Lotul Național de Informatică 2024, Cluj-Napoca
Seniori, Proba 1 de baraj
Descrierea Soluțiilor

Comisia Științifică

12 mai 2024

Problema 1: Triunghi

Putem descompune un triunghi în două triunghiuri disjuncte mai mici cu latură egală și un pătrat. Astfel, pentru un triunghi de latură de mărime K avem două cazuri:

- K este par, caz în care triunghiul poate fi descompus în două triunghiuri de latură $K/2$ și un pătrat de latură $K/2$;
- K este impar, caz în care triunghiul poate fi descompus în două triunghiuri de latură $(K-1)/2$ și un pătrat $(K+1)/2$; putem descompune mai departe acest pătrat într-unul de mărime $(K-1)/2$, o linie de mărime $(K-1)/2$, o coloană de mărime $(K-1)/2$ și un singur element.

Din descompunerea de mai sus, trebuie să vedem cum putem precalcula sumele și pe pătrate, linii și coloane. Similar, avem cazuri de paritate pentru linii și coloane. Liniile și coloanele se calculează ușor (se împarte în două jumătăți, eventual un element izolat ce trebuie adăugat artificial). Pentru pătrate:

- K este par, caz în care pătratul se calculează din patru pătrate de latură $K/2$;
- K este impar, caz în care pătratul se calculează din patru pătrate de latură $(K-1)/2$, o linie de latură $(K-1)/2$, o coloană de latură $(K-1)/2$ și un element izolat.

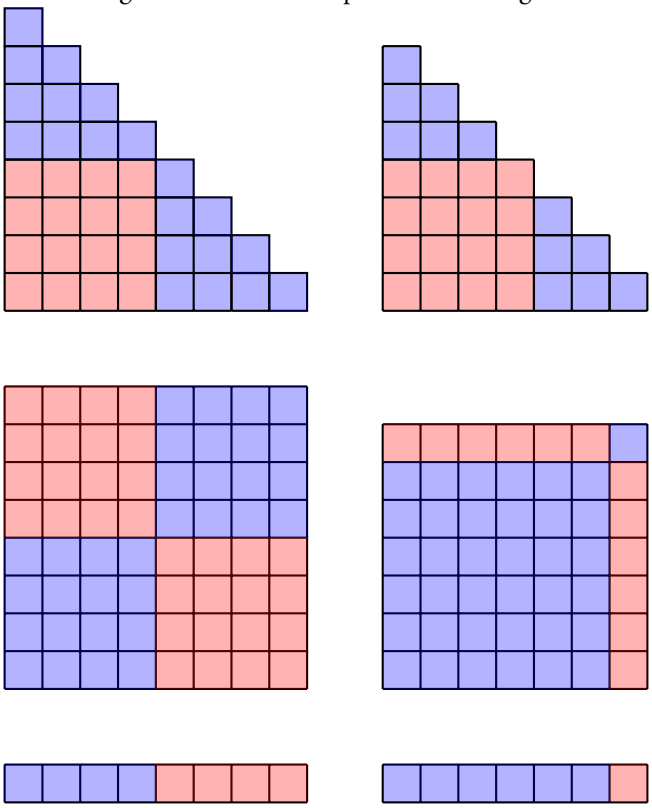
Vedem cazurile descrise în Figura 1.

A se observa că mereu obiectele de mărime K se vor calcula din obiecte mai mici de aceeași mărime, lucru ce simplifică implementarea. Este important să reținem doar nivelul anterior de mărime, nu pe toate, deoarece nu dispunem de $O(N^2 \log N)$ memorie.

Cum K se înjumătățește de fiecare dată, avem $\log K$ iterații, deci soluția va merge în $O(N^2 \log N)$ ce ia scor maxim.

Există și o soluție în $O(N^2)$ pe care o lăsăm ca temă cititorului.

Figura 1: Cazurile din problema Triunghi



Problema 2: Matrice Randomizată

În primul rând, colțurile trebuie să fie egale. După aceea, putem scoate colțurile, și într-un fel revenim la aceeași problemă: ne aflăm în niște poziții, acestea trebuie să fie egale, excludem elementele dacă sunt egale ș.a.m.d.

Asta ne duce la următoarea soluție cu programare dinamică: $\text{dp}[l][c][l'][c'] = \text{true}$ dacă există un drum palindromic de la poziția (l, c) până la (l', c'). Dacă cele două poziții sunt egale, dintr-o stare (l, c, l', c') ne ducem în stările $(l + 1, c, l' - 1, c')$, $(l + 1, c, l', c' - 1)$, $(l, c + 1, l' - 1, c')$ și $(l, c + 1, l', c' - 1)$.

Dinamica implementată direct cum e explicat mai sus o să iasă în complexitatea $O(N^4)$. Totuși, (l, c) și (l', c') trebuie să fie la distanțe egale de colțuri, deci numărul de stări relevante este de fapt $O(N^3)$.

Soluțiile de mai sus încă nu au exploatat faptul că matricea este generată aleator.

Una dintre soluții ar fi să ne restrângem spațiul de căutare la celulele la distanță cel mult 20 de diagonala *principală*. Intuitiv, această soluție merge deoarece se formează foarte multe șiruri care sunt distincte, iar atunci când cele două capete se întâlnesc la mijloc, există o șansă mare ca acestea să aiba un punct comun care să parcurgă aceleași șiruri.

Problema 3: Doi Arbori

Problema ne spune că ni se dă un arbore în care unele frunze sunt considerate active. Trebuie să răspundem la întrebări de genul (u, v): care este lungimea unui traseu minim de la nodul u la una dintre frunzele active + distanța de la respectiva frunză activă până la nodul v. Trebuie să suportăm update-uri de forma u (unde u e frunză): activează / dezactivează frunza u. Inițial, toate frunzele sunt dezactivate.

Soluție în $N \log^2 N$ O observație este faptul că la un query, cele două noduri vor merge pe lanțul dintre ele, iar de pe unul din nodurile acelea, vor continua să meargă pe frunza cea mai puțin depărtată de lanț.

Faptul că avem de calculat minime pe lanțuri ne duce la ideea de a băga Heavy Path Decomposition. Așadar, ne vom descompune arborele în lanțuri.

Dacă încercăm să menținem aceste distanțe în mod direct, putem să luăm punctele pe subtaskul 6.

Inițial voiam să calculăm care este cea mai apropiată frunză activată l_i care se află în subarborele lui i. Vom schimba puțin această definiție. Astfel, în l_i menținem cea mai apropiată frunză de nodul i din subarborele lui i, dar din acest subarbore, vom exclude subarborele cu muchia grea din decompoziție. Practic, noi menținem cele mai apropiate frunze pentru toate lanțurile 'light' care se toarnă în acel 'heavy'.

Atunci când o frunză se activează sau dezactivează, ne plimbăm din lanț în lanț și vom modifica doar valorile în care se toarnă aceste lanțuri, adică $O(\log N)$ valori. Pentru fiecare nod, pentru a calcula l_i, vom menține un set sau un heap cu frunzele activate pentru a face ușor activările sau dezactivările în $O(\log^2 N)$.

Pentru a face ușor minimele pe lanțuri dintre l_i, menținem un arbore de intervale pe decompoziție. A se observa că update-urile sunt în valori, nu în intervale complete. Așadar, un query se poate face în $O(\log^2 N)$.

Mai avem niște cazuri de tratat și anume:

- trebuie să tratăm subarborele heavy într-un fel; pentru fiecare lanț pe care îl parcurgem într-un query, nu vom da query doar pe un prefix de-al lui, ci pe tot lanțul; cum orice lanț din decompoziție are exact o frunză în capete, pe aceea o calculăm direct;
- dacă frunza nu se află în subarborele pe care se află tot lanțul, noi trebuie să ieșim din LCA și să mergem în continuare în sus; acest caz se tratează în mod similar, doar că trebuie să adăugăm/scădem adâncimea fiecărui nod în mod artificial.

Ca și recapitulare, descompunem arborele în lanțuri, pentru fiecare nod menținem un set cu cele mai apropiate frunze care vin din lanțuri 'light'. Menținem mai mulți arbori de intervale pe minime, unul cu $l_i - d_i$, unul cu $l_i + d_i$ și unul cu l_i. La un query, mergem din lanț în lanț, dăm query pe prefixul și pe sufixul fiecărui lanț folosind un arbore de intervale, iar odată ce trecem de LCA, trebuie să avem grijă din ce arbore de intervale dăm query. Complexitatea totală iese $O(Q \log^2 N)$.

Soluție în $N\sqrt{N}$ Vom nota cu F mulțimea frunzelor din arbore. De asemenea, vom nota cu $\text{dist}(u, v)$ distanța de la nodul u la nodul v. Pentru o întrebare (u, v), răspunsul este $\text{dist}(u, v) + \min(\text{dist}(w, f))$, unde w se află pe lanțul de la (u, v) și $f \in F, f$ activ.

Vom împărți cele Q operații în blocuri a câte $\sqrt{Q}$ operații consecutive. Acum, vrem să rezolvăm un bloc de operații. Vom nota cu A mulțimea nodurilor active pe toată durata blocului curent (în alte cuvinte, frunzele care au fost active înainte începerii blocului curent și care nu au fost modificate deloc pe parcursul lui).

Pentru un nod u, vom nota cu closest_u distanța până la cea mai apropiată frunză activă. Atunci când vrem să procesăm un bloc, vom vrea să recalculăm closest. Acest lucru se poate face ușor cu o parcurgere în lățime din toate nodurile $u \in A$.

Pentru un query (u, v) vrem să determinăm $\min(\text{closest}_w)$ cu w pe lanțul de la u la v. De vreme ce procesăm operațiile în calupuri a câte $\sqrt{Q}$ vom vrea să avem o structură care face operații de update foarte rapide, chiar dacă query-ul se face în timp cel mult $\sqrt{Q}$. Putem să facem asta cu o descompunere în radical pe stiva de noduri de la rădăcină la fiecare nod.

Ne lipsesc ca și candidați doar frunzele care nu sunt în A, dar au fost active la un moment dat în timpul blocului procesat. Totuși, vom avea doar $\sqrt{Q}$ astfel de noduri, prin care putem să iterăm și să calculăm $\text{dist}(u, f) + \text{dist}(f, v)$ (unde f este un astfel de nod).

Pentru că avem foarte multe operații de tip $\text{dist}(u, v)$ vom avea nevoie de o precalculare a LCA-ului în $O(N \log(N))$ pentru a răspunde în $O(1)$ la query.

Această soluție are complexitatea timp $O((N + Q)\sqrt{Q})$ și cea spațiu $O(N \log(N))$,