Secţiunea 11-12

DESCRIERE SOLUŢII

PROBLEMA 1 OLTENESC

Soluţie de 100 puncte

Să zicem că N este retinut ca un vector de cifre, N[0...sz-1], unde N[sz-1] reprezintă cifra unitătilor.
Mai întâi, observăm că numărul de puteri de 2 formate din cel mult 100 de cifre zecimale este relativ mic. Mai exact

$2^x < 10^{100} \iff x < 100 \log2 10$
$\iff x < 333$

Prin urmare, ne interesează doar numerele 2^3, 2^4, …, 2^{332} pe care le putem calcula folosind numere mari (trebuie implementată doar înmulţirea cu 2) şi apoi stoca ca şiruri de caractere. Facem acum observatia ca dacă o putere de 2 se regăseşte ca subsecvenţă a alteia, atunci cea din urmă este redundantă şi poate fi eliminată. Empiric, rămânem astfel cu doar 73 de puteri de 2, de interes.

Să ne imaginăm acum procesul incremental prin care construim un număr oltenesc cifră cu cifră, începând de la cea mai semnificativă. Fie X numărul nostru actual, căruia urmărim să îi mai adăugăm o nouă cifră c la coadă, obţinând astfel un număr de forma Xc. Trebuie întotdeauna să avem grijă ca adăugarea lui c să nu formeze o putere de 2 ca sufix al lui Xc. Pentru a ne asigura de acest lucru, este nevoie să menţinem o informaţie suplimentară, şi anume care este cel mai lung sufix al lui X care este prefix al unei puteri de 2 din cele de interes. Empiric, există 1397 de astfel de prefixe de puteri de 2 (incluzând, bineînţeles, şirul vid), pe care le vom indexa cu numere naturale de la 0 la 1396 şi le vom numi stări. Fără a restrange generalitatea, starea 0 va reprezenta şirul vid.

Pentru fiecare stare q vom încerca toate cifrele c posibile şi vom calcula o stare q' cu proprietatea că oricare ar fi X având cel mai lung sufix care este prefix de putere de 2 reprezentat de q, şirul Xc va avea cel mai lung sufix care este prefix de putere de 2 reprezentat de q'.
In acest caz vom nota $\partial(q,c) = q'$.
De asemenea, putem calcula mulţimea *F* a acelor stări care conţin o putere de 2 ca sufix.
Probabil ca mulţi dintre voi recunosc acum cele de mai sus ca fiind automatul Aho-Corasick. Totuşi, restricţiile permit o implementare neoptimă a constructiei automatului, un algoritm brute-force fiind suficient (vezi sursa oficiala).
In sfarsit, problema noastra se rezolvă prin programare dinamica. Definim

dp[i][q][less] = numărul de moduri de a construi un număr parţial X format din i cifre cu
proprietatea ca dacă less = 0, atunci X = N[0... i-1],
iar dacă less = 1, atunci X < N[0... i-1}. Desigur, mai impunem condiţia ca X să nu conţină nici o putere de 2 ca subsecvenţa:

Aici permitem ca X sa înceapă cu cifra 0 pentru a simplifica implementarea. Răspunsul final este dat de suma tuturor valorilor dinamicii de forma dp[sz][-][-]. Iniţializarea dinamicii este dată de dp[0][-][-] = 0, cu singura exceptie dp[0][0][0] = 1.
Dinamica se va calcula în ordine crescătoare dupa i.
Cel mai simplu va fi sa folosim dp[i][-][-] pentru a calcula dp[i+1][-][-] (tehnica dinamica înainte) şi nu să calculăm dp[i][-][-] pe baza dp[i-1][-][-] (tehnica dinamica inapoi).

Mai exact, pentru (i, q) fixate, vom updata în felul urmator:
Oricare ar fi 0 <=bc <=N[i], adunăm dp[i][q][0] la dp[i + 1][∂(q,c)][c < N[i]];
Oricare ar fi 0<= c<= 9, adunăm dp[i][q][1] la dp[i + 1][∂(q,c)][1].
Pentru a suprima numerele neolteneşti, vom ignora toate tranzitiile unde ∂(q, c) apartin lui *F* .

PROBLEMA 2 CARACATIŢA

Soluţie 100 Puncte

O observație importanta este că toate șirurile date în input se vor retine că o structura vector de char reprezentând caracterul și int numărul de repetiții. (de exemplu, abcccdd se va retine că { (a,1), (b,1), (c,3), (d,2) }.
Folosind aceasta observație verificăm pentru fiecare string din query dacă este subşir în șirurile vrednice și le reținem. (considerăm acest vector W)
Acum avem 2 variante pentru rezolvarea de 100 de puncte. În cazul în care query-ul este subşir, putem verifica brute force (depinzând de optimizările la nivel de interpretare) de câte ori apare restrâns în şirul vrednic restrâns.
Altfel, vom ţine cele N șiruri date și toate sufixele acestora într-un trie. (pentru abbccc vom tine abbccc, bbccc, ccc). Observam că nu este nevoie să ținem decât caracterele distincte (pentru abbccc ținem abc, bc și c) și în fiecare nod din trie un map în care v[i] = numărul de ori de care apare nodul respectiv în şirul i (din șirurile vrednice). Astfel, pentru string-ul s, vom caută în trie, iar când ajungem în nodul corespunzător ultimului caracter din s, ans = suma (map[x]), x = elementele din W (ne asiguram în modul acesta că se respecta ambele reguli).

PROBLEMA 3 DOUBLEDROP

Soluţie 100 Puncte

Principala observație că (*) exista Cn(numărul lui Catalan) n-melodii care k-rup. Deci numărul de n-melodii care k-rup este același pentru oricare k. Așadar ne interesează doar lungimea melodiilor.
Deci un query N M ne cere să calculăm: $\sum_{i=0}^{N} \quad C_i C_{N-i} = C_{N+1}$ (identitate cunoscută).
Așadar, trebuie să precalculam recursiv toate numerele Catalan în O(max(N)).
Apoi se raspunde la query în O(1). Complexitate finala: O(LEN + max(N)).

Demonstrație pentru (*):
Se ia o n-melodie care k-rupe. Fie i cel mai mic indice în care în prefixul corespunzător lui i diferența dintre numărul de V-uri și numărul de P-uri este strict pozitivă. Fie j indicele cel mai mic indice aflat după i pentru care diferența dintre V și P este 0. Atunci se poate obține o n-melodie care (k-1)-rupe prin interschimbarea prefixului (j-1) cu sufixul corespunzător lui (j+1).
Se poate observa că pentru o n-melodie care k-rupe, n-melodia care (k-1)-rupe corespunzătoare este unică. Procesul este reversibil deci am obținut o bijectivitate.
Prin extindere, exista o bijectivitate intre n-melodii care i-rup și n-melodii care j-rup pentru
0 <= i,j <= n. Deci numărul lor este egal. O n-melodie care 0-rupe este echivalenta cu o parantezare corecta. Exista Cn parantezari corecte. Deci exista Cn n-melodii care i-rup pentru orice i.

Secţiunea 11-12

PROBLEMA 4 LANŢURI

Soluție 100 de puncte

Problema se poate rezolva optim cu timp de calcul de ordin n printr-o abordare clasică: dinamică pe arbore.

Dacă rulăm un dfs dintr-un nod oarecare, putem ca pentru nodul curent să obţinem informaţii despre lanţurile care îl conţin şi care mai conţin doar noduri din subarborele a cărui rădăcină este el. Pentru aceasta este necesar să reţinem şi informaţii despre lanţuri care pleacă dintr-un nod în jos (în subarborele cu el rădăcină).

Calculele se fac la revenirea din recursivitate (după determinarea informaţiilor pentru fiii nodului curent).