Faza JUDEȚEANĂ a olimpiadei de INFORMATICĂ pentru LICEE

Mihai Stroe

În perioada 28 - 29 februarie a avut loc faza județeană a olimpiadei de informatică pentru licee. În continuare vă vom prezenta soluțiile problemelor propuse spre rezolvare la acest concurs.

Clasa a IX-a

P030419: Expresie

Rezolvarea problemei constă în descompunerea în factori primi a numerelor date și prelucrarea acestor factori.

Concret, se poate păstra un vector P de la 1 la 30000, în care P_i este 0 dacă i nu este prim, respectiv puterea la care apare i în descompunerea în factori primi a produsului dacă i este prim.

Valorile din P nu pot depăși $n \cdot \log_2 30000$, deci vor fi mai mici sau egale cu $5000 \cdot 14 = 70.000$; astfel, elementele vectorului vor fi de tip *longint*.

Fiecare din cele n numere este descompus în factori primi. În momentul în care se determină un factor prim F al unui număr din cele date, se incrementează P_F cu puterea la care F apare în descompunerea numărului respectiv. Nu este necesară memorarea separată a descompunerii fiecărui număr.

În final, pentru fiecare element din P se verifică dacă este multiplu de m. Dacă toate elementele îndeplinesc această condiție, expresia este un număr natural și se trece la afișare.

Se poate renunța la vectorul de 30000 de elemente, păstrându-se în locul acestuia un vector în care se memorează numerele prime mai mici decât 30000 și un vector care arată la ce puteri apar aceste numere în descompunere. Această abordare introduce în plus operații pentru a găsi indicele unui anumit număr prim; se poate folosi cu succes căutarea binară. Pe de altă parte, la descompunerea în factori primi se va testa numai împărțirea prin numere prime.

Analiza complexității

Descompunerea unui număr X în factori primi are ordinul de complexitate $O(\sqrt{x})$, dacă nu se folosește lista de numere prime.

Pasul de descompunere și completare a vectorului P are deci ordinul de complexitate $O(n \cdot \sqrt{30000})$.

Citirea datelor, verificarea dacă expresia este un număr natural și afișarea au ordinul de complexitate $O(n)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(n)$, dacă facem abstracție de constanta $\sqrt{30000}$.

P030420: Reactivi

Problema se poate rezolva prin metoda *greedy*.

O variantă mai explicită a enunțului este următoarea:

"*Se consideră N intervale pe o axă. Să se aleagă un număr minim de puncte astfel încât fiecare interval să conțină cel puțin unul dintre punctele alese.*"

Facem o primă observație: pentru orice soluție optimă există o soluție cu același număr de puncte (frigidere), în care fiecare punct să fie sfârșitul unui interval. Aceasta se poate obține mutând fiecare punct spre dreapta, până când ar ajunge la sfârșitul intervalului care se termină cel mai repede, dintre intervalele care îl conțin. Se observă că noua soluție respectă restricțiile din enunț.

În continuare ne vom concentra pe găsirea unei soluții de acest tip.

Sortăm reactivii după sfârșitul intervalului. Pentru intervalul care se termină cel mai repede, alegem ultimul punct al său ca temperatură a unui frigider. Se observă că această alegere este cea mai bună, dintre toate alegerile unui punct în intervalul respectiv, în sensul că mulțimea intervalelor care conțin punctul este mai mare (conform relației de incluziune), decât mulțimea corespunzătoare oricărei alte alegeri. Acest fapt este adevărat, deoarece mutarea punctului mai la stânga nu duce la selectarea unor noi intervale.

Intervalele care conțin punctul respectiv sunt eliminate (reactivii corespunzători pot fi plasați într-un frigider), iar procesul continuă cu intervalele rămase, în același mod.

Analiza complexității

Notăm cu D numărul de temperaturi întregi din intervalul care conține temperaturile din enunț. Se observă că D este cel mult 201.

Citirea datelor de intrare are ordinul de complexitate $O(N)$.

Sortarea intervalelor după capătul din dreapta are ordinul de complexitate $O(N \cdot \log N)$.

Urmează F pași, unde F este numărul de frigidere selectate. Deoarece fiecare frigider este setat la o temperatură întreagă, $F \leq D$.

În cadrul unui pas, determinarea intervalului care se termină cel mai repede, pe baza vectorului sortat, are ordinul de complexitate $O(1)$. Eliminarea intervalelor care conțin un anumit punct (sfârșitul intervalului care se termină cel mai repede) are ordinul de complexitate $O(N)$.

Afișarea rezultatului are ordinul de complexitate $O(1)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(N \cdot D + N \cdot \log N)$; deoarece în general $D > \log N$, considerăm ordinul de complexitate ca fiind $\boldsymbol{O(N \cdot D)}$.

Clasa a X-a

P030421: Perle

Inițial, problema pare dificilă, dar, după examinarea transformărilor posibile, se observă că este destul de simplă.

Fie un anumit șir de perle. Presupunem că acesta a fost obținut dintr-o perlă magică și ne oprim la prima contradicție.

Să determinăm, la început, perla magică din care s-ar putea obține șirul.

Dacă șirul are lungimea 1, el se poate obține dintr-o perlă magică A.

Dacă șirul începe cu `12` și are trei perle, s-ar putea obține dintr-o perlă C.

Dacă șirul începe cu `2` și are cel puțin două perle, singura șansă de a-l obține este de a începe cu o perlă magica de tip B. Similar dacă începe cu 1 și are mai mult de trei perle.

Dacă șirul începe cu `3`, s-ar putea obține numai dintr-o perlă de tip C.

Acum, având în vedere că am stabilit singura perlă magică din care sunt șanse să se obțină șirul, vom verifica dacă sirul poate fi într-adevăr obținut.

Pentru aceasta, putem scrie câte o funcție pentru fiecare perlă magică. O astfel de funcție se aplică pe șirul de intrare și execută operații (avansări pe șir sau apeluri ale funcțiilor pentru alte tipuri de perle) în funcție de tipul perlelor întâlnite.

Dacă la un moment dat nu există nici o regulă de continuare (de exemplu, pentru șirul `22`), șirul nu poate fi obținut.

Funcția pentru A nu este necesară, tratarea perlei magice A putând fi inclusă în celelalte două funcții.

Se observă că, în cadrul fiecărei funcții, acțiunile efectuate sunt alese determinist. Astfel, la fiecare pas se alege o acțiune posibilă sau se întrerupe căutarea și se semnalează faptul că șirul nu poate fi obținut.

Funcțiile se apelează recursiv. Pentru a evita depășirea stivei, recursivitatea se poate simula iterativ.

Analiza complexității

La fiecare moment, deciziile se iau în timp constant.
Ordinul de complexitate al operației de citire a datelor de intrare este $O(L)$, unde L reprezintă suma lungimilor șirurilor de perle din fișierul de intrare.

Fiecare caracter din fiecare șir este parcurs o singură dată.

Ordinul de complexitate al algoritmului este $O(L_i)$ pentru un șir de perle de lungime L_i.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $\boldsymbol{O(L)}$.

P030422: Romeo și Julieta

Problema se rezolvă folosind *algoritmul lui Lee*.

Se aplică acest algoritm folosind ca puncte de start poziția lui *Romeo* și poziția *Julietei*.

Vom folosi o matrice D în care vom pune valoarea 1 peste tot pe unde nu se poate trece, valoarea 2 în poziția în care se află *Romeo* inițial, valoarea 3 în poziția în care se află *Julieta* inițial și valoarea `0` în rest.

La fiecare pas k vom parcurge această matrice și în pozițiile vecine celor care au valoarea 2 vom pune valoarea 2, dacă acestea au valoarea `0` sau 3. Dacă o poziție are valoare 3, înseamnă că la un moment de timp anterior *Julieta* se putea afla în poziția respectivă. La același pas k vom mai parcurge matricea o dată și în pozițiile vecine celor care au valoarea 3 vom pune valoarea 3, dacă acestea au valoarea `0`.

Dacă la pasul k poziția vecină uneia care are valoarea 3, are valoarea 2, atunci ne vom opri și k reprezintă momentul minim de timp după care ce-i doi se întâlnesc, iar poziția care are valoare 2 reprezintă locul întâlnirii.

La prima vedere s-ar părea că numărul k nu reprezintă momentul de timp minim la care cei doi se întâlnesc. Vom demonstra că algoritmul este corect prin metoda reducerii la absurd. Pentru aceasta avem în vedere că pozițiile marcate cu 2 reprezintă toate locurile în care se poate afla *Romeo* după cel mult k pași, iar cele marcate cu 2 reprezintă toate locurile în care se poate afla *Julieta* după cel mult k pași. Dacă k nu reprezintă momentul de timp minim la care cei doi se întâlnesc înseamnă că acesta a fost determinat mai devreme și algoritmul s-a oprit deja.

Analiza complexității

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

Ordinul de complexitate al acestui algoritm este $O(k \cdot M \cdot N)$, unde k reprezintă momentul în care cei doi se întâlnesc.

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

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $\boldsymbol{O(k \cdot M \cdot N)}$.

Clasele a XI-a și a XII-a

P030423: Moșia lui Păcală

Această problemă se poate împărți în trei subprobleme mai mici.

Prima subproblemă constă în ordonarea punctelor în sens trigonometric sau anti-trigonometric pentru ca șirul lor să descrie un poligon.

Pentru aceasta vom alege cel mai de jos punct și îl vom pune pe prima poziție, iar apoi vom sorta celelalte puncte în funcție de unghiurile formate de segmentul care are ca și capete punctul ales la început și un punct dintre cele nesortate și dreapta orizontală care trece prin punctul ales la început.

După ce prima subproblemă a fost rezolvată se poate trece la rezolvarea celei de-a doua subprobleme.

A doua subproblemă constă în a determina surplusul de suprafață pe care îl aduce mutarea fiecăruia dintre pari și acesta este egal cu produsul dintre distanța maximă pe care poate fi mutat un par și distanța dintre cei doi pari vecini ai săi împărțită la 2.

La acest rezultat se ajunge prin câteva calcule simple din geometrie ținându-se cont de faptul că dacă mutăm un par, nu vom mai putea muta vecinii săi datorită restricțiilor impuse în enunț. Aceste surplusuri (costuri) sunt reținute într-un vector.

A treia subproblemă constă în a determina suprafața maximă cu care poate fi mărită moșia ținând cont de restricții.

Această subproblemă se rezolvă prin metoda programării dinamice și este asemănătoare problemei "*Oo*" care a fost propusă spre rezolvare la concursul de programare *Stelele Informaticii* al cărei enunț poate fi găsit în *GInfo* 8/2003, iar rezolvarea în *GInfo* 1/2004.

Diferența dintre aceste două probleme este dată de faptul că la problema *Oo*, fermierul *Ion* poate aduna ouăle din două sectoare și nu mai poate aduna ouăle din sectoarele vecine celor din care a adunat.

În cazul acestei subprobleme va trebui să construim un tablou unidimensional A cu N elemente. Elementul de pe o poziție i indică suprafața maximă cu care poate fi mărită moșia prin mutarea parului i.

Soluția este dată de cea mai mare valoare din acest vector.

Analiza complexității

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

Ordinul de complexitate a primei subprobleme este dat de ordinul de complexitate al operației de sortare utilizate și este $O(N \cdot \log N)$, dacă se folosește sortarea rapidă.

Ordinul de complexitatea a celei de-a doua și a treia subprobleme este $O(N)$.

Operația de scriere a rezultatelor se realizează în $O(1)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $\boldsymbol{O(N \cdot \log N)}$.

P030424: Lanterna

Se cere tipul de lanternă cu consumul cel mai mic, dintre cele cu care se poate ajunge la destinație în timp minim.

Cerința sugerează ideea folosirii căutării binare. Astfel, inițial se încearcă ajungerea la destinație cu $W_{min} = K$. Dacă se reușește ajungerea la destinație cu timpul minim, se va încerca cu un W_{min} mai mic; dacă nu se reușește, sau timpul obținut este mai mare decât timpul minim, se va încerca un W_{min} mai mare. Evident, timpul minim T_{min} este obținut pentru $W_{min} = K$.

Acestea fiind stabilite, avem de rezolvat următoarea subproblemă: *cum determinăm timpul minim în care putem ajunge la destinație pentru un anumit* W_{min}?

Construim o matrice A cu N linii și $W_{min} + 1$ coloane (de la 0 la W_{min}), în care $A_{i,j}$ semnifică timpul minim în care se poate ajunge din punctul de plecare în obiectivul i, având încă j *Wați* disponibili în lanternă pentru a continua mai departe.

Regulile de completare sunt următoarele:

- dacă i este un obiectiv prieten, de fiecare dată se va completa numai $A_{i,W_{min}}$, deoarece lanterna poate fi reîncărcată;
- dacă am completat $A_{i,j}$ cu o valoare V, atunci pentru orice drum de la i la un alt obiectiv p, de lungime L și consum C, am putea pune valoarea $V + L$ în $A_{p,j+C}$ dacă nu există deja o valoare mai mică. Evident, dacă $j + C > W_{min}$ această opțiune nu este luată în considerare. Dacă p este un obiectiv prieten, se folosește regula de mai sus.

Pentru a nu suprascrie prea des valori, vom completa elementele matricei A în ordinea timpului în care se poate ajunge în orașe. Pentru aceasta folosim o coadă de priorități, care va conține elemente de tipul (*obiectiv*, *wațiRămași*, *timpDeAjungere*), sortată după *timpDeAjungere*. Se extrage repetat elementul cu timpul de ajungere minim și, folosind regulile prezentate, se adaugă noi elemente în coadă, dacă este cazul, completându-se și pozițiile corespunzătoare din matricea A.

Putem implementa coada de priorități cu ajutorul unui *heap*.

Dacă dorim să nu suprascriem deloc valori, combinăm acest algoritm cu algoritmul lui *Dijkstra*. Lăsăm această îmbunătățire ca exercițiu pentru cititor.

În final se vor afișa valorile cerute.

Analiza complexității

Se execută $\log K$ pași.

La fiecare pas se completează $O(W_{min} \cdot N)$ poziții; pentru fiecare completare a unei pozitii se execută $O(\log W_{min} \cdot N)$ operații. Pentru acest calcul, considerăm împreună operațiile de introducere, respectiv extragere ale unui anumit element în/din coada de priorități.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $\boldsymbol{O(\log K \cdot \log(W_{min} \cdot N) \cdot (W_{min} \cdot N))}$.

În practică, nu toate pozițiile din matrice sunt completate, ceea ce face ca algoritmul să ruleze foarte repede.