DESCRIERE SOLUŢII

PROBLEMA 1 **Autor: student Cristian Dospra**
Facultatea de Informatică, Universitatea Bucureşti

GARD

Soluție O(2^K) – 20p

Se generează toate posibilitățile de a alege intervale și verificăm dacă putem obține o acoperire completă. Alegem soluția de cost minim.

Soluție O(N * K) – 50p

Rezolvarea începe prin a sorta intervalele date crescător după capătul drept. Parcurgem apoi cele K intervale construind pe parcurs următoarea dinamică: D[i] = Costul minim pentru a rezolva intervalul [1...i]. Pentru fiecare interval actualizăm dinamica astfel: D[capăt_dreapta] = Min (D[capăt_dreapta], D[capăt_stânga-1] + cost_interval). Trebuie apoi să actualizăm toate valorile din intervalul capăt_stânga... capăt_dreapta cu noua valoare.

Soluție O(K log N) – 100p

Rezolvarea constă în optimizarea soluției precedente. Folosim un arbore de intervale pentru a păstra valorile minime pe dinamica noastră. Când avem nevoie să calculăm minimul pe un interval ne vom folosi de apelul pe acesta, urmând apoi să actualizăm arborele odată cu dinamica. Pentru obținerea punctajului maxim este nevoie de o implementare ce folosește LazyUpdate pentru actualizarea unui interval.

PROBLEMA 2 **Autor: elev Moldoveanu Vlad**
Colegiul Național "Mihai Viteazul", Ploiești

MISSING

Se creează un tabel hash în două dimensiuni, una pentru lungimea cuvântului şi alta pentru suma literelor H[MAXL][MAXL*MAXCH]. Fie un cuvânt de lungime l şi suma literelor s, căutăm un cuvânt potrivit începând cu H[l+1][s] (în cazul în care litera lipsă este 'a') până la H[l+1][s+25] (în cazul în care litera lipsă este 'z').

Complexitate:

- Timp: O(MAXCH*M), însă, în practică, se apropie de O(M);
- Memorie: O(MAXCH*N)

PROBLEMA 3 **Autor: student Vlad Costin**
Facultatea de Informatică, Universitatea Bucureşti

MONKEY

Soluție 30 – 40 de puncte.

Soluția clasică pentru calculul lungimii celui mai lung subșir comun dintre două șiruri va obține doar 30 – 40 de puncte, în funcție de implementare.

Această soluție folosește programarea dinamică și se bazează pe următoarea idee: având șirurile A și B de lungime N, vom calcula matricea D de dimensiune N * N având următoarea semnificație: D[i][j] = lungimea maximă a celui mai lung subșir dintre prefixul de lungime i a șirului A și a prefixului de lungime j a șirului B. Atunci relația de recurență va deveni:

Ploiești, 26 martie 2017

D[i][j] = max (D[i][j-1], D[i-1][j]) dacă A[i] != B[j]
= D[i-1][j-1] + 1, dacă A[i] == B[j]

Acest algoritm are complexitatea timp $O(n^2)$, pe când cea a spațiului este O(n), deoarece avem nevoie doar de ultima linie a matricei pentru a continua calculul.

Soluție 100 de puncte.

Soluția de 100 de puncte ține cont de precizările şi restricțiile problemei noastre:

1. Elementele şirurilor sunt generate aleator în intervalul [1, K], ceea ce înseamnă că putem să presupunem că într-un şir de N numere, fiecare număr de la 1 la K, va apărea, în medie, de [N/K] ori.

2. Restricțiile 1 <= N <= 10000 şi 10 <= K <=1000000
10001 ≤ N ≤ 100000 si 1000 ≤ K ≤ 1000000
100.001 ≤ N ≤ 1.000.000 si 100.000 ≤ K ≤ 1.000.000

În soluția următoare voi presupune că numărul de apariții al oricărui număr este [N/K].

Pentru fiecare poziție din vectorul A, (fie ea i), vom calcula lungimea celui mai lung subșir comun dintre prefixul lui A de lungime i şi B care îl conține pe A[i]. Este evident că pentru A[i], ne interesează doar pozițiile din B care au aceeaşi valoare cu A[i]. Vom folosi un vector auxiliar V, inițial gol. Pentru fiecare poziție i din A, vom parcurge toate pozițiile din B, j astfel încât B[j] == A[i] şi vom căuta binar, în V, cea mai mică valoare, mai mare decât j. Fie p poziția pe care s-a găsit cea mai mică valoare mai mare decât j, atunci V[p] va deveni j. În caz că j este mai mare decât orice valoare din V, atunci îl vom adăuga pe j la V. Astfel, la fiecare pas, p-ul calculat va însemna lungimea celui mai lung subşir comun dintre prefixul lui A de lungime i şi B şi care se termină în B[j], iar lungimea lui V, după acest algoritm, va reprezenta lungimea celui mai lung subşir comun dintre A şi B.

Complexitate timp: O(**),** unde $Ap_A[i]$ = numărul de apariții al lui i în şirul A, analog Ap_B. Ținând cont de faptul că fiecare element apare de [N/K] ori, atunci putem rescrie complexitatea în $O(K*(N/K)^2*\log N) = O(N^2/K*\log N)$, ce se încadrează în limita de timp pentru restricțiile noastre.

Complexitate spațiu : O(N)

Ploieşti, 26 martie 2017