerchez, Liceul de Informatică „Grigore Moisil” Iași

Reprezentarea informațiilor

Vom memora șirul dat în variabila `s`

Vom nota cu `lg` lungimea șirului citit.

Vom utiliza un vector `poz`: `poz[i]`=poziția celei mai apropiate vocale care urmează după poziția `i`; (sau `-1` dacă nu mai urmează nici o vocală).

Idee 1.

Vom rezolva problema prin metoda backtracking.

Vom genera vectorul `sol` cu `n-1` componente; `sol[i]`=poziția de început al celui de-al `i`-lea cuvânt.

Considerăm că numerotăm cuvintele de la 0 (cuvântul 0 începe obligatoriu pe poziția 0).

În vectorul `solmin` memorăm soluția optimă, iar în variabila `nr` contorizăm soluțiile.

Când apelăm funcția recursivă `rezolva(k)` sunt fixate deja începuturile cuvintelor 0, 1, ..., `k-1`.

Trebuie să fixăm începutul celui de-al `k`-lea cuvânt.

În acest scop, vom fixa începutul astfel încât cuvântul precedent să conțină cel puțin o vocală, să mai existe vocale disponibile și pentru cuvântul curent, iar lungimea cuvântului să nu depășească 20 de caractere.

Pe parcursul generării calculăm și costul soluției deja construite.

Când parametrul `k==n` deducem că soluția este completă.

Verificăm dacă este validă (ultimul cuvânt are cel mult 20 de litere) și în caz afirmativ o contorizăm.

Apoi comparăm costul soluției curente cu costul minim și în cazul în care este mai mic reținem această soluție drept soluție optimă.

O astfel de abordare obține 40 puncte.

Idee 2.

Cerința 1.

Vom aborda problema prin programare dinamică.

Subproblemă:

Să se determine numărul de modalități de a construi k cuvinte din sufixul șirului care începe la poziția i ($0 < i < Lg$; $1 \leq k \leq N$).

Pentru aceasta vom construi un cuvânt de maxim 20 de caractere care începe pe poziția i (în toate modurile posibile).

$nr[i][k] = \sum nr[i+t][k-1]$, unde $1 \leq t \leq 20$ (cel mult 20 de litere) și $i+t \leq poz[i]$ (trebuie să conțină o vocală măcar)

Evident, nu însumăm valorile -1 (care indică imposibilitatea de construire a unei soluții).

Soluția problemei se va afla în $nr[0][N]$

Observăm că pentru a determina valorile nr nu este necesară toată matricea ci doar două coloane succesive (coloana curentă și coloana precedentă).

Cerința 2.

Vom rezolva și această cerință prin programare dinamică.

Subproblemă:

Să se determine cea mai armonioasă împărțire în k cuvinte a sufixului șirului s care începe la poziția i .

Vom memora în

$cost[i][k]$ = costul împărțirii (suma diferențelor absolute ale lungimilor cuvintelor consecutive) - sau $Infinit$ dacă o astfel de împărțire nu este posibilă

$urm[i][k]$ = poziția de început a următorului cuvânt (sau lg dacă un astfel de cuvânt nu mai există)

Modul de gândire este asemănător cu cel de la cerința 1, numai că în loc de a însuma numărul de posibilități, vom determina un minim:

$cost[i][k] = \min\{ t*t + cost[i+t][k-1], \text{ unde } 1 \leq t \leq 20 \text{ (cel mult 20 de litere) și } i+t \leq poz[i] \}$

$urm[i][k] = i + t_{min}$ (t_{min} fiind valoarea lui t pentru care se obține minimumul în relația precedentă).

Din relația de recurență a costului deducem că nu este necesară memorarea întregii matrice $cost$, sunt necesare doar 2 coloane.

În schimb, pentru reconstituirea soluției este necesară întreaga matrice urm .