DESCRIEREA SOLUȚIILOR, OLIMPIADA NAȚIONALĂ DE INFORMATICĂ,
BARAJ JUNIORI
9 APRILIE 2022

Autostrada

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

Problema poate fi împarțită în două subprobleme: generarea tabloului bidimensional și numărarea tipurilor de intersecții.

Generarea tabloului bidimensional. Prin generarea tabloului bidimensional ne referim la identificarea suprafețelor ce trebuiesc asfaltate.

Fie a tabloul bidimensional care va indica suprafețele ce trebuiesc asfaltate. Scopul nostru este ca generarea să fie realizată într-un mod ce facilitează numărarea tipurilor de intersecții.

Codificarea tipurilor de celule. Pentru a reține pentru fiecare celulă de ce tip este, putem asocia o anumită valoare pentru fiecare astfel de celulă.

O modalitate de codificare a tipului este ca pentru fiecare direcție față de centrul unei celule să reținem dacă traseul dronei trece prin celulă pe acea direcție utilizând puteri de 2.

Astfel, vom reține pentru celula (i, j) valorile:

1 – dacă traseul dronei trece prin nord
2 – dacă traseul dronei trece prin est
4 – dacă traseul dronei trece prin sud
8 – dacă traseul dronei trece prin vest

La final, vom putea număra câte intersecții + există numărând câte valori din a au exact 4 biți setați cu valoarea 1, câte intersecții $\top$ există numârând câte intersecții au 3 biți setati. Celelalte valori nenule vor reprezenta fie cotituri, fie linii drepte și vor fi numărate ca celule simple.

Dublarea dimensiunilor tabloului. Observăm că în soluția anterioară procesul de codificare este destul de complicat deoarece dacă două celule se învecinează, atunci traseul nu trece neapărat de la o celulă la cealaltă.

Putem totuși să evităm acest caz particular prin următoarea operație: o celulă de coordonate (i, j), se translatează în coordonatele $(2i-1, 2j-1)$. Această operație de *dublare* a tabloului, va crea spații goale intre oricare două celule din tabloul original, aceste spații goale sunt vizitate de dronă doar dacă drona inițial a trecut de la o celulă la cealaltă. Observăm ca celulele din tabloul original se vor afla după trasformare pe coordonate cu indicele liniei, respectiv al coloanei, numere impare.

Astfel, după această trasformare, pentru a determina tipul de celulă din tabloul original este suficient, să numărăm câți vecini cu valori nenule are.

Reunirea intervalelor traseului. Soluțiile precedente pot fi rezolvate în complexități diferite, cea mai simplă fiind parcurgerea iterativă a tuturor pozițiilor. Această metodă de parcurgere ar avea o complexitate $\mathcal{O}(K \cdot N)$ și obține aproximativ 47 de puncte.

Putem obține o complexitate mai bună făcând următoarea observație, o mutare a dronei crează un interval pe linia, respectiv coloana pe care aceasta se deplasează. Astfel, putem reține pentru fiecare linie și coloană toate intervalele de pe aceasta, le sortăm, iar mai apoi le reunim.

Această soluție poate fi implementată in $\mathcal{O}(N^2 + K \log(K))$ și obține aproximativ 90 de puncte.

Șmenul lui Mars (Difference Array). O îmbunătățire a soluției precedente este folosirea șmenului lui Mars (1) pentru a face reunirea intervalelor.

Astfel fiecare dintre cele K operații poate fi efectuată în $\mathcal{O}(1)$, iar complexitatea finală a soluției va deveni $\mathcal{O}(N^2 + K)$.

Această soluție obține 100 de puncte, indiferent de modul de codificare folosit.

Bug

Propusă de: Prof. Emanuela Cerchez – Colegiul Național "Emil Racoviță" Iași
Radu Voroneanu – Google Zürich

Descrierea algoritmului. Vom descrie un algoritm constructiv de determinare a rezultatului.

Vom citi cifrele numărului N şi le vom plasa într-un vector a cu lg elemente.

Vom denumi ,,*bloc*" o secvenţă de lungime minimă de cifre din a care conţine toate cifrele de la $\{0, 1, \ldots, 9\}$.

Vom parcurge a de la final către început şi vom identifica blocurile.

Pentru aceasta vom utiliza un vector caracteristic uz_{10}: $uz_i = 1$, dacă cifra i apare în secvenţa curentă, respectiv 0 în caz contrar.

Când vectorul uz conține 10 valori egale cu 1, am identificat un bloc, îl numărăm şi reţinem poziţia sa de început în vectorul inc.

Apoi resetăm uz (adică îl reiniţializăm cu 0) pentru a ne pregăti pentru construcţia următorului bloc.

După această parcurgere, am partiționat vectorul a în $nrbloc$ blocuri. La începutul lui a este posibil să rămână câteva cifre care nu formează un bloc (notăm acest prefix cu P).

$$a = PB_{nrbloc}B_{nrbloc-1}\ldots B_1$$

Să determinăm c, cea mai mică cifră nenulă care nu apare în prefixul P (1 dacă P este vid).

Toate numerele mai mici sau egale cu $(c-1)99\ldots9$ ($nrbloc$ de 9) pot fi obţinute. Cel mai mic număr care nu poate fi obţinut (rez) va începe cu c. Pentru a determina celelalte $nrbloc$ cifre din rez, parcurgem blocurile de la stânga la dreapta şi determinăm pentru fiecare bloc poziţia pe care apare ultima cifră plasată în rez în blocul respectiv (să notăm această poziţie i). Cifrele din bloc situate după poziţia i evident nu mai formează un bloc. Determinăm cea mai mică cifră care nu apare în blocul curent de la poziţia $i+1$ la finalul blocului. Această cifră va fi plasată în rez.

De exemplu:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
a	1	3	6	7	8	9	0	1	2	3	4	5	8	6	2	7	4	9	4	5	2	3	0	7	1	0

$B_1 = 86274945230710$ (poziţiile de la 13 la 26 din a)

$B_2 = 6789012345$ (poziţiile de la 3 la 12 din a)

$P = 13$ (poziţiile 1 şi 2 din a)

$nrbloc = 2$

Cea mai mică cifră nenulă care nu apare în P este 2. Cel mai mic număr care nu se poate obţine începe cu 2. În blocul B_2 cifra 2 apare pe poziţia 9. Cea mai mică cifră care nu apare în B_2 de la poziţiile 10 la 12 este 0.

Cifra 0 apare în B_1 pe poziţia 23. Cea mai mică cifră care nu apare în B_1 de la poziţia 24 până la 26 este 2. Rezultatul va fi 202.

Caz special: P nu este bloc, dar conţine toate cifrele nenule. În acest caz orice număr de $nrbloc + 1$ cifre se poate obţine. Ca urmare vom plasa la începutul rezultatului 10 şi continuăm parcurgerea blocurilor în acelaşi mod.

Corectitudinea algoritmului. Vom demonstra în primul rând că numărul rez nu se poate obţine.

Fie c prima cifră a lui rez. Aceasta nu apare în P, deci convenabil ar fi să utilizăm prima ei apariţie din B_{nrbloc}. Sufixul lui B_{nrbloc} care începe după prima apariţie a lui c nu este bloc, deci nu conţine toate cifrele. Fie acum c cea mai mică cifră care nu apare în acest sufix (următoarea din rez). Aceasta trebuie luată obligatoriu din următorul bloc ş.a.m.d. Când ajungem la ultima

cifră a lui *rez*, aceasta nu mai există în ultimul sufix (cel al lui B_1) deci clar *rez* nu se poate obţine.

Să demonstrăm că *rez* este cel mai mic număr care nu se poate obţine.

Să presupunem prin reducere la absurd că există $X < rez$ care nu se poate obţine. Dacă X are mai puţine cifre decât *rez*, sigur se poate obţine (iau cifra i din X din blocul B_i). Deci musai ca X să aibă aceeaşi lungime cu *rez*. Mai mult decât atât X trebuie să înceapă cu aceeaşi cifră ca *rez* (în caz că luăm o cifră mai mică numărul se poate obţine pentru că acea cifră mai mică apare în P, apoi celelalte cifre le luăm din blocurile următoare; în caz că luăm o cifră mai mare X nu ar mai fi mai mic decât *rez*). Prin acelaşi raţionament, a doua cifră a lui X trebuie să coincidă cu a doua cifră din *rez* ş.a.m.d.

Triprime

Propusă de: Cristian Frâncu – Clubul Nerdvana România

Să enumerăm câteva soluții în ordinea eficienței lor.

Soluția 1 – 9p. Parcurgem numerele X de la A la B și le descompunem în factori primi. Oprim descompunerea cel târziu când divizorul depășește rădăcina pătrată a numărului, sau imediat ce numărul nu mai poate fi triprim, de exemplu dacă găsim un divizor prim la putere mai mare ca unu.

Soluția va depăși timpul pentru $B > 1.5$ milioane

Timp: între $\mathcal{O}(\frac{(B-A)\sqrt{B}}{\log B})$ și $\mathcal{O}((B-A)\sqrt{B})$

Memorie: $\mathcal{O}(1)$

Soluția 2 – 15p. Aceeași soluție ca cea anterioară, testând doar divizorii impari.

Soluția va depăși timpul pentru $B > 2.5$ milioane

Timp: între $\mathcal{O}(\frac{(B-A)\sqrt{B}}{\log B})$ și $O((B-A)\sqrt{B})$

Memorie: $\mathcal{O}(1)$

Soluția 3 – 27p. Folosim o precalculare a numerelor prime pentru a descompune mai rapid numerele X în factori primi. Numerele triprime au exact trei factori primi. Cel mai mic factor prim fiind 2, rezultă că al doilea factor prim nu poate fi mai mare decât $\sqrt{390\,000\,000/2}$ care este aproximativ 14 000. Vom precalcula numerele prime până la 14 000 folosind ciurul lui Eratostene. La descompunerea unui număr X vom căuta primii doi factori primi printre numerele prime precalculate. Al treilea factor prim poate fi printre numerele precalculate sau nu. Dacă nu îl găsim printre ele îl vom căuta ca la soluția anterioară, printre numerele impare.

Soluția va depăși timpul pentru $B > 4.5$ milioane

Timp: între $\mathcal{O}(\frac{(B-A)\sqrt{B}}{\log B})$ și $O((B-A)\sqrt{B})$

Memorie: $\mathcal{O}(\sqrt{B} + \frac{\sqrt{B}}{\log B})$ adică un vector ciur de circa 14 000 de bytes și un vector de sub 2 000 de numere prime ce ocupă circa 8KB, în total aproximativ 22KB

Soluția 4 – 51p. Calculăm numărul de divizori primi al tuturor numerelor până la B folosind ciurul lui Eratostene. Memoria ne permite un vector ciur de caractere până la aproximativ 64 milioane. Soluția va depăși, însă, timpul, înainte de această limită.

Odată calculat ciurul numărului de divizori primi reparcurgem acest ciur. În această a doua parcurgere pentru fiecare număr prim P vom anula toate numerele divizibile cu P^2, setând numărul de divizori la 0.

La o a treia parcurgere a ciurului, între A și B vom număra câte numere cu trei divizori primi avem.

Soluția va depăși timpul pentru $B > 35$ milioane.

Timp: $\mathcal{O}(\text{B} \log B)$

Memorie: $\mathcal{O}(B)$ adică aproximativ 35MB la momentul când apare depășirea de timp

Soluția 5 – 85-100p. Care este numărul prim maxim care poate apărea într-un număr triprim? Alegând 2 și 3 ca cele mai mici două numere prime, al treilea poate fi maxim $\frac{B}{6}$, adică 65 de milioane. Un vector ciur de caractere va încăpea în memoria disponibilă de 64MB, adică 67 108 864 bytes.

Vom calcula numerele prime până la $\frac{B}{6}$. Apoi vom număra în ciur tripleții de numere prime al căror produs este mai mic sau egal cu X, să denumim acest număr N_X. Răspunsul cerut va fi $N_B - N_{A-1}$.

Cum calculăm numărul de numere triprime până la X? Prin metoda cunoscută unora drept 'metoda celor doi pointeri'. Vom folosi trei indici i, j, k în ciur, astfel încât i, j, k să fie prime și produsul lor să nu depășească X. La fiecare avans al lui i la următorul număr prim, vom calcula j ca fiind următorul număr prim după i și apoi vom porni de la următorul număr prim k și vom avansa k până ce produsul $i \times j \times k$ depășește X, numărând câte numere prime T se află între j și k. La acest moment putem adăuga T numere triprime la rezultat. Apoi avansăm j la următorul număr prim și reducem k către primul număr prim care aduce produsul $i \times j \times k$ sub X. Avem grijă să actualizăm T, numărul de numere prime dintre j și k. Apoi adunăm din nou numărul T la rezultat.

Continuăm în acest fel până ce j îl ajunge pe k. Moment la care avansăm i și reluăm calculul de mai sus.

Această metodă poate lua 100p, dar, în funcție de implementare, este posibil să depășească timpul pe unele teste mari.

Timp: $\mathcal{O}(B \log B)$ datorat numărării numerelor triprime.

Memorie: $\mathcal{O}(B)$ adică 61MB

Soluția 6 – 100p. Procedăm ca la soluția 5, dar calculăm doar numerele prime impare. Putem folosi un ciur al lui Eratostene modificat, care va ocupa jumătate din memorie, adică 30.5MB. Astfel avem loc pentru un vector separat în care vom stoca toate numerele prime din acest ciur. Sunt aproape 4 milioane de numere prime mai mici decât 65 milioane, care vor ocupa circa 16MB.

Pe vectorul de numere prime putem aplica numărarea de numere triprime mai eficient, folosind aceeași metodă de mai sus, a celor doi pointeri.

Timp: $\mathcal{O}(B \log \log B)$ deoarece numărarea de numere triprime este liniară în B

Memorie: $\mathcal{O}(B + \frac{B}{\log B})$ adică circa 30.5MB pentru ciur și încă circa 15.5MB pentru stocarea numerelor prime, aproximativ 46MB

Soluția 7 – 100p. Tot pentru a folosi mai puțină memorie putem proceda ca la soluția 5, dar folosind un vector ciur pe biți. Apoi procedăm ca la soluția 6, colectând separat numerele prime în vederea numărării eficiente a numerelor triprime.

Timp: $\mathcal{O}(B \log \log B)$

Memorie: $\mathcal{O}(B + \frac{B}{\log B})$ adică circa 7.5MB pentru ciur și 15.5MB pentru numerele prime, total aproximativ 21MB

Soluția 8 – 100p. Pentru a folosi și mai puțină memorie (ceea ce duce și la scăderea timpului de executare) putem combina soluțiile 6 și 7, calculând un ciur al lui Eratostene doar pentru numere impare și folosind un vector de biți.

Timp: $\mathcal{O}(B \log\log B)$

Memorie: $\mathcal{O}(B + \frac{B}{\log B})$ adică circa 3.5MB pentru ciur și 15.5MB pentru numerele prime, total aproximativ 19MB

Soluția 9 – 100p. O soluție cu o cu totul altă idee este următoarea: folosim soluția numărul 3 pentru a descompune în factori primi toate numerele de la 1 la 390 milioane. La fiecare 400 000 de numere scriem într-un fișier numărul de numere triprime găsite până acum. La final vom avea sume parțiale ale numerelor triprime până la X, cu X variind din 400 000 în 400 000. Vom obține 975 de astfel de numere. Timpul de executare al acestui program va fi de circa 10 minute.

Apoi, în programul soluție, vom include aceste 975 de numere ca vector preinițializat. Putem calcula acum N_X = numărul de numere triprime mai mici sau egale cu X astfel: calculând $X/400\,000$ aflăm rapid numărul de numere triprime până la ultimul interval întreg de 400 000. Pentru restul de numere, până la X, vom folosi algoritmul de la soluția 3.

Răspunsul la problemă va fi $N_B - N_{A-1}$.

Aceasta este una din cele mai rapide soluții.

Timp: între $\mathcal{O}(\frac{K\sqrt{B}}{\log B})$ și $\mathcal{O}(K\sqrt{B})$ unde am notat cu K lungimea unui interval de precalculare, adică 400 000.

Memorie: $\mathcal{O}(B/K + \sqrt{B} + \frac{\sqrt{B}}{\log B})$ este memoria ocupată de ciurul până la 14 000, vectorul de numere prime până la 14 000, circa 2 000 de elemente întregi și vectorul de sume parțiale de 975 de elemente, adică aproximativ 26KB.

Echipa

Problemele pentru această etapă au fost pregătite de:

- Nicoli Marius – Colegiul Național ,,Frații Buzești", Syncro Soft, Craiova
- Cerchez Emanuela – Colegiul Național ,,Emil Racoviță", Iași
- Frâncu Cristian – Clubul Nerdvana București
- Lica Daniela – Centrul Județean de Excelență Prahova, Ploiești
- Manolache Gheorghe – Colegiul Național de Informatică, Piatra Neamț
- Manz Victor – Colegiul Național de Informatică ,,Tudor Vianu", București
- Muntean Radu – ETH, Zürich
- Opriță Petru – Liceul ,,Regina Maria", Dorohoi
- Pițˇ-Rada Ionel-Vasile – Colegiul Național ,,Traian", Drobeta Turnu Severin
- Pop Ioan Cristian – Universitatea Politehnica București
- Șerban Marinel – Colegiul Național ,,Emil Racoviță", Iași
- Tulbă-Lecu Theodor-Gabriel – Universitatea Politehnica București
- Voroneanu Radu – Google, Zürich

Referințe

[1] Șmenul lui Mars – infoarena