MateInfoUB Informatică
Runda 2
Descrierea Soluțiilor

1 Sort

Merită să ne gândim la o abordare care construiește șirul B incremental, adăugând câte un element la final. În acest caz ne întrebăm în primul rând ce valoare ar trebui să alegem pentru $B[1]$. Este natural să alegem cea mai mică valoare posibilă, fiindcă această decizie ne oferă mai multă libertate în alegerea lui $B[2]$.

Acest argument se generalizează ușor pentru a obține soluția completă: pentru fiecare i, alegem cel mai mic $B[i]$ care este permutare circulară a lui $A[i]$ și satisface $B[i] \geq B[i-1]$.

Pentru a analiza complexitatea de timp, notăm că în general un număr de C cifre are cel mult C permutări circulare, iar timpul necesar pentru a itera peste toate este proporțional cu C (dacă generăm următoarea permutare în timp constant) sau C^2 (dacă generăm următoarea permutare în timp $O(C)$). În cazul nostru $C <= 7$, deci atât soluțiile în $O(n \times C)$, respectiv $O(n \times C^2)$ rulează suficient de repede pentru a obține punctaj maxim.

2 Kscale

Subtask-ul ($N = 1$)

Adesea, în cazul problemelor care implică o matrice sau alte structuri multidimensionale, este util să analizăm cazurile de dimensiune mai mică: în acest caz, o matrice 1-D.

La problema de față putem testa efectiv toate variantele de scalare cu $k_1 = 1; k_2 \in 1, 2, ..M$. Pentru arie minimă trebuie să găsim k_2 maxim. Pentru a testa dacă un k_2 este valid, putem împărți matricea unidimensională în intervale compacte de lungime k_2 ce nu se suprapun (k_2 trebuie să fie divizor al lui M). Dacă toate intervalele compacte astfel formate conțin un singur tip de caracter ('0' sau '1'), atunci există un A asociat cu valoarea k_2 care să reprezinte o soluție validă pentru funcția *scale*. Reținem și construim matricea soluție, A, folosind valoarea maximă k_2 ce a fost validată.

Subtask-ul ($k_1 = k_2$)

Similar cu subtask-ul 1-D, trebuie să verificăm toate variantele $k_1 = k_2 = x$, cu $x \in 1, 2..min(N, M)$. Complexitatea de timp este $\mathcal{O}(N^3)$.

Subtask-ul ($1 \leq N, M \leq 100$)

O abordare similară se poate aplica dacă fixăm separat k_1 și k_2.

Complexitatea timp $\mathcal{O}(N^4)$ se reduce la $\mathcal{O}\left(N^2\left(\sqrt{N}\right)^2\right)$ dacă luăm în considerare că putem fixa k_1, k_2 doar printre divizorii lui N, respectiv M.

Subtask-ul ($1 \leq N, M \leq 1000$)

Există cel puțin două abordări care pot obține punctaj maxim.

Prima variantă este să rămânem pe ideea de a fixa toate valorile fezabile pentru k_1, k_2, dar să optimizăm faza de validare a perechii, observând faptul că putem verifica dacă un anumit dreptunghi conține doar valori de 0 sau doar valori de 1 în timp constant, construind sumele parțiale ale matricei. Astfel, pentru o pereche k_1, k_2, complexitatea de verificare se schimbă din $\mathcal{O}(N * M)$ în $\mathcal{O}(N * M/(k_1 * k_2))$, iar suma rezultantă este suficient de mică pentru ca soluția să se încadreze în timp.

A doua variantă este să nu mai fixăm valori pentru k_1, k_2, ci să încercăm să le deducem din structura matricei. Să revenim la cazul 1-D pentru a dobândi niște intuiție.

În cazul 1-D trebuie să găsim k maxim pentru care există A astfel încât $scale(A, 1, k) = B$. Observăm că este necesar și suficient ca orice interval maximal de caractere egale să aibă lungimea multiplu de k. Pentru a găsi k optim putem, deci, aplica *cmmdc* pe toate lungimile de intervale maximale cu același caracter.

Pentru cazul general, $N, M \leq 1000$, putem aplica aceeași abordare de tip *cmmdc*, doar că independent pe intervale maximale orizontale și intervale maximale verticale.

Astfel, pentru cazul: $\mathrm{B} = \begin{matrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{matrix}$

k_2 va fi egal cu *cmmdc* dintre lungimile celor 8 intervale maximale 1-D orizontale:

$k_2 = cmmdc(4, 2, 4, 2, 2, 4, 2, 4) \rightarrow k_2 = 2$

și k_1 va fi calculat prin *cmmdc* dintre lungimile celor 10 intervale maximale 1D verticale:

$k_1 = cmmdc(2, 2, 2, 2, 4, 4, 2, 2, 2, 2) = 2$.

Pentru a demonstra corectitudinea soluției, trebuie să arătăm două lucruri:

- Perechea (k_1, k_2) este validă. Putem imparti matricea B in submatrice disjuncte de dimensiune (k_1, k_2). Daca luăm în considerare că orice lungime de interval maximal 1D orizontal este multiplu de k_2 si orice lungime de interval maximal 1D vertical este multiplu de k_1, atunci avem certitudinea că orice submatrice formata anterior contine un singur tip de caracter '0' sau '1'.
- Perechea (k_1, k_2) este optimă. Să presupunem prin absurd că există o pereche (k_1', k_2') cu $k_1' * k_2' > k_1 * k_2$. Atunci $k_1' > k_1$ sau $k_2' > k_2$. Putem analiza primul caz fără a restrânge generalitatea. Știm că există un interval vertical care nu este multiplu de k_1' (altfel *cmmdc*-ul calculat nu ar fi de fapt maxim). Putem arăta că acel interval invalidează soluția (k_1', k_2').

Această soluție poate fi implementată în complexitate $O(N * M)$

3 Secretariat

Vom face câteva observații generale, după care vom dezvolta soluția pe structura subtaskurilor.

1. Pentru a muta o persoană de la o coadă este necesar să mutăm toate persoanele din fața ei.
2. În soluția optimă fiecare persoană va fi mutată cel mult o dată: din coada la care se află inițial la una dintre cozile asociate în final problemei pe care încearcă să o rezolve. Notați că aceste două cozi pot coincide.

Putem astfel să reformulăm problema din minimizarea numărului de persoane mutate în maximizarea numărului de persoane care nu sunt mutate în timpul operațiilor. Să numim o astfel de persoană *fixată*.

În ce condiții poate o persoană să rămână *fixată*? Să luăm un exemplu de coadă:

Ghișeul X: 3 2 4 2 2 2

Observăm că dacă asociem ghișeului X o altă problemă decât 2, ultima persoană este nevoită să plece, deci vom fi nevoiți să mutăm întreaga coadă (datorită Observației 1). Dacă în schimb îi asociem problema 2, putem fixa ultimele 3 persoane de la coadă (dar nu mai mult, fiindcă persoana cu problema 4 trebuie să plece, iar toate persoanele din fața ei trebuie să facă la fel).

Să notăm cu **suffixLen[i]** lungimea sufixului maximal de persoane cu aceeași problemă pentru coada cu numărul i, respectiv cu **suffixProb[i]** indicele acestei probleme. În general, dacă asociem problema **SuffixProb[i]** ghiseului i, vom fixa **SuffixLen[i]** persoane la acest ghiseu. Altfel, vom fixa 0 persoane.

În acest punct putem calcula usor, dată fiind o asociere între ghișee și probleme, numărul maxim de persoane care pot fi fixate. Fie **AssignedProb[i]** problema asociată în final ghișeului i. Numărul total de persoane fixate este atunci suma valorilor **SuffixLen[i]** pentru toți i care respectă **AssignedProb[i]** = **SuffixProb[i]**.

Ne rămâne problema găsirii unui **AssignedProb[]** optim, pe care o vom analiza pentru fiecare subtask.

Subtask-ul ($N = K = 2$)

În acest caz există doar două variante de a asocia probleme ghișeelor (1 2 și 2 1), putem pur și simplu să le încercăm pe ambele și să o alegem pe cea mai bună.

Subtask-ul ($N = K \leq 1\,000$)

În acest caz există $N!$ asocieri, deci nu avem timp să le încercăm pe toate. Totuși, știm că pentru fiecare problemă va exista exact un ghișeu asociat, deci putem încerca să analizăm independent alegerea corectă pentru fiecare problemă. Este tentant, de exemplu, să asociem fiecărei probleme i un ghișeu j cu **SuffixProb[j]** = i si **SuffixLen[j]** maxim posibil. Observăm că unele probleme nu vor găsi deloc un astfel de ghișeu. Vom numi aceste probleme *simple*, fiindcă ele nu pot contribui la totalul de persoane fixate. Din acest motiv, problemele *simple* au prioritate scăzută și putem să ne imaginăm că le vom asocia arbitrar ghișeele rămase după ce prioritizăm problemele care pot fixa persoane.

Ne conduc aceste alegeri la o soluție optimă global? Din fericire, da. Deoarece un ghișeu poate contribui cu un număr de persoane fixate pentru o singură problemă, știm că două probleme diferite nu vor concura pe același ghișeu decât în cazul în care ambele sunt *simple*, caz în care sunt irelevante.

Subtask-ul ($K \leq N \leq 1\,000$)

În subtask-ul cel mai general soluția optimă poate asocia mai multe ghișee aceleiași probleme. Este natural să încercăm să generalizăm soluția precedentă.

În acest sens, putem încerca să hotărâm problema fiecărui ghișeu iterând peste ghișee în ordine descrescătoare a valorilor **SuffixLen[]**. Ideal, am dori să asociem cât mai des ghișeul i problemei **SuffixProb[i]**. Care sunt situațiile în care nu putem face acest lucru?

Singura situație este cea în care prin asocierea problemei **SuffixProb[i]** la ghișeul i am face imposibil ca problemele rămase încă fără niciun ghișeu să mai găsească vreunul. Dar această situație poate fi verificată ușor: în timpul iterării putem păstra un contor pentru ghișeele rămase libere, respectiv pentru problemele care nu au primit încă măcar un ghișeu. Inițial ambele contoare sunt egale cu N. Apoi la fiecare pas putem calcula cum se modifică aceste contoare dacă asociem ghișeului i problema **SuffixProb[i]**, respectiv dacă îl lăsăm liber. La final, mai facem o trecere și asociem problemele rămase ghișeelor lăsate libere.

Dacă notăm cu M numărul total de persoane, complexitatea de timp a soluției este $O(M + sort(N))$ unde $sort(N)$ este complexitatea sortării folosite pentru N elemente. Având $N \leq 1\,000$, o sortare de complexitate $O(N^2)$ ar fi intrat în timp fără probleme.

4 Grile

Vom enumera câteva strategii posibile, menționând că acestea pot fi puternic influențate de diverse optimizări.

- Cea mai simplă strategie este să trimitem pe rând, câte un răspuns pentru prima întrebare. Când o rezolvăm, începem să ghicim a doua întrebare. Fiindcă răspunsurile sunt distribuite aleator, această metodă trimite în medie aprox. **70** de seturi.
- O optimizare simplă: dacă am rămas cu o singură variantă neîncercată pentru o întrebare, știm că e cea corectă.
- Observăm că ne permitem să încercăm în paralel răspunsuri pentru întrebări din 2 categorii diferite, fiindcă acestea având punctaje diferite, putem deduce din punctajul total care dintre cele două răspunsuri au fost corecte. Această strategie ne poate duce în zona a aprox. **40** de seturi.
- Analizând posibilitatea de a încerca în paralel 3 întrebări, observăm că putem obține o ambiguitate în cazul scorului total 5: nu știm dacă e corect răspunsul de la întrebarea grea sau celelalte două. Totuși, putem rezolva ambiguitatea printr-un set suplimentar (care poate include și întrebări neimplicate în primul). Mai mult, observăm că probabilitatea de a obține această ambiguitate este cel mult 1/4, iar în majoritatea cazurilor este mult mai mică. Din acest motiv, numărul mediu de variante excluse per set de această strategie este foarte apropiat de 3.
- O soluție care urmează această strategie paralelă cu 3 răspunsuri, rezolvă eficient ambiguitatea și are grijă să-și adapteze strategia atunci când nu mai sunt disponibile 3 întrebări din categorii diferite (coborând inițial la strategia cu 2 întrebări diferite cât timp e posibilă) poate obține sub **30** de seturi și 100 de puncte.
- Există strategii care optimizează seturile în cadrul unei singure categorii de întrebări. Spre exemplu, dacă la un moment dat avem M întrebări cu câte 2 variante posibile rămase, putem face mai bine decât M întrebări în medie. Putem trimite răspunsuri la câte două întrebări simultan. Cu probabilitate 1/2 ambele sunt corecte sau ambele sunt greșite (deci, ambele rezolvate). Cu probabilitate 1/2 trebuie să trimitem un set în plus pentru a rezolva ambiguitatea. Numărul mediu de seturi este astfel 1.5 pentru 2 intrebari, deci $0.75 * M$ pentru toate M.
- Optimizarea descrisă mai sus poate fi generalizată într-un algoritm de tip divide-et-impera care începe prin a număra câte întrebări au răspunsul 0, 1, etc. și continuă prin a face același lucru recursiv pe submulțimi mai mici, apelul recursiv putând fi oprit când toate întrebările din submulțimea apelată sunt rezolvate de răspunsul fixat, sau dimpotrivă, niciuna dintre întrebări nu este rezolvată de răspunsul fixat.
- Strategiile și optimizările de mai sus pot fi generalizate și combinate. O astfel de soluție hibridă poate obține chiar un număr mediu de **23.9** de seturi. În curând vom publica problemele pe infoarena.ro și vă vom invita să coborâți sub acest prag :).