Magic

Prof. Zoltan Szabo, Gr. Şc. „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...2*n,* iar conţinutul liniilor este echivalent cu numerele naturale $0..2^n-1$.

Intr-adevăr dacă înlocuim 1 cu 0 şi 9 cu 1 şi 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 combinarile 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 enunţ.

Dacă un element al matricei completat orizontal intră în contradicţie 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 soluţie. celelalte soluţii se vor număra, astfel vom avea răspuns la ambele cerinţe ale problemei.

Reacţii

Prof. Marinel Şerban, Liceul de Informatică „Grigore Moisil” Iaşi

Soluţia 1 – utilizarea unei stive

Se utilizează o stivă `S`, ale cărei elemente sunt subsecvenţe `l, l+1, ..., l+k` ale mulţimii `{1, 2, ..., n}`. De fapt, un element al stivei are doar două componente, elementul minim şi respectiv elementul maxim din subsecvenţă. În general, pentru a adăuga o substanţă `s` la stiva `S`, se verifică dacă substanţa din vârful stivei `S` formează o nouă substanţă împreună cu `s`, cu alte cuvinte dacă reuniunea celor două seturi de valori este o substanţă (conform regulii). În caz afirmativ, se extrage substanţa din vârful stivei `S`, se formează noua substanţă, apoi se repetă paşii anteriori (verificare, extragere, formare nouă substanţă). Dacă reuniunea a două seturi de valori (substanţe) nu formează o nouă substanţă, atunci ultima nouă substanţă obţinută se pune în vârful stivei `S`.

Astfel, setul de substanţe este parcurs o singură dată; se consideră iniţial fiecare element ca fiind o substanţă separată şi se adaugă fiecare dintre aceste substanţe la `S`, în modul descris mai sus.

Dacă la terminarea parcurgerii setului de valori, stiva `S` conţine un singur element (o singură substanţă), adică valorile `1` şi `n`, substanţa finală este stabilă, în caz contrar nu.

Pentru valorile `2, 1, 4, 5, 3` şi notaţiile tradiţionale

`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 obţinerii unei substanţe stabile poate să nu fie unică. De exemplu pentru substanţele `4, 5, 3, 1, 2` pot exista două modalităţi de combinare:

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

Soluţia indicată mai sus validează varianta `a)`.

Observaţie

Funcţie de implementare algoritmul poate utiliza mai multă sau mai puţină memorie. Daca verificarea posibilităţii de combinare se face utilizând vector caracteristic şi în stivă se reţine întreaga substanţă, se utilizează memorie multă şi timpul este mai mare. Dacă fiecare element este reţinut pe stivă doar prin două valori (min şi max numerelor de ordine a substanţelor care o compun), memoria utilizată este mult mai mică şi, de asemenea, timpul de verificare a posibilităţii de combinare este foarte mic.

Soluţia2 – divide & impera

Se consideră că substanţa finală este stabilă şi se încearcă (utilizând divide & impera) să se detecteze cele două substanţe care o compun. Apoi, recursiv, pentru fiecare substanţă detectată se repetă procedeul. Dacă în final s-au obţinut `n` substanţe distincte, substanţa iniţială a fost stabilă, iar dacă, la un anumit pas, nu s-au putut separa, pentru o numită substanţă, cele două componente, substanţa iniţială nu a fost stabilă. Funcţie de implementări şi de optimizări de pot obţine între 40-60% din punctaj.

Soluţia3 - brut

Se caută două substanţe care pot reacţiona, se reunesc, apoi se reia procesul de la început. Funcţie de implementare se pot obţîne până la 50 % din punctaj.

TEXT - Descrierea soluţiei

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

Reprezentarea informaţiilor

Vom memora şirul dat în variabila `s`

Vom nota cu `lg` lungimea şirului citit.

Vom utiliza un vector `poz`: `poz[i]`=pozitia celei mai apropiate vocale care urmează după poziţia `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]`=poziţia de început al celui de-al `i`-lea cuvânt.

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

În vectorul `solmin` memorăm soluţia optimă, iar în variabila `nr` contorizăm soluţiile.

Când apelăm funcţia 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 conţină cel puţin o vocală, să mai existe vocale disponibile şi pentru cuvântul curent, iar lungimea cuvântului să nu depăşească 20 de caractere.

Pe parcursul generării calculăm şi costul soluţiei deja construite.

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

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

Apoi comparăm costul soluţiei curente cu costul minim şi în cazul în care este mai mic reţinem această soluţie drept soluţie optimă.

O astfel de abordare obţine 40 puncte.

Idee 2.

Cerinţa 1.

Vom aborda problema prin programare dinamică.

Subproblemă:

Să se determine numărul de modalităţi de a construi `k` cuvinte din sufixul şirului care începe la poziţia `i` (`0<i<Lg`; `1≤K≤N`).

Pentru a aceasta vom construi un cuvânt de maxim `20` de caractere care incepe pe pozitia `i` (în toate modurile posibile).

`nr[i][k]=Σnr[i+t][k-1]`, unde `1<=t<=20` (cel mult `20` de litere) şi `i+t >poz[i]` (trebuie să conţină o vocală măcar)

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

Solutia problemei se va afla în `nr[0][N]`

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

Cerinţa 2.

Vom rezolva şi această cerinţă prin programare dinamică.

Subproblemă:

Să se determine cea mai armonioasă împărţire în `k` cuvinte a sufixului şirului `s` care începe la poziţia `i`.

Vom memora în

`cost[i][k]=` costul împărţirii (suma diferenţelor absolute ale lungimilor cuvintelor consecutive) - sau `Infinit` dacă o astfel de împărţire nu este posibilă

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

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

`cost[i][k]= min{ t*t + cost [i+t][k-1]`, unde `1<=t<=20` (cel mult 20 de litere) şi `i+t >poz[i]}`

`urm[i][k]=i+tmin` (tmin fiind valoarea lui `t` pentru care se obţine minimul în relaţia precedentă).

Din relaţia de recurenţă a costului deducem că nu este necesară memorarea întregii matrice `cost`, sunt necesare doar 2 coloane.

În schimb, pentru reconstituirea soluţiei este necesară întreaga matrice `urm`.