Lotul Național de Informatică 2024, Orăștie
Seniori, Proba 3 de baraj
Descrierea Soluțiilor

Comisia Științifică

26 Mai 2024

Problema 1: Gurger Brill

Soluție în $4\sqrt{n}$ interogări

Ca și mai sus, dublăm toate interogările. Împărțim persoanele în $R = \sqrt{N}$ blocuri de cîte R persoane. Cu alte cuvinte, le grupăm după cîtul împărțirii la R. Facem $2R$ interogări, rotind permutarea cu R elemente după fiecare pereche de interogări. Analizînd rezultatele, putem afla, pentru fiecare persoană, blocurile în care se află cei doi vecini ai săi.

Acum, regrupăm persoanele după restul împărțirii la R și facem 2R interogări similare. Astfel aflăm, pentru fiecare persoană, cîturile și resturile vecinilor la împărțirea prin R.

De exemplu, pentru $n = 100$ de persoane numerotate de la 0 la 99, după primele 20 de interogări aflăm cifrele zecilor pentru cei doi vecini ai fiecărei persoane, iar după încă 20 de interogări aflăm cifrele unităților.

Ca și la soluția anterioară, încă nu avem permutarea completă. Dacă vecinii lui X au cifrele zecilor 1 și 2, iar cifrele unităților 3 și 4, atunci vecinii lui X ar putea fi 13 și 24 sau 14 și 23. Putem folosi aceleași remedii ca mai sus.

Soluție în $4 \log n$ interogări

Pentru fiecare bit semnificativ b, unde $0 \leq b < \lceil \log n \rceil$, vom folosi 4 întrebări pentru a afla informații despre bitul b al ambilor vecini ai tuturor persoanelor.

Ca regulă generală, pentru fiecare interogare P facem și o interogare pentru permutarea răsturnată. Astfel, pentru o persoană X iau naștere trei cazuri:

- X ajunge devreme în prima interogare: ambii vecini ai lui X se află la dreapta lui X în P.
- X ajunge devreme în a doua interogare: ambii vecini ai lui X se află la stînga lui X în P.
- X ajunge tîrziu în ambele interogări: X are cîte un vecin de fiecare parte în P.

Pentru fiecare bit b, partiționăm numerele după bitul b și facem două interogări. Apoi răsturnăm cele două intervale și facem încă două interogări. Iată un exemplu pentru $n = 16$ și bitul 0. Am numerotat persoanele de la 0 la $n - 1$ pentru claritate.

```
P = 0 2 4 6 8 10 12 14   1 3 5 7 9 11 13 15   (și răsturnata)
Q = 14 12 10 8 6 4 2 0   15 13 11 9 7 5 3 1   (și răsturnata)
```

Analizînd cazurile posibile putem deduce informații despre bitul b în toți vecinii. Iată, de exemplu, posibilitățile pentru $X = 6$ în funcție de numărul de vecini în stînga pentru interogările de mai sus:

răspuns la P	răspuns la Q	paritatea vecinilor
0	0	doi vecini impari
0	1	un vecin par și unul impar
0	2	doi vecini pari
1	0	un vecin par și unul impar
1	1	doi vecini pari
2	0	doi vecini pari

La final nu aflăm vecinii exacți, ci distribuția biților în cei doi vecini (adică dacă pe fiecare poziție avem 0, 1 sau 2 biți de 1). De exemplu, dacă pe toate pozițiile avem cîte un 1 și un 0, vecinii pot fi 0 și 15 sau 3 și 12 sau 7 și 8 etc. De aceea, poate fi nevoie să alegem o persoană de start A și să îi testăm, pe rînd, toți vecinii posibili B care respectă configurațiile pe biți. Odată fixat B, putem calcula restul mesei în $O(n)$, căci cunoscînd A și B, C este impus, apoi din B și C, D este impus, etc.

Pe teste aleatorii, acest algoritm găsește soluția. Ea poate fi pedepsită cu teste adversariale în care toate persoanele au mulți biți variabili între cei doi vecini. Atunci algoritmul găsește, cel mai probabil, o soluție greșită. Există două remedii:

- Identificarea deterministă a unui vecin al lui A, cu încă $\log n$ interogări (de exemplu, o căutare binară în care A variază, iar restul permutării rămîne nemodificat).
- Indirectarea tuturor întrebărilor și răspunsurilor printr-o permutare aleasă aleatoriu.

Soluție în $12 + 3 \log n$ interogări

Prima parte a soluției se reduce la a partiționa persoanele în 3 mulțimi independente (din care cel mult una dintre mulțimi ar putea fi vidă), unde o mulțime de persoane este independentă dacă nu există 2 persoane vecine din ordinea secretă care se află în aceasta.

Pentru a extrage cele 3 mulțimi independente de persoane, se consideră mai întâi un procedeu de a extrage o mulțime independentă maximală de persoane, care va fi prima dintre acestea. Mulțimea independentă maximală se definește ca o mulțime independentă la care nu se mai pot adăuga alte elemente astfel încât să-și păstreze proprietatea de a fi independentă.

Pentru a extrage mulțimea independentă maximală putem considera o nouă partiționare a mulțimii inițiale obținută prin interogări succesive (prin interogare se înțelege operația de interogare descrisă în enunț). Este evident că o interogare returnează o mulțime independentă (asta reiese ușor din faptul că odată ce se adaugă o persoană la rezultatul interogării, vecinii săi sigur nu mai pot fi adăugați pentru că persoana adăugată este un vecin al lor care apare deja în interogare). Astfel, se pot trimite interogări până când toate persoanele au fost partiționate într-o mulțime independentă (aici trebuie ordonate elementele cu atenție, pentru că odată ce unele elemente au fost deja partiționate, acestea trebuie trimise în interogări la finalul permutării din interogare și să nu fie adăugate la noile mulțimi independente chiar dacă sunt returnate în răspunsul interogării). Pentru a realiza aceasta fază a problemei în cât mai puține operații, se pot trimite persoanele relevante pentru o interogare în ordine aleatorie. Această parte de randomizare este necesară pentru că, intuitiv, s-ar vrea ca fiecare interogare să aibă răspunsul cât mai lung. Dacă se consideră că fiecare persoană care vine la întâlnire primește un număr de ordine egal cu poziția sa în interogare, răspunsul interogării devine numărul de valori minime locale (mai mici decât ambii vecini) din așezarea secretă de la masă, care este de așteptat să fie mai mare pe o ordine aleatorie (un exemplu de situație în care numărul de minime locale ar fi mic dacă nu se randomizează ordinea ar fi permutarea identică). După un număr de 1000 de rulări ale acestei metode, s-a ajuns în cel mai rău caz la 6 operații necesare.

După ce se obține partiționarea descrisă anterior, se poate afla mulțimea independentă maximală "combinând" toate mulțimile independente obținute anterior prin interogări succesive (după fiecare interogare se păstrează rezultatul acesteia și aici trebuie din nou ca elementele "irelevante" pentru scopul interogării curente să fie plasate la finalul acesteia). Combinările de mulțimi se fac succesiv ("două câte două"), deoarece algoritmul descris funcționează doar pentru a combina 2 mulțimi independente și a obține mulțimea independentă maximală din reuniunea acestora (cu mențiunea că la o interogare elementele celor două mulțimi nu trebuie intercalate când sunt afișate), dar nu funcționează pe un caz general cu N mulțimi. Astfel, această parte consumă încă $6 - 1 = 5$ operații în cel mai rău caz, deci, în total, aflarea mulțimii independente maximale consumă cel mult $6 + 5 = 11$ operații.

Odată ce se află mulțimea maximală independentă, dacă considerăm așezarea secretă de la masă, nu va exista o secvență continuă formată din mai mult de două persoane din afara acestei mulțimi, deoarece din 3 sau mai multe persoane adiacente, cel puțin una s-ar putea adăuga la mulțimea independentă, ceea ce ar contrazice proprietatea acesteia de a fi maximală. Având în vedere această observație, se pot afla celelalte două mulțimi independente dintr-o singură interogare, care are ca persoane relevante persoanele care nu apar în mulțimea independentă maximală obținută anterior: prima dintre mulțimi va fi formată din persoanele rezultate din interogare, iar a doua din restul de persoane care nu apar în primele două mulțimi obținute (a doua mulțime ar poate fi și vidă). În total, partiționarea persoanelor în 3 mulțimi independente consumă $6 + 5 + 1 = 12$ interogări.

Având acum această partiționare în 3 mulțimi independente la îndemână se poate începe căutarea vecinilor pentru fiecare persoană. Se consideră o procedură în care avem la îndemână 2 mulțimi independente notate cu A, respectiv B și care returnează pentru toate persoanele din mulțimea A vecinii lor din mulțimea B (dacă există). Să considerăm că pentru o singură persoană din mulțimea A vrem să-i găsim vecinul din B care se află pe cea mai mică poziție în B (simetric se poate afla și cel care se află pe cea mai mare poziție). Acest vecin se poate căuta binar folosind operația din enunț astfel: dacă la iterația curentă din căutarea binară vrem să vedem dacă vecinul persoanei din A se află după primele X persoane din B (unde X este fixat de căutarea binară), se face o interogare în care primele X persoane sunt primele X persoane din B, persoana cu numărul $X + 1$ este persoana fixată din A, iar restul șirului din interogare poate fi completat în mod aleator. Dacă răspunsul la interogare conține elementul curent din A înseamnă că vecinul căutat pentru persoana din A nu se află printre primele X, iar în caz contrar vecinul se află în primele X.

Dacă ar fi să se caute binar pentru fiecare persoană din mulțimea A vecinul dorit, s-ar folosi $O(n \log n)$ interogări, dar acest pas se poate optimiza prin a căuta binar în paralel vecinii pentru toate persoanele din mulțimea A, folosind astfel doar $O(\log n)$ interogări. Această optimizare funcționează pentru că se pot combina într-o singură interogare toate interogările care s-ar fi făcut individual pentru fiecare element din mulțimea A.

Având procedura de căutare a unui vecin definită anterior, se notează cu A, B, respectiv C partiția persoanelor formată din cele 3 mulțimi independente, iar cu XY reuniunea mulțimilor X și Y, unde X și Y sunt disjuncte și pot fi A, B sau C. Pentru a găsi vecinii tuturor celor n persoane, se poate exploata faptul că fiecare persoană are exact 2 vecini care se află într-o mulțime diferită de a sa (mulțimile fiind independente), deci se poate folosi procedura de aflare a vecinilor prin căutare binară în paralel pentru perechea de mulțimi (A, BC) de două ori pentru a afla vecinul cu numărul mai mic, respectiv mai mare pentru toate persoanele din mulțimea A (în mod evident se află și pentru persoanele din mulțimile B și C unicul vecin din mulțimea A (dacă acesta există), respectiv pentru perechea de mulțimi (B, C) pentru a afla vecinul din mulțimea C (dacă acesta există) pentru toate elementele din mulțimea B (și invers). Această modalitate de a afla vecinii folosește cel mult $3 \log n$ interogări. Astfel, pentru a rezolva întreaga problemă se folosesc cel mult $12 + 3 \log n$ interogări, 12 pentru a afla cele 3 mulțimi independente în care se partiționează persoanele și cel mult $3 \log n$ pentru a afla vecinii persoanelor, utilizând partiția calculată la început.

Problema 2: LaNaic

Soluție în $O(n^2)$

Menționăm o soluție ineficientă, dar ușor de codat și care face o observație teoretică importantă. Pornim cîte un DFS din fiecare nod și, pentru nodul descendent curent, reținem numărul de modificări necesare pentru a putea ajunge la el. Este nevoie să reținem lungimea sufixului de muchii de aceeași culoare din stiva DFS curentă. Oricînd acest contor depășește K, incrementăm numărul de modificări și resetăm contorul.

Observăm că numărul de modificări necesare pentru a sparge un lanț de lungime L este întotdeauna $\lfloor \frac{L}{K+1} \rfloor$.

Soluție în $O(n \log n)$

Să considerăm o descompunere în centroizi a arborelui. Pentru centroidul curent x, ne punem întrebarea: care este lanțul K-alternant maxim care trece prin x? Dacă rezolvăm acest pas în $O(S(x))$, unde $S(x)$ este mărimea subarborelui lui x, atunci complexitatea globală va fi $O(n \log n)$.

Construim o pereche de vectori C_0 și C_1, unde $C_i(j)$ este **costul minim** necesar pentru a construi un lanț K-alternant care pornește din x pe o muchie de valoare i, are lungime j și se termină undeva în subarborele lui x. Indicii j iau valori între 0 și $H(x)$, unde $H(x)$ este înălțimea în muchii a subarborelui lui x.

Definim **costul minim** al unui astfel de lanț ca fiind o pereche: numărul minim de modificări necesare, iar, în caz de egalitate, lungimea minimă a prefixului de muchii de valoare i cu care începe lanțul. Raționamentul din spatele acestei definiții este că lanțul optim prin x va consta din două lanțuri (sau, în cazul degenerat, un singur lanț) care pornesc pe doi fii diferiți. Dacă cele două lanțuri pornesc pe prefixe de aceeași valoare, și dacă suma lungimilor acestor prefixe depășește K, este necesară o modificare suplimentară, altfel nu. Nu avem niciun motiv să preferăm un lanț care face mai multe modificări, dar are prefix mai scurt. Putem amîna acele modificări pînă cînd și dacă devin necesare. Mai remarcăm și că vectorii C sînt monotoni: costul crește cu adîncimea.

Pentru fiecare fiu y putem calcula un astfel de vector, C_y. Există unul singur, deoarece muchia (x, y) are fie valoarea 0, fie 1. Este necesară o parcurgere DFS. O implementare rezonabilă este: lansăm o primă parcurgere pe muchii de aceeași valoare cu muchia (x, y), al cărei scop este să determine lungimea prefixului de aceeași valoare. Din ea comutăm la o a doua parcurgere DFS cînd întîlnim prima muchie de valoare opusă.

Pe măsură ce fiii lui x își încheie parcurgerile, x are de făcut două operații: (1) să caute lanțul maxim între fiul curent y și oricare din fiii anteriori și (2) să adauge informația din C_y la C_0 sau C_1. Deoarece vectorii sînt monotoni, putem face pasul (1) în $O(H_{max} + H(y))$, unde H_{max} este adîncimea maximă a oricărui fiu anterior. Folosim tehnica *two pointers* pentru a căuta suma maximă, dar nu mai mult de M, a două elemente din doi vectori sortați. Pasul (2) este trivial, constînd doar din compararea celor doi vectori poziție cu poziție.

Pentru complexitate optimă, este important să procesăm fiii lui x în ordinea crescătoare a adîncimii. Altfel H_{max} poate crește rapid, de exemplu dacă explorăm fiul cel mai adînc primul. După sortarea fiilor, efortul total pentru x va fi $O(S(x))$.

Problema 3: Siclam

Fără pierdere de generalitate, presupunem că T se află în stânga lui R. Dacă rezolvăm acest caz, putem după să răsturnăm vectorul și indicii de query și să răspundem în același mod.

Intuitiv, ambii jucători vor să ocupe cât mai mult teren. Astfel, dacă sunt foarte depărtați, ei se vor mișca cât se poate de mult unul către celălalt până când se întâlnesc, după care cucerec toate celelalte poziții pe care le mai pot cuceri.

Când cei doi sunt apropiați, avem mai multe cazuri. În aproape toate cazurile, strategia lor este aceeași. Singurul caz în care strategia este ușor diferită este când aceștia încep de pe poziții adiacente.

T poate să se miște în stânga, caz în care acesta va lua un prefix, iar la sfârșit se va întoarce să mai ia poziția din dreapta lui R, iar R va lua un sufix. În celălalt caz, T se mișcă la dreapta. Astfel, R are de luat decizia dacă ia un sufix sau un prefix. Pentru T, această mișcare nu are sens decât dacă ar câștiga luând sufixul, dar asta ar însemna că R nu joacă optim.

În acest punct, strategia devine irelevantă și știm că T va lua un prefix și R va lua un sufix.

Definim o poziție x ca fiind o frontieră dacă x este cea mai din stânga poziție pe care o ocupă R și $x + 1$ este cea mai din dreapta poziție pe care o ocupă T. Ne propunem să numărăm pentru fiecare poziție de frontieră numărul de configurații inițiale.

Din strategia de mai sus, cum cei doi se îndreaptă unul spre celălalt, distanța dintre ei va scădea cu 5, deci are sens să ne gândim la clase de resturi modulo 5. Când analizăm jocurile cu distanțe mai mici sau egale cu 5, vom obține cazuri diferite. Putem lua aceste cazuri pe foaie, să vedem diferențele de poziții și să le automatizăm în mod similar cu vectorii de direcție (simplifică implementarea). Astfel, pentru fiecare frontieră, obținem toate perechile de poziții de distanță mai mică sau egală cu 5. Aceste perechi le vom numi cazuri de bază, de la care se vor construi toate celelalte cazuri cu distanță mai mare.

Astfel, dacă (i, j) este un caz de bază și T câștigă, atunci și $(i - 2, j + 3)$ este o poziție câștigătoare pentru T. Generalizat, dacă (i, j) este un caz de bază, atunci și poziția $(i - 2x, j + 3x)$ este o poziție câștigătoare pentru orice x (atâta timp cât pozițiile nu ies din șirul inițial.

O altă observație este faptul că dacă începem jocul cu 6 poziții mai la dreapta (sau la stânga), acesta va avea exact același comportament, doar că T va lua un prefix mai lung (respectiv mai scurt) și R va avea un sufix mai scurt (respectiv mai lung). Astfel, avem o relație de monotonie pe clasele de resturi modulo 6.

Modificăm puțin definiția unui caz de bază. O configurație inițială (i, j) este un caz de bază, dacă distanța dintre ele este mai mică sau egală cu 5, iar dacă mutăm jocul cu 6 poziții la stânga, se schimbă câștigătorul. Astfel, pozițiile câștigătoare se pot genera de la cazurile de bază cu următoarea formulă: dacă (i, j) este un caz de bază, atunci toate perechile $(i - 2x + 6y, j + 3x + 6y)$ sunt perechi câștigătoare cu oricare x și y. Cu această definiție nouă, în loc de $5N$ cazuri de bază, avem acuma $5 \cdot 6$ cazuri de bază.

Putem fixa fiecare x posibil și să vedem pentru câte y-uri avem soluții. Ne scriem condițiile de existența pentru fiecare caz ($i - 2x + 6y \geq l$ și $j + 3x + 6y \leq r$), aflăm valoarea maximă a lui y în funcție de x. După ce obținem această formulă de forma $\min \ldots, \ldots$, observăm că cele două ramuri ale funcției au o relație de monotonie. Putem calcula punctul în care se schimbă ramura funcției.

După ce desfacem ramurile în funcție de x, putem regrupa termenii și să folosim din nou o formulă (suma lui Gauss).

Complexitatea acestei soluții este $O(Q \cdot 30)$.