Lotul Național de Informatică 2024, Cluj-Napoca
Seniori, Proba 2 de baraj
Descrierea Soluțiilor

Comisia Științifică

14 mai 2024

Problema 1: As de Spade

Vom nota cu min_len numărul de cifre ale lui A și cu max_len numărul de cifre ale lui B.

Pentru început, este mai ușor să considerăm cazul în care numărul de cifre al lui A este egal cu numărul de cifre al lui B. În acest caz, e ușor de observat că trebuie doar să ordonăm numerele în ordine descrescătoare. Pentru cazul în care numărul de cifre al lui A este diferit de numărul de cifre ale lui B, putem să considerăm două numere x și y. Singurul caz non-trivial este acela când x este prefix al lui y (sau invers). În acest caz, fie l numărul de cifre al lui x. În acest caz, diferența o face prima cifră (în ordine ciclică în y) și prima cifră (în ordine ciclică în x, începând cu poziția $l + 1$). Astfel, obținem faptul că avem o ordine totală a numerelor, cu condiția de ordonare pentru două numere (x și y) fiind dată de relația $xy > yx$ (unde ab reprezintă concatenarea numărului a cu numărul b).

Așadar, știind că numerele de aceeași lungime apar în șirul final în ordine descrescătoare, putem determina poziția unui număr x. Pentru fiecare lungime l (de la min_len la max_len) vom căuta binar câte numere de l cifre apar în șir înainte lui x, căutând binar prin fiecare clasă de numere (unde numerele dintr-o clasă au aceeași lungime).

Acum, pentru fiecare lungime l, vom căuta cel mai din dreapta număr x care nu începe la o poziție mai mare decât k. Cu această informație putem găsi ușor cea de a k-a cifră.

Această soluție are complexitate $O(L^2 \log(B - A)^2)$, unde $L = max\_len - min\_len$.

Putem acum să optimizăm soluția de mai sus. Pentru început, vom încerca să optimizăm modul în care calculăm pe ce poziție se află numărul x în șir. În continuare, pentru fiecare lungime l trebuie să găsim cel mai din dreapta număr care apare înaintea lui x în șir. Observația este că aceste numere sunt formate din concatenări ale lui x, la care se poate adăuga un prefix al lui x, cu precizarea că ultima cifră poate să fie incrementată cu 1. Așa putem calcula în $O(L)$ toate y-urile pentru un x fixat.

Rămâne să optimizăm pasul în care determinăm calupul de numere care sunt apropiate de poziția k. Momentan, pentru fiecare lungime fixată l căutăm cel mai din dreapta număr de l cifre care începe înainte de poziția $k + 1$. Observăm că aceste numere au prefixe comune foarte lungi. Așadar, vom căuta binar acel număr pentru cea mai mare lungime (care nu se comportă diferit din cauza constrângerii că numărul trebuie să fie mai mic sau egal cu B). Pentru a determina restul candidaților, este suficient să scoatem în mod succesiv câte o cifră de la finalul lui acestui număr și să verificăm dacă începe înainte de poziția $k + 1$.

Astfel, se obține complexitate $O(L^2 + L \log(B - A))$ per query.

Soluție alternativă (căutare cifră cu cifră)

Să pornim de la un exemplu, un subșir al lui G pentru $A = 1$, $B = 999\,999$:

număr	poziție în G
252526	4890865
252525	4890871
2525	4890877
25	4890881
252524	4890883
252523	4890889
25252	4890895
252522	4890900
252521	4890906

Observăm că printre numerele de lungime maximă apar prefixe mai scurte. Concret, definim o **zonă** ca fiind un număr împreună cu toate prefixele numărului care sunt adiacente cu acesta în G. De exemplu, zona lui 252525 include prefixele 2525 și 25 și are lungime totală 12. În schimb, prefixul 25252 nu este în această zonă, ci în zona lui 252522. Zona lui 252521 conține doar numărul însuși, având lungime 6.

Putem deduce că un prefix se află în zona numărului de lungime maximă obținut prin completarea ciclică a prefixului până la lungimea maximă. De exemplu, 5287 face parte din zona lui 528752.

Cu această informație, pentru o interogare K putem afla zona de la poziția K. Completăm rezultatul cifră cu cifră. Întâi ne întrebăm: care este lungimea totală a zonelor care încep cu 9? Dar cu 8? Dar cu 7? Scădem aceste valori din K cât timp se poate. Dacă am stabilit, de exemplu, că prima cifră a zonei este 5, ne întrebăm: care este lungimea totală a zonelor care încep cu 59? Dar cu 58? Dar cu 57? Etc.

Fie $L(P, N)$ lungimea zonelor care încep cu prefixul P și nu depășesc valoarea N. Atunci răspunsul la întrebările de mai sus va fi $L(P, B) - L(P, A-1)$, pentru prefixe P succesiv mai lungi. Pentru a-l afla pe $L(P, N)$ este nevoie de:

- Suma lungimilor numerelor care încep cu P, au între $|P|$ și $|N|$ cifre și nu depășesc N. Aceasta necesită doar aritmetică naivă.
- Suma lungimilor prefixelor lui P care sunt în zona lui P. Putem face acest lucru în $O(|P|)$ dacă răspundem în $O(1)$ la întrebarea: este Q în zona lui P? Vom da un exemplu aici: $Q = 523$ (3 cifre) este în zona lui $P = 5235235235$ (10 cifre) deoarece, prin repetare ciclică a lui Q până la 10 cifre, îl obținem exact pe P. Echivalent, primele $10 - 3 = 7$ cifre ale lui P coincid cu ultimele 7 cifre, iar acest test îl putem face în $O(1)$ dacă avem precalculate puterile lui 10.

Ultimul pas necesar, odată găsită zona, este să izolăm cifra dorită din zonă. Am abordat naiv acest pas, sortând toate numerele din zonă. Este dificil să anticipăm ordinea numerelor din zonă. De exemplu, pentru numărul 894848948, prefixul 89484894 îl precedă în G, dar prefixul 89484 vine după el.

Complexitatea totală este $O(RD^2)$ per interogare, unde R este baza de numerație (10), iar D este lungimea maximă (18).

Problema 2: 6 de Pentagrame

Cel mai natural pas în această problemă este să folosim o tehnică de tip baleiere. Cum problema ne cere în enunț să afișăm răspunsul pentru fiecare x posibil, vom face baleierea de la stânga la dreapta.

Pentru un x fixat, vom vedea cum se modifică răspunsul în funcție de y-ul pe care îl alegem. Dacă începem cu cel mai mic y posibil, e clar că diferența maximală dintre cadrane va fi foarte mare, iar dacă mărim valoarea lui y, acea diferență maximală o să scadă. Mai exact, atâta timp cât cadranul minim din cele patru este sub y și cadranul maxim este deasupra, când mutăm y la $y + 1$, diferența maximală o să scadă (maximul o să scadă și minimul o să crească). Analog, dacă începem de la cel mai mare y posibil și îl tot scădem, diferența o să descrească, deoarece valoarea minimă a cadranelor o să crească și cea maximă o să scadă.

La mijloc, vom avea un interval de y-uri în care cadranele minime, respectiv maxime se vor afla pe aceeași parte (ambele fie deasupra lui y, fie sub y). Fie acest interval $[y_1, y_2]$. Răspunsul va fi fie dezechilibrul calculat in $y_1 - 1$, fie dezechilibrul calculat în $y_2 + 1$ sau dezechilibrul dat de un y din acest interval. Pentru primii doi candidați vom rezolva separat.

Atunci când minimul și maximul sunt de aceeași parte față de y, vom analiza doar unul dintre cazuri, celelalte trei rezolvându-se identic. Să presupunem că valoarea cadranului de jos-stânga (cadranul C din figură, $< x$, $< y$) este minimă și valoarea cadranului de jos-dreapta (cadranul D din figură, $> x$, $< y$) este maximă.

Atunci dacă asociem fiecărui punct din stânga (orice punct cu prima coordonată $< x$) valoarea lui pozitivă, și fiecărui punct din dreapta (orice punct cu coordonata $> x$) valoarea lui negativă, răspunsul va fi

$$\min_{y=y_1}^{y_2} S_{y_1,y}$$

unde am notat cu $S_{y_1,y}$ suma valorilor punctelor cu coordonate între y_1 și y.

Acest lucru ne duce la ideea de a menține un arbore de intervale cu următoarele informații:

- Suma valorilor din stânga ($< x$) și suma valorilor din dreapta ($>x$).
- Minimul din suma valorilor din stânga minus suma valorilor din dreapta pe orice prefix (analog dreapta minus stânga). Să le notăm cu U și V.
- Minimul din suma valorilor din stânga minus suma valorilor din dreapta pe orice sufix (analog dreapta minus stânga), să le notăm cu W și T.

Soluția actuală are complexitate $O(N \log N)$ și ar trebui să obțină punctajul maxim însă este destul de dificil de implementat. Se pot face însă câteva observații:

1. W și T nu trebuie menținute, ele se deduc din suma valorilor totale pe fiecare parte și U și V.
2. Pentru intervalul $[y1, y2]$ nu trebuie tratate separat cele patru cazuri pentru care e cadranul de valoare maximă și care e cadranul de valoare minimă. Se pot calcula toate cele 4 valori minime (pentru fiecare caz), iar răspunsul va fi mereu valoarea maximă dintre acestea.

Problema 3: Roata Norocului

Cazul cu un singur query

Observăm că mereu este optim să facem la final unirea nodului cu valoare minimă de celelalte noduri. Astfel, putem implementa o soluție care extrage mereu nodul minim din arbore și rezolvă recursiv pe toți subarborii rămași.

O observație suplimentară care ne duce la o soluție în $O(NlogN)$ este aceea că o strategie echivalentă cu a lua nodul minim ultima oară este să luăm muchia cu valorile cât mai mari în ambele capete prima oară.

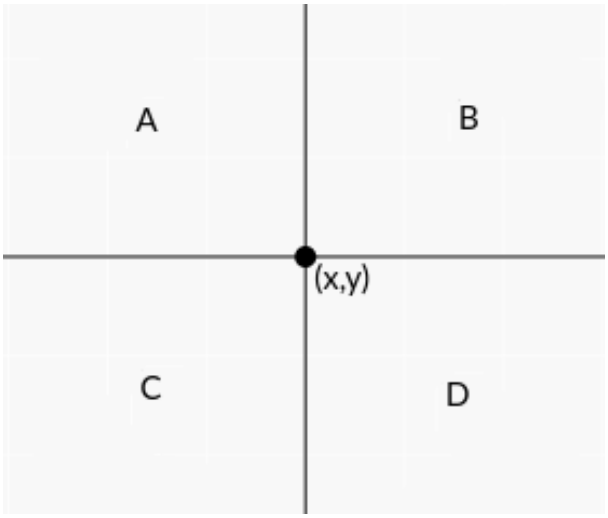

Figura 1: Cadranele impartite

Cazul cu lanț

Extragerea minimului și împărțirea în mai multe componente are exact aceeași structură ca un arbore cartezian. Deci prin construcția arborelui cartezian putem simula prima strategie.

Cazul fără update-uri

Aici există mai multe soluții bune, unele bazându-se pe arborele cartezian al arborelui, dar doar o soluție ne ajută și pe cazul cu update-uri.

Analizăm fiecare nod din arbore. Dacă un nod x nu are valoarea minimă din arbore, atunci exact o dată va fi unit cu un nod cu o valoare mai mică decât a lui, deci va avea o contribuție pozitivă ($+a_x$). Întrebarea este de câte ori va avea un nod o contribuție negativă ($-a_x$)? Răspunsul este chiar numărul de vecini cu o valoare mai mare decât a nodului.

De ce? Există două cazuri, fie subarborele nodului delimitat de vecin are doar valori mai mari decât nodul x, caz în care clar nodul x va avea o contribuție negativă, fie subarborele nodului delimitat de vecin are și valori mai mici decât nodul x. Să notăm nodul vecin cu y și un nod din subarborele vecin cu o valoare mai mică decât a_x cu z. Avem $a_z < a_x < a_y$. Observăm că operația de unire a lui y cu z are același cost cu unirea lui z cu x și apoi unirea nodului rezultat cu y ($a_z - a_y = a_z - a_x + a_x - a_y$), deci, mereu putem să îl inserăm pe x într-o operație de unire, fără vreun cost adițional.

Cazul cu update-uri

Trebuie să simulăm rapid calculul sumei contribuțiilor unui subarbore. Pentru asta putem să ne folosim de `Arbori de Intervale` sau `AIB-uri` (folosind liniarizare în ambele cazuri), și de `Arbori de Intervale` pentru determinarea valorii minime dintr-un subarbore (este cel mai simplu să presupunem că toate nodurile au contribuție pozitivă și să extragem minimul la final).

Pentru a modifica rapid contribuția unui nod atunci când avem un update, avem nevoie de o structură de date care să calculeze rapid numărul de noduri vecine cu o valoare mai mică decât acel nod, ceea ce putem face cu mai multe structuri de date: `Arbori de Intervale` implementați dinamic, `Trie` pe biți, `AIB` dinamic, orice `BST` cu o complexitate bună.

În principiu, o implementare bună, duce la o complexitate de $O((N+Q)log(N))$