Saptamana 3 - Dinamica Partea 1

Cuprins

Tilling Terrace

Rezumat cerinta

Se da un grid 1 X N format din caracterele . (sol) si # (piatra), cu maxim 50 de #.
Acest grid va trebui acoperit cu mai multe tipuri de *tile-uri*:

- tip 1: Acopera o celula cu "." si valoreaza G1
- tip 2: Acopera 2 celule consecutive cu ".." si valoreaza G2
- tip 3: Acopera 3 celule consecutive cu ".#." si valoreaza G3
  Care este suma cea mai mare a *tile-urilor*
  $N \leq 10^5$

Observatie 1

Maxim 50 de # $=>$ Maxim 50 de $tile$-uri de tip 3.

Observatie 2

Un tile de tip 2 <=> doua tile-uri de tip 1 => Daca $G1 * 2 > G2$, mereu vrem sa inlocuim un tile de tip 2 cu doua de tip 1.

Solutie

Putem calcula pentru fiecare numar de tile-uri de tip 3, care este numarul maxim de tile-uri de tip 2 care pot fi puse. Vom stii in consecinta, cate tile-uri de tip 1 suntem obligati sa folosim si putem sa schimbam mai tarziu orice tile de tip 2 in doua de tip 1 daca este mai optim.

Vom avea $dp[i][k] =$ numarul maxim de tile-uri de tip 2 care pot fi puse, daca am acoperit primele i pozitii si am folosit k tile-uri de tip 3.

$dp[0][0] = 0$
Tranzitiile arata in felul urmator:
$dp[i][k] --+--> dp[i+1][k]$ (punem un tile de tip 1, deci trebuie ca caracterul de la pozitia i + 1 sa fie .)
$dp[i][k] --+--> dp[i+2][k]$ (punem un tile de tip 2, trebuie ca caracterele de la pozitie i + 2 si i + 2 sa fie .)
$dp[i][k] --+--> dp[i+3][k+1]$ (punem un tile de tip 3, trebuie ca doar caracterul de la pozitia i + 1 sa fie #)
Acum putem afla numarul de tile-uri de tip 1 obligatorii daca avem k fixat, acesta va fi $n - 2 * dp[n][k] - k$. De aici, putem transforma greedy, anumite tile-uri de tip 2 in tile-uri de tip 1 ca sa maximizam raspunsul final.

Exista si alte solutii alernative interesante.

Bonus

Rezolvati problema pentru numarul de # ≤ 1000, si $N \leq 10^6$

Jungle Job

Cerinta

Se da un arbore cu **n** noduri. Sa se calculeze pentru fiecare **k** = {1..n} cati subarbori **conexi** de dimensiune **k** ai subarborelui initial exista.

Input

N -> nr. noduri arbore
N - 1 valori $\mathbf{p_i}$ (**i** = {1, N-1}) -> al **i**-lea nod are $\mathbf{p_i}$ drept parinte (nodul **0** e radacina)

Restrictii

N <= 1000

Rezolvare

Fixam radacina arborelui initial ca fiind nodul **0**.

O idee initiala ar fi sa partitionam intr-un fel subarborii de dimensiune k (pentru un **k** din cerinta fixat), in asa fel incat sa ii putem numara mai usor. O astfel de partitionare ar fi urmatoarea: fiecarui subarbore ii asociem nodul cel mai apropiat de radacina (care e unic).

Astfel, problema se reduce la a calcula, pentru fiecare nod **nd** din arborele initial, cati subarbori de dimensiune **k** sunt formati din nodul **nd** si din noduri din subarborele sau (in

sensul arborelui initial).

Sa notam cu **dp[i][k]** = numarul de subarbori care au nodul **i** drept radacina si au dimensiune **k**. Atunci problema se reduce la un rucsac pe care il construim pe baza urmasilor nodului **i** folosind dp-urile asociate lor.

Recurenta pentru un nod **i** si o dimensiune **k** ar arata in felul urmator:

dp[i][k] = $\sum_{k_1+k_2+...+k_t=k-1} dp[x_1][k_1] * dp[x_2][k_2] * \cdots * dp[x_t][k_t]$

Modul in care se impart $k_1, k_2, k_3 \ldots$ de mai sus e echivalent cu partitionarea lui **k - 1** in **t** submultimi, ce are o complexitate exponentiala. Alegerile pe care le facem din dp-urile urmasilor lui **i** sunt independente intre ele, deci putem construi dp[i][k] prin update-uri succesive.

Adaugarea dp-ului unui urmas la rucsacul actual arata in felul urmator (unde *x* e indicele urmasului):
$dp\_temp\_nou[i][k] = \sum_{j=1}^{k} dp\_temp[i][j] * dp[x][k-j]$

Initial, dp_temp[i][1] = 1 (pentru $k \neq 1$ fiind egal cu 0, intrucat suntem fortati sa alegem nodul **i**)

Dupa ce am epuizat toti urmasii, dp-urile asociate lui **i** se calculeaza in felul urmator:
$dp[i][0] = 1$
$dp[i][j] = dp\_temp[i][j] \ \ (j \neq 0)$

Pentru un k din cerinta, valoarea cautata este: $\sum_{i=0}^{n-1} dp[i][k]$

Complexitate

La prima vedere, algoritmul e $O(n^3)$ (costul adaugarii unui urmas la rucsacul actual e $O(n^2)$, sunt $O(n)$ urmasi pentru toate nodurile).

In realitate, daca iteram doar prin starile de dp valide din recurentele de mai sus ($dp[i][j]$ e considerata valida daca $j \leq subsize[i]$, intucat nu putem alege mai multe noduri din subarborele unui nod decat exista), atunci complexitatea devine $O(n^2)$. De ce?

Putem demonstra prin inductie acest fapt. Pasul inductiv ar arata in felul urmator:

Sa presupunem ca ne aflam in situatia in care ne aflam intr-un nod **i** si luam in considerare dp-ul unui urmas al sau, **x**. Daca dimensiunea maxima a rucsacului (dp_temp) de pana acum este s_1 si dimensiunea subarborelui lui **x** este s_2, atunci complexitatea totala a obtinerii noului rucsac (dp_temp_nou) va fi
$O({s_1}^2 + {s_2}^2 + s_1 * s_2) = O({s_1}^2 + {s_2}^2 + 2 * s_1 * s_2) = O(\ (s_1 + s_2)^2)$.

Glass Half Spilled

Cerinta

Se dau **n** pahare cu apa. Fiecare pahar are o capacitate maxima, din care o parte e deja ocupata de un volum de apa. Putem turna dintr-un pahar un volum **v** de apa in alt pahar, dar jumatate din volum se va pierde in timpul transferului (asta daca paharul din care turnam are un volum de apa in el mai mare sau egal cu v, iar cel in care turnam are o capacitate ramasa de cel putin v/2).
Pentru fiecare $k \in \{1..n\}$, sa se calculeze volumul maxim de apa care se poate obtine intr-o submultime a paharelor daca putem turna apa din unele in altele ca mai sus.

Input

N -> nr pahare
N perechi (c_i, v_i) -> (capacitate_maxima, volum actual) ($c_i \geq v_i$)

Restrictii

N, c_i, v_i <= 100

Rezolvare

Pentru un **k** ales, are sens sa fixam de la inceput cele **k** pahare ce vor fi alese la final. Astfel, problema s-ar reduce la a alege o submultime de **k** pahare din sirul initial.

Daca gandim problema din perspectiva alegerilor pe care le-am face in cadrul unui algoritm de backtracking pentru generarea submultimilor, observam ca nu ne intereseaza intreaga stiva a alegerilor anterioare, ci doar numarul de pahare finale alese pana acum, capacitatea paharelor alese si volumul maxim de apa utilizabil in urma alegerilor (paharele alese contribuie cu volumul lor intreg de apa, cele care nu au fost alese contribuie cu jumatate din volumul lor).

Prin urmare, recurenta reiese natural:

$dp[i][j][t]$ = volumul maxim de apa utilizabil daca am parcurs **i** pahare, am ales **j** pahare si am o capacitate totala a paharelor alese de **t**

$dp[i][j][t] = max(dp[i-1][j-1][t-c_i] + v_i,\ \ dp[i-1][j][t] + \frac{v_i}{2})$

Pentru un **k** fixat, raspunsul este $\max\limits_{i=0}^{sum\_cap}(dp[n][i][k])$.

Complexitatea finala este $O(\sum\limits_{i=1}^{n} c_i\ *\ n^2)$. ($\approx\ 100^4$)

LEGO Bricks

Rezumat cerinta

Se dau N piese de LEGO mov 2x2 intr-un spatiu 3D. A i-a piesa are coordonatele $(x_i, 0, h)$, $x_i \leq x_{i+1}$.

Restrictii:

$N \leq 300, 0 \leq h, x_i \leq 10^9$

Observatie 1

Structura va semana cu cea a unei paduri. In total ar exista n lanturi, una pentru fiecare piesa de LEGO si se va intampla ca doua lanturi sa divearga la un anumit moment de maxim n-1 ori.

Solutie

Consideram $dp[l][r] =$ numarul minim de piese ca sa sustinem piesele mov cu indexii de la l la r.
Pentru $l = r, dp[l][r] = h[l]$,
altfel avem $dp[l][r] = min(dp[l][m] + dp[m+1][r] - cost(h, x_l, x_r) | l \leq m \leq r)$, unde $cost(h, xl, xr) = max(0, h + 1 - \lceil (xr - xr)/2 \rceil)$, reprezentand cate piese am economisi daca am combina arborii pentru piesele din $[l, m]$ cu cele din $[m+1, r]$ intr-un singur arbore (incercam sa combinam lanturile piesei l si a piesei r cat mai repede)

Adunare2

Rezumat cerinta

Ni se dau 3 randuri de cifre, fiecare cu N cifre. Exista Q actualizari a unei cifre de pe un anumi rand si coloana. Vrem sa aflam inainte de toate actualizare si dupa fiecare, in cate moduri se pot sterge anumite coloane, astfel incat daca numerele care s-ar forma pe primele 2 randuri, ar da insumate, numarul de pe cel de al treilea rand.

Restrictii

$1 \leq N, Q \leq 10^5$

Observatie

Vom considera numarul de la dreapta la stanga, in ordinea normala de adunare. Daca pe o anumita coloana, pe primul rand numarul este x, pe randul al doilea y, respectiv z pe randul

al 3-lea atunci exista 2 cazuri bune:

- (x + y) % 10 = z si nu avem niciun rest venit de la adunarea cifrelor din dreapta (daca suma lor a fost peste 9)
- (x + y + 1) % 10 = z si avem rest 1 venit de la adunarea cifrelor din dreapta.

Astfel, tot ce ne pasa cand ajungem pe o anumita coloana, este daca caram un rest de 1 sau nu.

Solutie

O prima solutie ar fi sa calculam urmatoarea dinamica:
$dp[col][carry = 0/1] =$ numarul de moduri de a sterge coloanele (pana in coloana col), astfel incat adunarea sa fie corecta si cifra de trasport ramasa pentru urmatoarele coloane sa fie egala cu carry
Aceasta solutie ar avea complexitate timp de $O(N * Q)$ recalculand dinamica dupa fiecare modificare.

Observam ca ordinea operatiilor nu este relevanta daca am stii pentru fiecare operatie daca ajungem sau nu la ea cu $carry = 1$.

Astfel, am putea combina raspunsul a oricare doua intervale de coloane => operatia devine asociativa => putem folosi un arbore de interval in care tinem
$dp[carryBefore = 0/1][carryAfter = 0/1]$ pentru fiecare nod care reprezinta un interval pe care vrem sa aflam cate dintre submultimile de coloane sunt bune daca consideram ca venim cu carryBefore si terminam lasand carryAfter ca rest pentru operatiile care urmeaza.

Se poate observa ca putem combina dinamica fiilor daca carryAfter al fiului drept este egal cu carryBefore al fiului stang (dupa operatiile facute din intervalul din dreapta, lasam fix restul de care are nevoie intervalul din stanga ca adunarile sa dea ce trebuie).

Raspunsul se va afla in radacina si va fi $dp[0][0]$ (incepem fara niciun rest si trebuie sa terminam fara niciunul). La o actualizare a unei cifre trebuie sa actualizam frunza care reprezinta coloana pentru schimbare.

Complexitatea finala este $O((N + Q) * log(N))$ sau $O(N + Q * log(N))$ in functie de implementare.