Cursul 4 - Convex Hull Trick

Frog 3

Cerinta

Avem un rau cu **N** pietre numerotate cu indici de la **1** la **N**. Fiecare piatra are o inaltime h_i. Avem, de asemenea, o broasca care se afla initial pe piatra cu numarul **1** si care se poate deplasa prin salturi de pe o piatra la alta. Daca aceasta se afla pe piatra **i**, atunci poate sari pe o piatra **j** (**j** $>$ **i**), saltul avand efortul asociat egal cu $(h_j - h_i)^2 + C$. (unde C e in input) Sa se calculeze efortul minim pe care trebuie sa il depuna broasca pentru a ajunge de pe piatra **1** pe piatra **N**.

atcoder.jp/contests/dp/tasks/dp_z

Input

N -> nr de pietre
C -> constanta din formula
h_i -> inaltime pietre

Restrictii

$2 \leq N \leq 2 \times 10^5$
$1 \leq C \leq 10^{12}$
$1 \leq h_1 < h_2 < \cdots < h_N \leq 10^6$

Rezolvare

O recurenta in $O(n^2)$ e destul de usor de observat:

dp[i] = $\min_{j<i}(dp[j] + (h_i - h_j)^2 + C)$

Problema e ca nu e suficient de rapida, asa ca, probabil, va trebui sa acceleram tranzitiile folosind o structura de date.

Incercam sa rescriem recurenta in asa fel incat sa separam informatiile ce tin de starea anterioara de informatiile ce tin de starea in care ajungem:

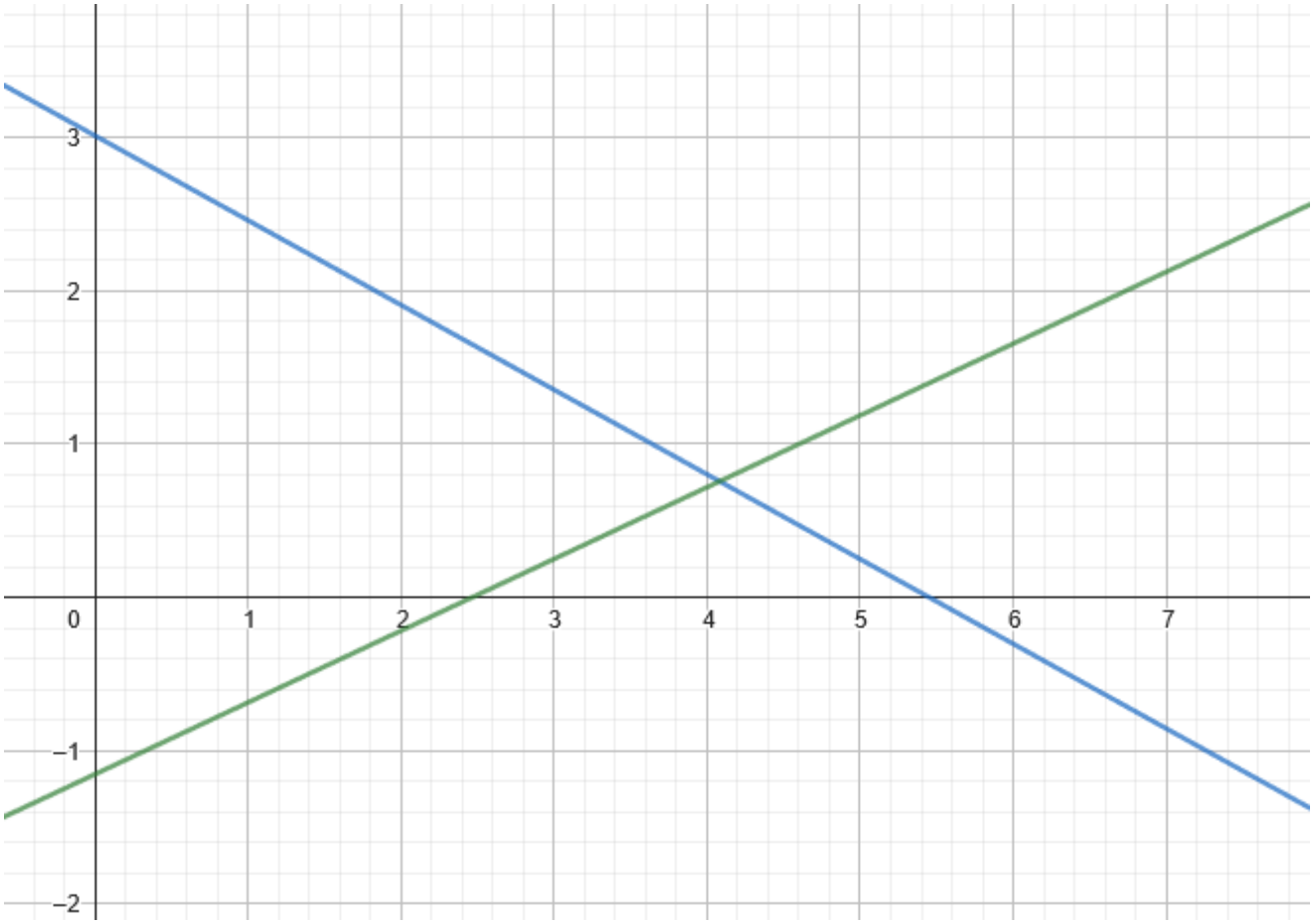
dp[i] = $\min_{j<i}(dp[j] + {h_j}^2 - 2 * h_j * h_i + {h_i}^2 + C)$ = $\min_{j<i}((dp[j] + {h_j}^2) + (-2 * h_j) * h_i + ({h_i}^2 + C))$

Prin urmare, singurele proprietati ale pietrelor anterioare care influenteaza costul saltului sunt:

$b_j = dp[j] + {h_j}^2$
$m_j = -2 * h_j$

Putem observa ca cele 2 proprietati descriu, practic, o dreapta, printr-o panta si o distanta fata de origine. Astfel, problema se rezuma la a determina cel mai jos punct al intersectiei dintre o axa verticala (data de h_i) si o multime de drepte (date prin m_j si b_j).

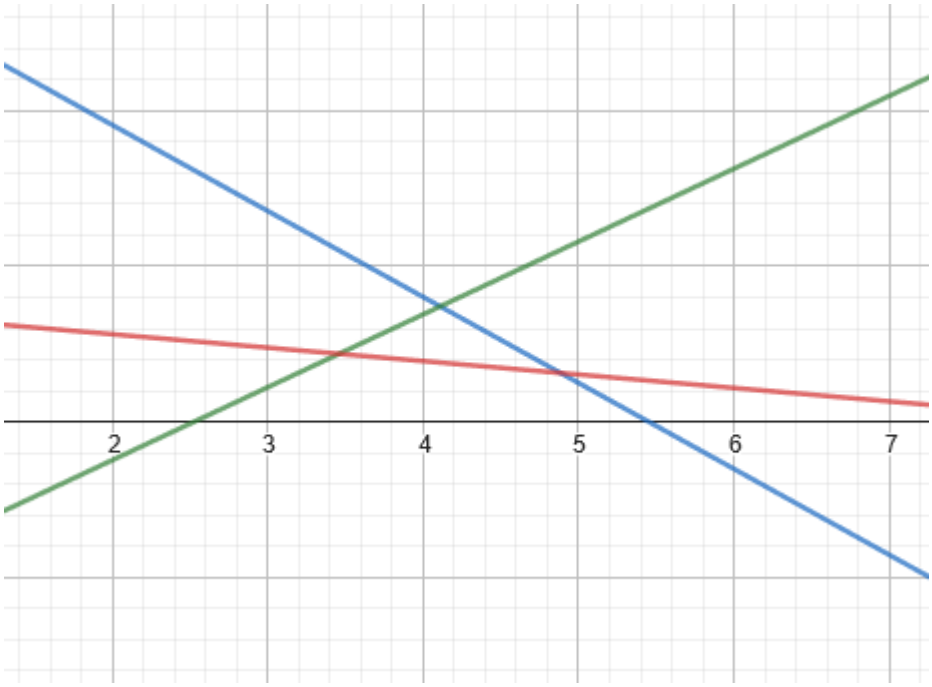
Sa presupunem ca avem 2 drepte cu pante diferite. Care dreapta ofera valoarea minima si pentru ce interval de axe verticale?



Observam ca valoarea minima este data de dreapta cu panta mai mare (colorata cu verde) pana la coordonata X a punctului de intersectie al celor 2 drepte, iar de la intersectie la $+\infty$ valoarea minima e data de dreapta colorata cu albastru (cu panta mai mica).

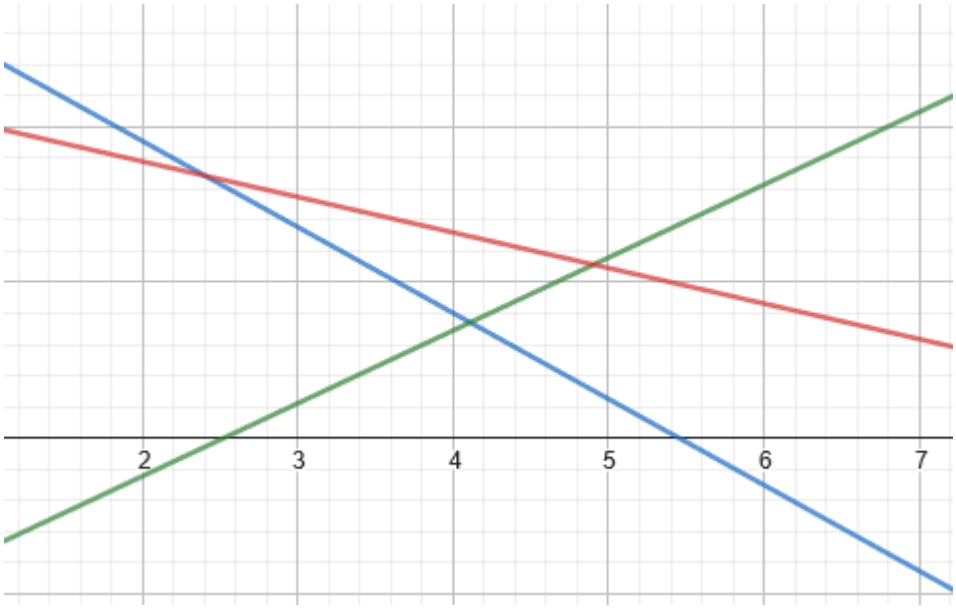
Astfel, se observa ca fiecarei drepte ii revine o sectiune din multimea axelor verticale, in ordinea pantelor lor.
Ce se intampla daca avem 3 drepte?

Putem avea o situatie precum aceasta:



In care dreapta verde da minimul pana la intersectia cu dreapta rosie, dreapta rosie da minimul de la intersectia cu cea verde pana la intersectia cu cea albastra, iar cea albastra da minimul de la intersectia cu dreapta rosie pana la $+\infty$.

Dar putem avea situatii si precum urmatoarea:



In care linia rosie nu da niciodata valoarea minima.

Astfel, atunci cand adaugam o dreapta noua la structura de date, trebuie sa verificam daca are sens sa se regaseasca in acea structura de date (daca da minimul pentru un interval de axe verticale sau daca e eclipsata de celelalte drepte). De asemenea, trebuie sa verificam daca, prin prezenta ei, invalideaza alte drepte aflate deja in structura de date (adica ofera minime mai bune decat anumite drepte din structura pe tot intervalul de axe asociat acelor drepte).

Prima verificare nu e necesara, intrucat stim ca, parcurgand pietrele de la stanga la dreapta, inaltimile sunt strict crescatoare, deci pantele vor fi strict descrescatoare, iar ultimele drepte adaugate vor oferi intotdeauna valori mai mici (cand axele verticale se apropie de $+\infty$).

A doua verificare se rezuma la a vedea, pentru 3 drepte consecutive (**i**, **i + 1**, **i + 2**) in sirul de drepte sortat descrescator dupa panta, unde se regasesc relativ unul la celalalt intersectiile dreptelor (**i**, **i + 2**) si (**i**, **i + 1**). Daca dreapta **i + 2** se intersecteaza cu dreapta **i** la dreapta intersectiei sale cu dreapta **i + 1**, atunci dreapta **i + 1** se afla deasupra celor doua si nu poate contribui cu minime. Avand in vedere ca dreptele sunt introduse in structura in ordinea descrescatoare a pantelor, verificarea a doua se poate realiza cu o stiva.

Astfel, data fiind o axa verticala, cautarea punctului minim se poate face prin cautare binara (pentru ca intervalele de axe verticale ce apartin fiecarei drepte sunt in ordinea pantelor).

Complexitate finala: $O(n\ log(n))$

Harbingers

Cerinta

Se da un arbore cu **N** noduri. Radacina arborelui este nodul 1. Muchiile care conecteaza nodurile au lungimi exprimate in kilometri. In fiecare nod se regaseste cate un mesager, care are un timp de pregatire si o viteza (exprimata in minute / kilometru). In cazul in care un nod este atacat, atunci mesagerul nodului atacat va porni cu un mesaj spre radacina arborelui (alegand drumul de lungime minima). Ajuns fiind intr-un alt nod, acesta poate pasa mesajul mesagerului din nodul respectiv. Sa se calculeze, pentru fiecare nod, timpul minim necesar pentru ca mesajul sa ajunga la radacina in cazul in care nodul este atacat.

www.infoarena.ro/problema/harbingers

Idei

O recurenta in $O(n^2)$ e destul de usor de observat:

dp[i] = $\min_{j \ stramos \ strict \ pt. \ i}(dp[j] + (dist\_to\_root[i] - dist\_to\_root[j]) * v[i] + prep[i])$

Aceasta poate fi rescrisa in felul urmator:

dp[i] = $prep[i] + dist\_to\_root[i] * v[i] + \min_{j \ stramos \ strict \ pt. \ i}(dp[j] + (-dist\_to\_root[j]) * v[i])$

Asemanator ca la problema anterioara:

$m = -dist\_to\_root[j]$
$b = dp[j]$

Coborand pe un drum din radacina spre un nod al carei valoare vrem sa o calculam, observam ca $dist\_to\_root$ creste (intrucat ne departam de radacina), prin urmare $-dist\_to\_root$ descreste. Ne aflam in aceeasi situatie cu problema anterioara, in care dreptele vin in ordine descrescatoare a pantelor si vrem sa calculam minimul.

Diferenta fata de problema anterioara consta in faptul ca trebuie sa stergem drepte din containerul de linii (intrucat, intr-un dfs clasic, trebuie sa ne intoarcem dintr-un urmas al unui nod pentru a calcula dp-urile pentru ceilalti urmasi), ceea ce e destul de dificil. Garantiile de complexitate la inserare de la problema precedenta nu mai tin (acestea fiind $O(1)$ amortizat - o dreapta poate fi introdusa si stearsa o singura data).

Astfel, ar avea sens sa salvam, intr-un fel sau altul, sirul de drepte active si sa revenim la el cand ne intoarcem din recursivitate. Nu are sens sa salvam container-ul in fiecare nod si sa ne intoarcem la el (intrucat complexitatea ar fi $O(n^2)$ daca arborele ar fi un lant).

Solutia, in acest caz, pleaca de la urmatoarea observatie: in loc sa mentinem un container pentru intregul drum de la radacina pana la nodul actual, putem pastra mai multe containere

pentru bucati din drum. Astfel, valoarea unui query va fi data de minimul valorilor obtinute pentru fiecare container.

Pentru a mentine un numar minim de astfel de containere, putem aplica un rationament asemanator cu cel de la *heavy-light-decomposition*, in care, pentru fiecare nod, consideram drept urmas greu acel urmas care are subordinea sa cel putin jumatate din nodurile ce se afla in subordinea parintelui sau ($subSize[urmas] \geq subsize[parinte] \ / \ 2$).

Astfel, containerul asociat bucatii de drum care include nodul actual va fi extins prin urmasul greu al nodului actual. Atunci cand ajungem de la un nod la un urmas usor al sau vom crea un nou container (intrucat consideram ca incepe un bucata noua de drum de la urmasul usor). Numarul de containere pe care trebuie sa facem interogarea este $O(log(n))$ (de fiecare data cand intalnim un container nou - coborand dinspre radacina spre nodul pentru care calculam dp-ul - stim ca intram intr-un urmas usor, cu subSize-ul cel mult jumatate din cel al parintelui; injumatatirea se poate face doar de log_2 ori).

O observatie legata de implementarea dfs-ului e urmatoarea: atunci cand apelam recursiv calculul pentru urmasii unui nod, trebuie sa lasam urmasul greu (daca exista) la final, altfel riscam sa extindem containerul asociat bucatii de drum a nodului cu drepte care nu afecteaza un urmas usor al acestuia (si obtinem niste valori (posibil) mai mici decat ar trebui sa fie).

Prin urmare, complexitatea finala e $O(n * log_2(n)^2)$.
($n\_noduri * log\_containere\_per\_drum * cautare\_binara$)

Realizatori:
Apostol Ilie-Daniel
Popescu Stefan-Alexandru