Iași, România

Olimpiada BALCANICĂ de informatică

În continuare vă vom prezenta soluțiile celor șase probleme propuse la cea de-a XI-a ediție a Olimpiadei Balcanice de Informatică. Aceste soluții au fost realizate de redacția GInfo pe baza soluțiilor oficiale prezentate de către autorii problemelor.

P060313: Secvența Farey

Pentru a rezolva problema, vom aborda următoarea subproblemă:

"*Dându-se o fracție A / B, nu neapărat ireductibilă, să se afle câte fracții din secvența Farey sunt mai mici decât ea.*"

Având o funcție care rezolvă această subproblemă, algoritmul pentru problema din concurs ar fi următorul:

- determinăm numărul X astfel încât răspunsul să fie între X/N și $(X + 1)/N$, folosind căutarea binară pentru X și apelând funcția de mai sus de log N ori;
- determinăm toate fracțiile A / B din intervalul X/N ... $(X + 1) / N$. Pentru un B fixat există cel mult un A care să corespundă; acesta poate fi determinat în $O(1)$;
- sortăm aceste fracții în $O(N \cdot \log N)$ și determinăm rezultatul.

Cum construim funcția descrisă mai sus?

Fie $C[j]$ numărul de fracții ireductibile i/j mai mici decât X/N.

Algoritmul este bazat pe următoarea observație:

$$C[j] = \lfloor X \cdot B / N \rfloor - \sum_{D|j} C[D].$$

O implementare directă, care testează fiecare D dacă este divizor al lui j, are ordinul de complexitatea $O(N^2)$.

O implementare mai bună, inspirată din *ciurul lui Eratostene*, este următoarea: la pasul j, cunoscând $C[j]$, îl scădem din toți multiplii lui j. Timpul de execuție al funcției este $O(N \cdot \log N)$.

Analiza complexității

Ordinul de complexitate al operațiilor de citire a datelor de intrare și scriere a rezultatelor este $O(1)$.

Operația de determinare a numărului X are ordinul de complexitate $O(N \cdot \log^2 N)$.

Pasul de după determinarea lui X are ordinul de complexitate $O(N \cdot \log N)$, dat de sortare.

În concluzie, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(N \cdot \log^2 N)$.

P060314: Traversare

Problema se rezolvă prin metoda *programarii dinamice*.

Toate calculele pentru numărul de posibilități de traversare se vor face modulo 9901.

Trebuie calculat, pentru fiecare cutie i, $A[i]$ = numărul de posibilități de a traversa lacul, ultima cutie fiind i.

Metoda clasică, folosind $j = 1, 2, ..., i - 1$ ca penultima cutie (dacă diferența de înălțime o permite) și calculând $A[i]$ pe baza valorilor din $A[j]$, ar avea ordinul de complexitate $O(N^2)$.

Există mai multe metode de complexitate $O(N \cdot \log N)$. Vom prezenta una dintre ele.

Inițial sortăm înălțimile cutiilor.

Construim o structură de date sub forma unui arbore binar complet.

Frunzele arborelui corespund înălțimilor cutiilor. A K-a frunză corespunde celei de-a K-a înălțimi ($1 \le K \le N$) și va conține numărul de posibilități (modulo 9901) în care se poate realiza o traversare, terminând cu o cutie care are această înălțime.

Nodurile intermediare din arbore conțin suma valorilor din cei doi fii modulo 9901.

Inițial considerăm că s-au folosit `0` cutii, deci arborele este plin cu `0`.

La introducerea unei cutii de înălțime X se realizează următoarele operații:

- se caută binar înălțimile $X - H$ și $X + H$ în vectorul sortat de înălțimi;
- se calculează, în $O(\log N)$, numărul de posibilități în care se poate realiza o traversare, terminând cu cutii având înălțimea între $X - H$ și $X + H$. Lăsăm detalierea acestui

pas ca exercițiu pentru cititor; având explicată structura de date, exercițiul nu este foarte dificil.
- se adună 1 la acest număr, datorită posibilității de a folosi cutia nouă ca primă cutie;
- numărul de posibilități pentru înălțimea X crește cu numărul obținut și se actualizează valorile din arbore, pentru nodul tată, tatăl acestuia etc., în $\log N$ pași.

În final, din numărul total de posibilități (dat de rădăcina arborelui) se scade N, deoarece nu sunt acceptate posibilitățile de traversare cu o singură cutie.

Analiza complexității

Ordinul de complexitate al operației de citire a datelor de intrare este $O(N)$.

Calculele pentru introducerea unei noi cutii au ordinul de complexitate $O(\log N)$. Pentru toate cele N cutii, operațiile cu arborele au ordinul de complexitate $O(N \cdot \log N)$.

Ordinul de complexitate a operației de scriere a rezultatelor este $O(1)$.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(N \cdot \log N)$.

P060315: Compania

Diferența cerută este 0 dacă N este divizibil cu 3 și 1 în caz contrar. Există mai multe metode, bazate pe *greedy* și pe *programare dinamică* cu ajutorul cărora se poate obține o soluție cu diferență minimă.

Vom prezenta soluția autorului problemei, lăsând ca exercițiu cititorului găsirea unei soluții bazate pe programarea dinamică, fără a folosi re-colorarea (prezentată mai jos). Precizăm că o astfel de soluție necesită atenție deosebită la implementare, datorită numărului mare de cazuri.

Compania este un arbore, care va fi colorat cu trei culori.

Fie un subarbore al arborelui, având ca rădăcină un nod R. Algoritmul va colora arborele respectiv în trei culori, astfel încât diferența să fie minimă. Această colorare poate fi modificată pe parcurs, după cum se va vedea în continuare.

Dacă R este o frunză, atunci va fi colorată cu culoarea 1.

Dacă R are un singur fiu, atunci se colorează subarborele care are ca rădăcină fiul respectiv, după care R este colorat cu culoarea cea mai puțin folosită, care este diferită de culoarea fiului. Dacă sunt două culori care îndeplinesc această proprietate, se va alege una oarecare. Analizând algoritmul, se poate demonstra că întotdeauna va exista cel puțin o culoare disponibilă.

Dacă R are doi fii, *FiuStânga* și *FiuDreapta*, se vor colora independent cei doi subarbori care au ca rădăcini acești fii. Apoi, arborele pornind din *FiuDreapta* este recolorat, folosind o permutare a celor trei culori și colorarea existentă, astfel încât să se respecte condițiile din enunț și diferența în subarborele cu rădăcina R să fie minimă. R se colorează cu culoarea rămasă disponibilă.

Se observă că o astfel de re-colorare menține respectarea condițiilor în arborele cu rădăcină în *FiuDreapta*, deoarece două noduri colorate diferit rămân colorate diferit după re-colorare.

Analizând toate cazurile, se observă că o re-colorare care să ducă la o soluție validă pentru subarborele cu rădăcina în R există întotdeauna, indiferent de colorarea subarborelui cu rădăcina în *FiuStânga*.

Pentru a alege permutarea pentru re-colorare se vor încerca toate cele $3! = 6$ variante. Nu se va realiza re-colorarea efectivă, ci se va păstra, pentru *FiuDreapta*, permutarea folosită.

Algoritmul este rulat din rădăcina arborelui. Se obține astfel o strategie de colorare validă, care conduce la diferența minimă.

Presupunând că s-ar fi făcut re-colorarea efectivă la fiecare pas, culorile nodurilor ar fi fost determinate. Aceasta ar fi condus însă la o creștere a complexității, inacceptabilă pentru limitele date.

Pentru afișarea soluției se realizează o a doua parcurgere a arborelui. În cadrul acesteia se realizează colorarea efectivă, combinând permutările culorilor din fiecare nod. Pentru nodurile unde nu s-a făcut recolorare se va considera permutarea identică (1 2 3).

Analiza complexității

Operațiile de citire a datelor de intrare și scriere a rezultatelor au ordinul de complexitate $O(N)$.

La prima parcurgere, pentru fiecare nod se execută un număr de operații limitat de o constantă (cel mult două apeluri recursive și analizarea celor șase permutări posibile pentru re-colorarea subarborelui drept). Ordinul de complexitate al acestei parcurgeri este $O(N)$. Dacă s-ar fi făcut re-colorarea efectivă, se observa că unii subarbori ar fi fost re-colorați de mai multe ori, ceea ce ar fi condus la o creștere a complexității.

Cea de-a doua parcurgere are același ordin de complexitate, deoarece numărul de operații pentru fiecare nod este de asemenea limitat.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $O(N)$.

P060316: Consiliu de trib

Reformulăm cerința problemei: *Se consideră un arbore. Să se aleagă o ordine de colorare a nodurilor, astfel încât oricare două noduri să fie colorate diferit, fiecare nod să fie colorat cu prima culoare disponibilă și numărul de culori folosit să fie maxim. Se cere doar numărul de culori, nu și o colorare.*

Rezolvarea se bazează pe metoda *programării dinamice*.

Introducem următoarea propoziție ajutătoare:

Fie un subarbore cu o rădăcină fixată. Dacă putem colora rădăcina cu culoarea K, folosind numai nodurile din subarbore, atunci o putem colora cu orice culoare mai mică decât K.

Demonstrația este simplă și se bazează pe schimbarea ordinii de colorare a fiilor. Concluzia este că trebuie determinată, pentru fiecare nod, culoarea maximă care se poate

obține pentru acesta, folosind nodurile din subarborele care îl are ca rădăcină.

Pentru a rezolva problema se poate alege, pe rând, fiecare nod ca fiind rădăcină a arborelui (deci ca fiind ultimul nod colorat). Pentru fiecare astfel de alegere se calculează culorile maxime pentru fiecare nod, folosind un algoritm simplu de programare dinamică cu ordinul de complexitate O(N). Complexitatea acestei metode este $O(N^2)$, deci metoda nu se va încadra în timp. Pe de altă parte, dacă se pornește dintr-un singur nod, se poate demonstra că soluția este cu cel mult 1 mai slabă decât soluția optimă; pornind din mai multe noduri, fără a ieși din timp, șansele de a obține soluția optimă cresc.

Pentru un nod oarecare K, considerat ca ultimul nod colorat (deci ca nouă rădăcină, reorientând arborele), ne interesează culoarea maximă cu care poate fi colorat fiecare vecin din arbore, folosind pentru aceasta numai nodurile din subarborii induși de vecinii respectivi.

Se observă că metoda precedentă ar fi dus la repetarea a numeroase calcule. Există, deci, șanse să găsim o metodă mai eficientă.

Privind problema din punct de vedere al muchiilor, ne interesează, pentru fiecare muchie, numărul maxim de culori cu care pot fi colorate capetele ei, dacă scoatem muchia respectivă și analizăm separat cei doi subarbori formați. Trebuie să calculăm două valori pentru fiecare muchie. Acestea se pot calcula folosind o procedură recursivă; se observă că fiecare muchie este parcursă de un număr constant de ori.

Pentru aflarea optimului, se analizează fiecare muchie. Dacă maximul pentru ambele capete este egal cu o valoare M, atunci arborele se poate colora cu $M + 1$ culori, alegând unul din capete ca ultim nod. Dacă maximele sunt diferite, atunci se alege ca ultim nod capătul cu maximul mai mare.

Se alege varianta cea mai bună dintre cele N - 1 variante date de muchiile arborelui.

Analiza complexității

Ordinul de complexitate al operației de citire a datelor de intrare este $O(N)$, iar al operației de scriere a rezultatelor este $O(1)$.

În concluzie, ordinul de complexitate al algoritmului este $O(N)$, deoarece fiecare muchie este parcursă de un număr constant de ori.

P060317: Euro

Există mai multe rezolvări pentru această problemă.

Soluția evidentă, bazată pe metoda *programării dinamice*, are ordinul de complexitate $O(N^2)$. Aceasta constă în a calcula, pentru fiecare i între 1 și N, valoarea $C[i]$ a câștigului maxim care se poate obține în primele i zile. Acesta depinde de câștigul maxim care se poate obține în primele j zile, cu $j < i$. Această rezolvare nu s-ar încadra în timp pentru testele mari.

Să vedem cum se poate îmbunătăți această soluție.

Se observă că, dacă sunt mai multe schimbări, prima sumă schimbată este negativă. De ce? Dacă ar fi pozitivă, o putem aduna la următoarea sumă schimbată; în acest mod facem o schimbare în loc de două, economisim taxa T și obținem un profit mai mare, deoarece cursul valutar crește în timp.

Pe baza acestei observații putem împărți cele N valori în intervale. Astfel, primul interval conține primele K elemente din șir, astfel încât suma primelor K elemente să fie negativă, dar suma primelor j elemente să fie cel puțin 0 pentru orice $j < K$. Dacă un astfel de K nu există, se va efectua o singură schimbare.

Următorul interval este ales în același mod, luând în considerare numai elementele de după primul interval și așa mai departe. Se observă că toate sumele intervalelor, exceptând, eventual, ultimul interval, sunt negative; dacă una dintre sume ar fi pozitivă, am concatena intervalul respectiv cu intervalul următor, obținând o soluție mai bună.

În acest moment putem aplica metoda *programării dinamice*. Fiecare schimbare va fi efectuată exact după unul din intervale. Varianta descrisă mai sus, în care am înlocui elementele cu intervale, ar avea ordinul de complexitate $O(NR^2)$, unde NR este numărul intervalelor. Pe cazul cel mai defavorabil, se obține tot $O(N^2)$

Putem îmbunătăți această soluție.

Se observă că este preferabil ca numărul de intervale consecutive dintre două schimbări să fie mic.

Fiecare schimbare, cel mult exceptând-o pe ultima, aduce o pierdere.

Presupunem că la o anumită schimbare folosim K intervale. Dacă am "sparge" această schimbare în una formată din primele J din cele K intervale și una formată din celelalte $K - J$ intervale, ar exista o penalizare suplimentară T, dar și o scădere a pierderii, deoarece pierderea dată de primele J intervale ar fi micșorată, datorită schimbării la un curs mai mic. Pentru K suficient de mare, scăderea pierderii este semnificativ mai valoroasă decât penalizarea suplimentară T.

O limită superioară pentru K este $2 \cdot \sqrt{T}$. Dacă schimbarea ar avea mai mult de $2 \cdot \sqrt{T}$ intervale, atunci am putea-o "sparge" după $\sqrt{T}$ intervale. Aceasta ar aduce o penalizare T, dar și o scădere a pierderii. Scăderea ar fi dată de suma valorilor absolute ale pierderilor pe cele $\sqrt{T}$ intervale (egală cu cel puțin $\sqrt{T}$, deoarece pe fiecare interval pierderea este de cel puțin -1), înmulțită cu $K - \sqrt{T}$ (datorită diferenței între cursurile la care se face schimbarea). Evident, această scădere a pierderii este mai importantă decât penalizarea T.

Aceasta conduce la o îmbunătățire a algoritmului de programare dinamică. Pentru calculul $C[i]$ (câștigul după primele i intervale) se iau în considerare valorile $C[j]$, cu $j - \sqrt{T} < j < i$. Ordinul de complexitate este deci $O(NR \cdot \sqrt{T})$. Acest algoritm ar fi obținut punctajul maxim.

Analiza complexității

Ordinul de complexitate al operației de citire a datelor de intrare este $O(N)$.

Determinarea intervalelor descrise mai sus are ordinul de complexitate $O(N)$.

Calculul valorilor $C[i]$ are ordinuul de complexitate $O(N \cdot \sqrt{T})$, deoarece numărul de intervale este limitat superior de N.

Ordinul de complexitate al operației de scriere a rezultatelor este $O(1)$.

În final, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este $\mathbf{O(N \cdot \sqrt{T})}$.

P060318: Nava spațială

Aceasta a fost cea mai simplă problemă din concurs.

Pentru rezolvare, putem porni invers, de la ultimul zbor spre primul.

O echipă de N oameni poate să conțină cel mult unul din ultimii S oameni de știință care au părăsit Pământul, cel mult doi din ultimii $2 \cdot S$ oameni etc.

Construim o matrice A, unde $A[i, j]$ reprezintă numărul de echipe posibile de j oameni de știință care au fost aduse cu ultimele i zboruri.

Evident, $A[1, 1] = S$ și $A[i, j] = 0$ pentru $j > i$.

Pentru $i > 1$, să analizăm cum se poate forma o echipă cu j oameni după i zboruri. O astfel de echipă poate conține cel mult P oameni din cel mai puțin recent zbor, cărora li se vor alătura $j - P$ oameni din celelalte zboruri. Evident $P \le j$ și $P \le S$.

Cei P oameni se pot alege în C_S^P moduri. Pentru fiecare astfel de alegere, ceilalți $j - P$ oameni se pot alege în $A[i - 1, j - P]$ moduri. Deci pentru P fixat avem $C_S^P \cdot A[i - 1, j - P]$ moduri.

Modalitățile pentru $P = 0$, $P = 1$, $P = 2$ etc. se adună, obținându-se $A[i, j]$. Se folosesc calcule cu numere mari.

Deoarece sunt maxim 400 de posibilități de date de intrare, valorile pentru fiecare intrare posibilă pot fi scrise într-o tabelă de constante, într-un alt program. Se obține astfel un program care citește datele de intrare și afișează direct rezultatul.

Analiza complexității

Ordinul de complexitate al operației de citire a datelor de intrare și cel al operației de scriere a rezultatelor este $O(1)$.

Programul dat spre evaluare constă în citirea datelor și afișarea rezultatelor. Deoarece numărul de cifre al rezultatului este limitat superior de o constantă, ordinul de complexitate al rezolvării este $O(1)$.

De-ale programatorilor...

Orice bug suficient de avansat nu poate fi deosebit de o facilitate.

Dacă un program este folositor, atunci va trebui modificat cu siguranță.

Dacă un program este complet inutil, atunci va trebui documentat.

Programatorii adevărați nu își comentează niciodată codul sursă. Datorită faptului că a fost depus un efort deosebit pentru a scrie codul, este normal să trebuiască un efort și mai mare pentru a-l înțelege.

Hardware-ul este un produs care se strică dacă te "joci" mult cu el. Software-ul este un produs care funcționează numai după ce te "joci" mult cu el.

Viața ar fi mult mai ușoară dacă am putea să ne uităm peste codul sursă.

Cea mai bună modalitate de a mări viteza unui calculator este folosirea unei accelerații de 9,8 m/s².

Orice femeie este de nota 10. Diferența o face baza în care este scris acest număr.

Acronime

MS-DOS = Maybe SomeDay an Operating System (poate, într-o zi, un sistem de operare)

MACINTOSH = Machine Always Crashes - If Not The Operating System Hangs (mașina se blochează întotdeauna, dacă nu, cade sistemul de operare)

IBM = It's Better Manually (este mai bine de mână)

MIPS = Meaningless Indicator of Processor Speed (indicator fără sens al vitezei procesorului)

Depanare în C

- dacă este posibil, nici măcar nu încerca; lasă pe altcineva
- introdu linii vide în poziții aleatoare, recompilează și execută
- aruncă cu apă sfințită pe tastatură
- sună la 112 și țipă
- rescrie programul în limbaj de asamblare; aceasta nu va rezolva problemele dar, cel puțin, vei fi sigur că nimeni altcineva nu le va descoperi și nu te va putea face nimeni să te simți prost