DESCRIEREA SOLUȚIILOR
LOTUL NAȚIONAL DE INFORMATICĂ
CLUJ, 11-18 MAI 2022
BARAJ JUNIORI 3

Cai

Propusă de: prof. Marius Nicoli – Colegiul Național "Frații Buzești", Syncro Soft – Craiova

Problema admite o soluție constructivă, greedy și are diverse abordări.

Pentru N impar se poate porni din mijlocul primei linii apoi se continuă cu salturi de tip N până se completează conturul. Se ajunge ca ultima celulă acoperită pe acest contur să fie vecină cu aceea de pornire, apoi se poate face un salt de tip $N-1$ și ajungem în mijlocul altei laturi a conturului de ordin $N-2$ care mai rămâne de completat. Procedeul se repetă până la completarea întregii matrice.

Pentru N par, odată ce pornim dintr-un element al conturului exterior vom putea acoperi jumătate dintre elementele acestuia cu salturi de tip N. Este apoi necesar un salt de tip $N-1$ și apoi reușim să completăm și celelalte elemente de pe primul contur cu salturi de tip N. După un alt salt de ordin $N-1$ ne poziționăm pe conturul de ordin $N-2$ și putem relua procedeul.

Se pot face abordări similare cu poziții diferite de start. Atenție trebuie avută însă la cazul în care N este impar, deoarece ultimul salt trebuie să fie făcut de dimensiune 2, nu 1. Spre exemplu, pentru $N = 3$:

1	4	7
6	9	2
3	8	5
Incorect		

6	1	4
3	9	7
8	5	2
Corect		

O abordare matematică tratează cele patru etape ale acoperirii unui inel cu mutări. O abordare greedy alege întotdeauna o mutare maximală.

LoopOver

Propusă de: student Theodor-Gabriel Tulbă-Lecu – Universitatea Politehnica București
student Pop Ioan Cristian – Universitatea Politehnica București

Cerința 1. Pentru a rezolva prima cerință este suficient să observăm că dacă operațiile sunt fie pe linii, fie pe coloane, atunci acestea sunt independente. Astfel, este suficient să rezolvăm fiecare linie sau coloană în parte. Ne rămâne să calculăm care este numărul minim de mutări care trebuie făcute pentru a permuta ciclic o linie (sau o coloană). Deoarece avem două direcții (mutări de tip L sau R) vom alege minimul dintre acestea.

Complexitatea acestei soluții este $\mathcal{O}(n^2)$ iar rezolvarea primei cerințe obține 19 puncte.

Cerința 2.

Soluția 1. Întrucât pentru primul subtask se garantează că numărul total de aplicări este maxim 10 000 putem să simulăm întregul proces astfel:

(1) ne inițializăm matricea de $n \times n$ – acest pas are complexitate $\mathcal{O}(n^2)$
(2) aplicăm cele m operații (fiecare operație poate fi realizată în $\mathcal{O}(n)$) – acest pas are complexitate $\mathcal{O}(nm)$
(3) dacă matricea la pasul curent este identică cu matricea inițială, ne oprim și afișăm numărul de pași, altfel continuăm – acest pas poate fi realizat în $\mathcal{O}(n^2)$

Astfel, complexitatea acestei soluții este $\mathcal{O}(nr\_pasi(n^2+nm))$ și obține aproximativ 9 puncte.

Soluția 2. În continuare, pentru a rezolva problema, va trebui să facem următoarea observație: Întrucât matricea conține n^2 elemente distincte de la 1 la n^2, atunci aceasta este o permutare. Mai mult, dacă am liniariza matricea (adică am construi un vector de lungime n^2 ce este alcătuit din concatenarea liniilor matricei) inițială atunci aceasta este chiar permutarea identitate ($p[i] = i$).

În continuare, vom definii ordinul unei permutări p ca fiind numărul de aplicări ale acesteia ($p[p[p[\dots p[i]\dots]]]$) astfel încât să revenim la permutarea identitate.

Astfel, problema se reduce la a afla ordinul permutării obținute din liniarizarea matricei.

Pentru a afla ordinul unei permutări va trebui să mai facem câteva observații:

Definim un ciclu de lungime k al unei permutări p o submulțime de poziții din permutare $a_1, a_2, \dots, a_k$ astfel încât $p[a_i] = a_i + 1$ pentru oricare $1 \leq i < k$ și $p[a_k] = a_1$.

Dacă aplicăm o permutare p de x ori, atunci fiecare element dintr-un un ciclu de lungime k va deveni: $a_i \to p[p\dots p[a_i]\dots]] = a_{i+x(mod\ k)}$. Deci, dacă x este multiplu de k, atunci $p[a_i] = a_i$.

În concluzie, numărul minim de aplicări ale unei permutari va fi cel mai mic număr care este multiplu al tuturor lungimilor ciclilor acelei permutari.

Pentru a calcula lungimile ciclilor, putem să alegem orice element i din permutare nevizitat, îl vizităm, iar apoi continuăm construcția ciclului de la $p[i]$, până găsim primul element deja vizitat. Astfel, cum vom vizita fiecare element o singură dată, complexitate acestui algoritm este $\mathcal{O}(n^2)$.

Pentru a calcula cel mai mic multiplu comun al indicilor modulo un număr dat, va trebui să ne construim un vector de frecvență $fr[i]$ = numărul de apariții ale numărului i în descompunerea în factori primi celui mai mic multiplu comun al ciclilor. De fiecare dată când găsim un nou ciclu, îl vom descompune în factori primi, și pentru fiecare divizor prim x care apare de y ori în factorizare, vom calcula $fr[x] = \max(fr[x], y)$. La final, vom calcula răspunsul cerut folosind acest vector de frecvență.

Complexitatea finală a acestei soluții este $\mathcal{O}(n^2 + nm)$.

Switch Letters

Propusă de: prof. Dan Pracsiu, Liceul Teoretic "Emil Racoviță" Vaslui
Radu Voroneanu – Google, Zürich

Soluția 1 – 64-70p. propusă de Dan Pracsiu

Deoarece sunt 26 de litere mici, asociem literei a valoarea 0, literei b valoarea 1,..., iar lui z valoarea 25.

Pentru fiecare literă $i = 0\dots25$ construim vectorul $a[i]$ de n valori binare în care, dacă la poziția j în șirul s litera este i, atunci $a[i][j] = 1$, în caz contrar $a[i][j] = 0$. Deci dacă litera a apare în șirul s la pozițiile 3, 7, și 9, atunci, pentru că lui a i s-a asociat numărul 0, $a[0][3] = 1$, $a[0][7] = 1$ și $a[0][9] = 1$.

O operație $switch(i, j, c_1, c_2)$ executată brut înseamnă parcurgerea secvenței $s[i]$ și pentru orice poziție p, $i \leq p \leq j$, dacă $a[i_1][p] = 1$, atunci $a[i_2][p]$ devine 1, iar $a[i_1][p]$ devine 0 (unde am notat cu i_1 numărul asociat literei c_1 și cu i_2 numărul asociat literei c_2).

Pentru a optimiza efectuarea operației switch procedăm astfel. Fiecare $a[i]$ ($i = 0\dots25$) este un vector de tip `unsigned long long` de lungime 1024, și deoarece fiecare componentă este reprezentată pe 64 de biți fără semn, atunci sunt în total $64 \times 1024 = 65536$ biți. Pentru fiecare operație $switch(i, j, c_1, c_2)$ împărțim intervalul $[i, j]$ în subintervale de lungime 64 și acum parcurgerea secvenței $s[i\dots j]$ se va executa de 64 de ori mai rapid.

Solutia 2 – 100p. Radu Voroneanu

Vom itera de la stânga la dreapta prin șirul s, ținând cont la fiecare pas de care operații *switch* includ poziția curentă. Dacă știm care operații afectează poziția i, atunci putem determina care operații afectează poziția $i + 1$ astfel:

- Se elimină operațiile ce au capătul din dreapta pe poziția i.

- Se adaugă operațiile ce au capătul din stânga pe poziția $i+1$.

Avem acum nevoie de o structură de date care să ne permită adăugarea și eliminarea de operații și recalcularea eficientă a transformărilor. Vom folosi un arbore de intervale pentru aceasta. Atenție, arborele este ținut pentru cele m operații *switch*, nu pe șirul inițial de caractere s. Pentru fiecare nod al arborelui, vom calcula un vector $a[nod]$ unde $a[nod][i]$ reprezintă litera în care se transformă litera i, folosind operațiile *switch* din interval lui nod. Pentru o stocare eficientă se reindexează literele de la 0 la 26.

Vom prezenta mai întâi cum se poate recalcula $a[nod]$ folosind informația din fiul stâng st și fiul drept dr. Pentru o literă oarecare ch, litera o sa fie mai întâi transformată în $a[st][ch]$ folosind operațiile *switch* din prima jumătate a intervalului lui nod, și apoi această literă o să se transforme în $a[dr][\,a[st][ch]\,]$ folosind operațiile din a doua jumătate a intervalului. Într-un final, dacă știm ca $a[st]$ sau $a[dr]$ au fost schimbate, putem sa recalculăm și pe $a[nod]$ în $\mathcal{O}(1)$ (cu o constantă de 26). Folosind această abordare, putem să adaugăm sau să eliminăm operații în arbore în $O(\log(m))$.

La un pas oarecare i în iterația prin s, se vor adăuga în arbore toate operațiile care încep pe poziția i și se vor elimina toate operațiile care au capătul drept pe poziția $i-1$. După acest update al arborelui, răspunsul îl găsim în nodul rădăcina, ce reprezintă tot intervalul - mai specific în $a[radacina][s[i]]$. Astfel, schimbarea literei are complexitatea $\mathcal{O}(1)$.

Complexitatea totală este $\mathcal{O}(\Sigma \cdot m \log m + n)$.

Soluția 3 – 64 - 90p. Radu Voroneanu

Vom construi un arbore de intervale pentru șirul de caractere, în care în fiecare nod ținem un vector $a[nod]$ cu semnificația că litera i se transformă în litera $a[nod][i]$. Update-urile trebuiesc făcute într-un mod *lazy* pe arbore. Din cauza modului de construcție *lazy*, întotdeauna operațiile dintr-un nod tată vor fi efectuate după operațiile din copiii acestuia. Astfel, în operația de împingere a operațiilor din tată în copii se poate folosi recurenta $a[fiu][i] = a[tata][a[fiu][i]]$. Query-urile sunt efectuate la o singură poziție, nu la un interval.

Deoarece lungimea șirului poate să fie mai mare decât numărul de operații, se pot normaliza pozițiile șirului în funcție de capetele din stânga și dreapta a tuturor operațiilor *switch*, obținându-se astfel un arbore de intervale pe un șir de lungime $2m$.

Comparativ cu Soluția 2, aceasta din păcate prezintă o constantă foarte mare în practică.

Soluția 4 – 64 - 80p. prof. Ionel Vasile Pit Rada

Se împarte șirul dat în $\sqrt{n}$ secvențe de lungime $\sqrt{n}$, pe care le vom numi în continuare „*cutii*". Pentru fiecare cutie se construiesc 26 de liste înlănțuite, câte o listă pentru fiecare literă mică. În aceste liste se salvează pozițiile corespunzătoare pentru fiecare literă. Pentru fiecare interogare (i, j, c_1, c_2) se stabilesc mai întâi cutiile din care fac parte pozițiile i și respectiv j. Vom nota aceste cutii cu k_1 și k_2, evident $k_1 \leq k_2$. Dacă avem $k_1 = k_2$, atunci parcurgem în cutia k_1 pozițiile cuprinse între i și j din lista corespunzătoare literei c_1 și le mutăm în lista literei c_2. Dacă avem $k_1 < k_2$, atunci:

- parcurgem în cutia k_1 pozițiile mai mari sau egale cu i din lista corespunzătoare literei c_1 și le mutăm în lista literei c_2,
- parcurgem în cutia k_2 pozițiile mai mici sau egale cu j din lista corespunzătoare literei c_1 și le mutăm în lista literei c_2,
- parcurgem cutiile $k_1+1, \ldots, k_2-1$ și mutăm, pentru fiecare cutie în $\mathcal{O}(1)$, întreaga listă de poziții pentru c_1 în lista de poziții pentru c_2.

Pentru fiecare interogare complexitatea este $\mathcal{O}(\sqrt{n})$. Complexitatea totală este: $O(\Sigma\sqrt{n} + n + m\sqrt{n})$.

Soluția 5 – 64 - 90p. Mihail Cosmin Pit Rada

O soluție similară pe ideea de împărțire în secvențe de lungime $\sqrt{n}$. Vom menține vectorul $a[c][l]$ în care se pun toate literele din cutia cu ordinul c care s-au transformat în litera l. Aceste litere nu se vor ține ca liste, ci se va folosi o mască binară - spre exemplu, litera a corespunde bitului cu ordinul 0, litera b bitului cu ordinul 1 ș.a.m.d.. Pentru efectuarea unei operații de tip

$switch(i, j, c_1, c_2)$, se va calcula intervalul de cutii care trebuiesc schimbate $[k_i, k_j]$. Pentru orice cutie $k_i < k < k_j$, tot ce trebuie făcut este mutarea măștii binare a lui $a[k][c_1]$ în masca binară a lui $a[k][c_2]$ printr-un SAU logic, și apoi resetarea măștii binare a lui $a[k][c_1]$. Pentru cutiile corespunzătoare lui i și j (c_i și c_j) se va schimba mai întâi șirul original s folosind informațiile din $a[k_i][]$ și $a[k_j][]$, după care se va aplica tot pe s schimbările rămase pentru operația *switch*.

Complexitatea totală este $\mathcal{O}(n + m\sqrt{n})$, însă aceasta se comporta mult mai bine în practică datorita constantei mici din fata complexității.

Echipa

Problemele pentru această etapă au fost pregătite de:

- Nicoli Marius – Colegiul Național ,,Frații Buzești", Syncro Soft, Craiova
- Boian Flavius – Colegiul Național ,,Spiru Haret", Târgu-Jiu
- Bunget Mihai – Colegiul Național ,,Tudor Vladimirescu", Târgu-Jiu
- Cerchez Emanuela – Colegiul Național ,,Emil Racoviță", Iași
- Chește Ciprian – Liceul Tehnologic ,,Grigore Mosil", Buzău
- Frâncu Cristian – Clubul Nerdvana București
- Lica Daniela – Centrul Județean de Excelență Prahova, Ploiești
- Manz Victor – Colegiul Național de Informatică ,,Tudor Vianu", București
- Nodea Gheorghe-Eugen – Colegiul Național ,,Tudor Vladimirescu", Târgu-Jiu
- Oprița Petru – Liceul ,,Regina Maria", Dorohoi
- Pintea Adrian – Inspectoratul Școlar Județean Cluj
- Pițt-Rada Ionel-Vasile – Colegiul Național ,,Traian", Drobeta Turnu Severin
- Pițt-Rada Mihai-Cosmin – Bolt
- Pop Ioan Cristian – Universitatea Politehnica București
- Pracsiu Dan – Liceul Teoretic ,,Emil Racoviță", Vaslui
- Șerban Marinel – Colegiul Național ,,Emil Racoviță", Iași
- Tulbă-Lecu Theodor-Gabriel – Universitatea Politehnica București
- Voroneanu Radu – Google, Zürich