Etapa FINALĂ

Vă prezentăm soluțiile oficiale ale celor cinci probleme propuse spre rezolvare la etapa finală a celei de-a treia ediții a concursului internațional de programare Bursele Agora, organizat de redacția GInfo și editura Agora Media.

P060219: Conducte

Putem privi sistemul de rafinării și benzinării ca fiind un graf neorientat complet ale cărui noduri reprezintă obiectivele (benzinării sau rafinării), iar costurile muchiilor sunt date de distanțele (lungimile conductelor) care leagă două obiective. Evident, nu putem uni printr-o conductă două rafinării, așadar vom considera că muchiile dintre rafinării au costul 0 (nu sunt necesare conducte pentru ca benzina să ajungă de la o rafinărie la alta deoarece există deja benzină în toate rafinăriile).

Acum, problema se reduce la a determina un arbore parțial de cost minim în graful construit. Pentru aceasta vom folosi ***algoritmul lui Prim*** (vezi *CLR*[1], p. 435); în final vom afișa muchiile care fac parte din arborele parțial minim dar nu au ca extremități două noduri care corespund unor rafinării.

Datorită faptului că putem avea până la 1000 de benzinării nu vom putea reprezenta în memorie graful folosind o matrice de adiacență. De asemenea, nu este posibilă păstrarea unei liste de muchii deoarece numărul acestora este prea mare. De aceea, vom păstra doar coordonatele benzinăriilor și rafinăriilor și vom calcula costurile muchiilor (distanțele dintre obiective) în momentul în care avem nevoie de aceste informații. Din acest motiv, nu poate fi folosit ***algoritmul lui Kruskal*** de determinare a arborelui parțial minim pentru că acesta implică sortarea tuturor muchiilor, deci păstrarea acestora în memorie.

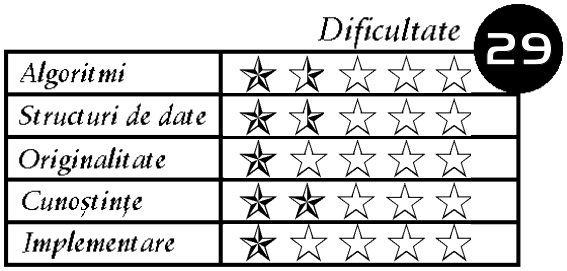
Analiza complexității

Datele de intrare constau în coordonatele rafinăriilor și benzinăriilor, deci operația de citire a acestora are ordinul de complexitate $O(n) + O(k) = \boldsymbol{O(n+k)}$.

Este obligatoriu ca nodurile corespunzătoare rafinăriilor să fie legate între ele în arborele parțial minim folosind muchii de cost 0, operație care se realizează în timp liniar (ordinul de complexitate este $O(k)$). Pentru celelalte noduri vom folosi *algoritmul lui Prim*, ordinul de complexitate al acestuia fiind $O(n^2)$. Așadar, determinarea arborelui parțial minim are ordinul de complexitate $O(n^2)+ O(k) = \boldsymbol{O(n^2 + k)}$.

Datele de ieșire constau în cele n muchii care leagă benzinăriile între ele sau de rafinării, deci ordinul de complexitate al operației de scriere a acestora este $\boldsymbol{O(n)}$.

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate $O(n + k) + O(n^2 + k) + O(n) = \boldsymbol{O(n^2 + k)}$.



http://www.ginfo.ro/revista/12_7/surse/conducte.cpp

P060220: Dune

Această problemă este, de fapt, o "schimbare la față" a problemei *Matrice* de la prima rundă a ediției a treia a concursului de programare *Bursele Agora*. Puteți găsi o scurtă prezentare a soluției în numărul 12/1 (ianuarie 2002) al *GInfo* (www.ginfo.ro/revista/12_1). Vom prezenta din nou soluția ținând cont de noul enunț.

Pentru rezolvarea problemei vom folosi metoda programării dinamice. Considerăm matricea de intrare X. Se observă că există cel mult 64 de modalități de a plasa bombe (mai exact colțurile din stânga-sus ale secțiunilor pătratice afectate de acestea) pe o linie a matricei. Valoarea 64 se obține datorită faptului că secțiunile trebuie să fie incluse integral în matrice (bombele nu pot atinge zone din afara *Câmpiei Arakeenului*), deci colțul din stânga-sus al unui pătrat poate fi plasat numai pe una dintre primele șase coloane. De fapt, acest număr poate fi redus la 37, deoarece unele posibilități de plasare sunt inutile.

De exemplu, se observă că patru bombe sunt suficiente pentru a acoperi integral o linie, deci nu are sens să plasăm cinci bombe pe o linie, deoarece cel puțin una dintre ele va

1. *Cormen T. H., Leiserson C. E., Rivest R. L., Introducere în Algortmi, Computer Libris Agora, Cluj-Napoca, 2000*

afecta poziții atinse și de alte bombe, deci bomba poate fi folosită în alte zone, valoarea diferenței dintre numărul *harkonnenilor* și cel al *atreizilor* uciși rămânând aceeași. De asemenea, nu are rost să plasăm colțurile din stânga-sus a zonelor afectate de trei bombe în trei coloane consecutive, deoarece pozițiile afectate de bomba din mijloc, sunt afectate de celelalte două bombe.

Presupunem că știm, pentru o anumită linie, sumele maxime care pot fi obținute dacă plasăm 0, 1, 2, ..., K bombe pe primele i linii, folosind pe linia i o anumită configurație. Apare întrebarea dacă putem determina aceleași informații pentru linia $i + 1$.

Răspunsul este afirmativ și ne oferă rezolvarea problemei. Construim (cel puțin teoretic) tabloul tridimensional $M[1 .. N - 1, 0 .. K, 1 .. 37]$. Un element $M[i, j, l]$ reprezintă suma maximă obținută dacă plasăm j pătrate pe primele i linii, folosind configurația l pe linia i.

Elementele matricei $M[i + 1]$ pot fi determinate folosind matricea $M[i]$. Pentru a calcula un element al matricei $M[i + 1]$ avem nevoie de următoarele informații:

- suma valorilor de pe linia $i + 2$ corespunzătoare unor zone afectate de bombe care vor fi plasate pe linia $i + 1$, folosind configurația l;
- numărul $nr[l]$ al bombelor care sunt folosite pentru obținerea configurației l;
- suma valorilor de pe linia $i + 1$ care corespund unor zone afectate dacă se folosește configurația l, dar nu și folosind o configurație c (determinăm valorile sumei pentru orice configurație c);
- valoarea maximă din tabloul M pentru fiecare configurație c plasată pe linia i care conține $j - nr[l]$ pătrate.

Programul este construit astfel:

- configurațiile posibile de pe o linie sunt păstrate în matricea de constante p, iar numărul de pătrate folosit pentru obținerea fiecărei configurații este păstrat în vectorul nr;
- tabloul tridimensional M este simulat cu ajutorul a două matrice A și B (adică $A = M[i]$, se calculează $B = M[i + 1]$, apoi A devine B etc.)
- pentru fiecare rând unde se pot plasa bombe:
  - se calculează suma adăugată pe rândul respectiv dacă plasăm o anumită configurație, în funcție de configurația plasată pe rândul anterior;
  - se calculează suma adăugată pe rândul următor dacă plasăm o anumită configurație pe rândul curent;
  - se inițializează matricea B;
  - se realizează calculul valorilor din B (datorită faptului că operațiile din această zonă sunt executate de foarte multe ori, codul trebuie optimizat cât mai mult posibil;
  - A devine B.

Soluția este dată de elementul maxim obținut în final în matricea A.

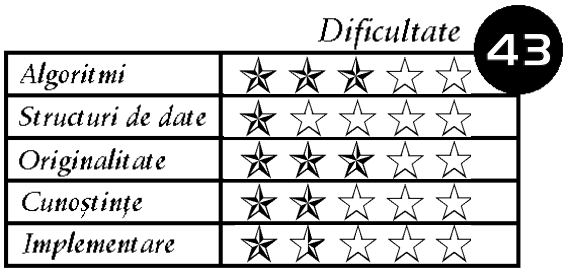
Analiza complexității

Citirea unei linii a matricei care reprezintă *Câmpia Arakeenului* se realizează în timp constant deoarece matricea are întotdeauna șapte coloane. Datorită faptului că trebuie citite n linii, ordinul de complexitate al operației de citire este $O(1) \cdot O(n) = O(n)$.

Matricele A și B au, fiecare, k linii și 37 de coloane. Operația de determinare a elementelor matricei B, folosind matricea A, are ordinul de complexitate $O(k)$, deoarece numărul operațiilor efectuate pentru determinarea uneia dintre linii este $O(37) \cdot O(37) \cdot O(7) = 37 \cdot O(1) \cdot 37 \cdot O(1) \cdot 7 \cdot O(1) = 9583 \cdot O(1) = O(1)$. Pentru fiecare dintre cele n linii ale matricei care caracterizează câmpia, se vor calcula elementele matricei B, deci ordinul de complexitate a operației de determinare a valorilor finale ale elementelor matricei B este $O(n) \cdot O(k) = O(n \cdot k)$.

Pentru afișarea diferenței maxime trebuie parcursă matricea finală, deci această operație are ordinul de complexitate $O(k)$.

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate $O(n) + O(n \cdot k) + O(k) = O(n \cdot k)$.



http://www.ginfo.ro/revista/12_7/surse/dune.cpp

P060221: Frăția Inelului

Vom construi un graf orientat ale cărui noduri reprezintă reperele și ale cărui arce reprezintă drumurile care leagă două repere. Sensul unui arc va fi de la reperul de la care pornește drumul la reperul la care ajunge drumul. Datorită faptului că nu se poate ajunge de la un reper la unul anterior, deducem că graful este aciclic.

Acum, problema se reduce la determinarea numărului de drumuri din acest graf de la nodul care reprezintă reperul `VALCEAUADESPICATA` la nodul care reprezintă reperul `MUNTELEOSANDEI`.

Pentru simplitate vom codifica reperele prin numere cuprinse între 0 și $N - 1$, unde N reprezintă numărul total al reperelor.

Pentru a determina numărul drumurilor dintre două noduri ale unui graf orientat vom folosi metoda programării dinamice. Vom construi un șir care reprezintă numărul de posibilități de a ajunge la un anumit reper (numărul de drumuri care pornesc din `VALCEAUADESPICATA` și ajung la reperul respectiv).

Pentru un anumit nod vom putea determina numărul de posibilități dacă se cunoaște numărul de posibilități în care se poate ajunge la fiecare dintre predecesorii săi din graf. Dacă toate aceste informații sunt cunoscute, atunci numărul posibilităților de a ajunge la acest reper va fi dat de suma numerelor posibilităților de a ajunge la reperele corespunzătoare predecesorilor.

Inițial considerăm că există o singură posibilitate de a ajunge la VALCEAUADESPICATA. Datorită aciclicității grafului suntem siguri că, la fiecare pas, vom determina cel puțin un nod pentru care cunoaștem valorile corespunzătoare tuturor predecesorilor. Așadar, la fiecare pas, vom verifica pentru toate nodurile pentru care nu cunoaștem numărul posibilităților, dacă sunt cunoscute numerele posibilităților corespunzătoare predecesorilor. Pentru fiecare nod pentru care sunt cunoscute aceste informații vom calcula valoarea corespunzătoare. Datorită faptului că la fiecare pas vom determina cel puțin un astfel de nod, suntem siguri că algoritmul se va termina după un număr finit de pași (cel mult N - 1).

În final, vom afișa valoarea corespunzătoare nodului care reprezintă MUNTELEOSANDEI.

Analiza complexității

Datele de intrare constau în drumurile dintre repere, deci ordinul de complexitate al operației de citire a acestora este $O(M)$.

Pentru fiecare reper citit se determină codul acestuia, deci ordinul de complexitate al operației de determinare a codurilor este $O(M) \cdot O(N) = O(M \cdot N)$ datorită faptului că acest cod este căutat într-un șir care conține cel mult N elemente.

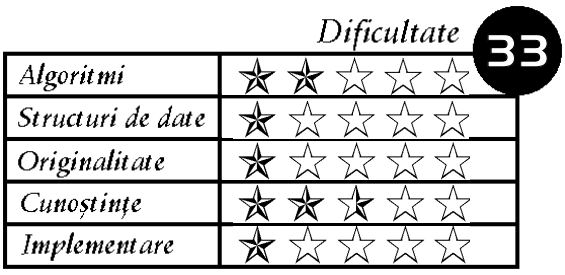
Crearea listelor predecesorilor se realizează pe parcursul citirii datelor de intrare, operația având ordinul de complexitate $O(M)$.

Stabilirea faptului că există o posibilitate de a se ajunge în VALCEAUADESPICATA implică determinarea codului acestui reper, deci are ordinul de complexitate $O(N)$.

La fiecare pas al algoritmului vom verifica, pentru fiecare nod, dacă au fost determinate informațiile pentru predecesori. Pentru că un nod poate avea, teoretic, până la N - 1 predecesori, operațiile efectuate la fiecare pas au ordinul de complexitate $O(N) \cdot O(N) = O(N^2)$. Deoarece sunt efectuați cel mult N - 1 pași, operația de determinare a numărului de posibilități de a ajunge la MUNTELEOSANDEI are ordinul de complexitate $O(N) \cdot O(N^2) = O(N^3)$.

Scrierea datelor de ieșire implică determinarea codului pentru MUNTELEOSANDEI, deci are ordinul de complexitate $O(N)$.

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate $O(M) + O(M \cdot N) + O(M) + O(N) + O(N^3) + O(N) = O(N^3 + M \cdot N) = O(N \cdot (N^2 + M))$.



http://www.ginfo.ro/revista/12_7/surse/lords.cpp

P060222: Litere

Problema se reduce la determinarea unui cuvânt care a fost codificat folosind transformarea ***Burrows-Wheeler*** dacă se cunoaște rezultatul codificării. Modul în care se realizează codificarea este prezentat în enunțul problemei.

Decodificarea cuvântului se realizează folosind următorul algoritm:

- se determină frecvența de apariție a literelor, ceea ce permite obținerea primei coloane a matricei (prima coloană conține întotdeauna literele ordonate lexicografic).
- se construiește un șir de corespondențe A între literele de pe prima coloană și cele de pe ultima, ținând cont de faptul că celei de-a i-a apariții a caracterului c de pe prima coloană îi corespunde cea de-a i-a apariție a caracterului c de pe ultima.
- pentru determinarea cuvântului se scriu literele corespunzătoare pozițiilor $A[1]$, $A[A[1]]$, $A[A[A[1]]]$ etc; pentru aceasta se folosește un indice i care, după fiecare pas va primi valoarea $A[i]$.

Acest algoritm a fost demonstrat matematic de către cei doi cercetători a căror nume a fost dat acestei metode.

Analiza complexității

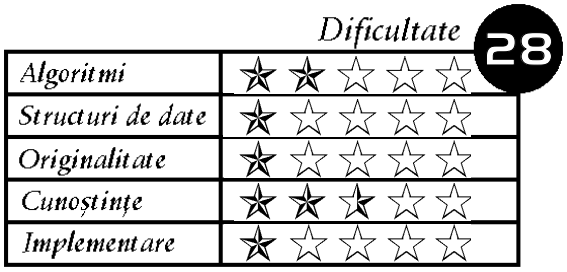
Datele de intrare constau în citirea șirului literelor de pe ultima coloană; deoarece avem N litere, ordinul de complexitate al operației este $O(N)$.

Determinarea frecvențelor aparițiilor literelor se realizează în timp *liniar* prin simpla parcurgere a șirului care conține literele.

Construirea șirului de corespondențe se realizează tot în timp *liniar* deoarece la fiecare pas este realizată câte o corespondență.

Pentru afișarea datelor de ieșire (a cuvântului căutat) se realizează o simplă parcurgere a șirului de corespondențe, deci această operație are ordinul de complexitate $O(N)$.

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate $O(N) + O(N) + O(N) + O(N) = O(N)$.



http://www.ginfo.ro/revista/12_7/surse/litere.cpp

P060223: Zăvor

În cele ce urmează vom folosi următoarele denumiri pentru a ne referi la anumite stări ale automatului:

- *stare inițială* (stare de bază) - starea în care se află automatul la început sau după resetare (prima stare);
- *stare finală* (stare de deschidere) - starea în care se află automatul atunci când se deschide cifrul;

- *stare intermediară* - una dintre stările în care se află automatul pe parcursul introducerii parolei corecte (una dintre cele N - 1 stări corespunzătoare primelor N - 1 cifre ale parolei);
- *stare de eroare* - starea în care se ajunge dacă se recepționează un semnal incorect (a (N + 2) - a stare a automatului).

Vom descrie în continuare cele patru situații în care putem detecta că cipul nu funcționează corect:
- cipul poate fi deschis și în una dintre stările intermediare, nu numai în starea finală;
- există posibilitatea de a "sări" peste anumite stări intermediare;
- există posibilitatea de a ajunge într-o anumită stare în mai multe moduri;
- există posibilitatea de a reveni din starea de eroare în una dintre celelalte stări.

În cele ce urmează vom explica pe larg caracteristicile fiecărei situații și vom arăta modul în care poate fi găsită o modalitate alternativă de a deschide cifrul dacă a fost detectată o astfel de situație.

Pentru prima situație vom folosi cifrele din parola corectă și, la fiecare pas, vom verifica dacă una dintre stările intermediare i este stare de deschidere.

Parola alternativă va fi formată din cifrele de pe pozițiile 1, 2, ..., i ale parolei corecte.

Pentru a doua situație vom lua în considerare toate perechile i, j ($i < j$) și vom verifica dacă cifrul se deschide folosind cifrele de pe pozițiile 1, 2, ..., i, j + 1, ..., n, unde n este lungimea parolei corecte.

Parola alternativă va fi formată din cifrele de pe aceste poziții.

Pentru a treia situație vom lua în considerare toate tripletele i, j, k și vom verifica dacă din starea i se poate ajunge în starea j folosind semnalul k, unde semnalul k nu este cifra care duce la trecerea din starea i în starea i + 1.

Parola alternativă va fi formată din cifrele de pe pozițiile 1, 2, ..., i ale parolei corecte, semnalul k și cifrele de pe pozițiile j, j + 1, ..., n ale parolei corecte.

Pentru a patra situație vom lua toate perechile i, j și vom verifica dacă din starea de eroare putem ajunge în starea i folosind semnalul j. Pentru a ajunge în starea de eroare vom "trimite" la început un semnal z diferit de prima cifră a parolei.

Parola alternativă va fi dată de semnalul z, semnalul j și cifrele de pe pozițiile i, i + 1, ..., n ale parolei corecte.

Dacă nu s-a detectat nici o parolă alternativă, aceasta este scrisă în fișierul de ieșire.

În caz contrar, în fișierul de ieșire se va scrie șirul de caractere `CORECT`.

Analiza complexității

Trebuie remarcat faptul că operațiile efectuate de bibliotecă se realizează în timp ***constant***.

Datele de intrare constau în cele n cifre ale parolei corecte, deci citirea acestora are ordinul de complexitate $O(n)$.

Pentru prima situație vom trece pe rând dintr-o stare în alta (pornim din starea inițială și parcurgem stările intermediare) și vom testa la fiecare pas dacă s-a ajuns într-o stare de deschidere. Așadar, ordinul de complexitate al acestei operații este $O(n)$.

Pentru cea de-a doua situație există $n \cdot (n - 1) / 2$ perechi (i, j) pe care le vom lua în considerare. Pentru fiecare astfel de pereche vom parcurge stările intermediare corespunzătoare (vor exista cel mult n - 1 astfel de stări), deci ordinul de complexitate al operațiilor efectuate pentru fiecare pereche este $O(n)$. Așadar, pentru cea de-a doua situație ordinul de complexitate al tuturor operațiilor este $O(n^2) \cdot O(n) = O(n^3)$.

Pentru cea de-a treia situație există n^2 perechi (i, j) și $9 \cdot n^2$ triplete (i, j, k). Pentru fiecare triplet vom parcurge stările intermediare corespunzătoare care vor fi în număr de cel mult $2 \cdot n + 1$, deci ordinul de complexitate al operațiilor efectuate pentru fiecare triplet este $O(n)$. Așadar, pentru cea de-a treia situație ordinul de complexitate al tuturor operațiilor este $O(9) \cdot O(n^2) \cdot O(\mathrm{n}) = 9 \cdot O(1) \cdot O(n^2) \cdot O(n) = O(1) \cdot O(n^2) \cdot O(n) = O(n^3)$.

Pentru cea de-a patra situație există $10 \cdot n$ perechi (i, j) și pentru fiecare dintre acestea vom parcurge stările intermediare corespunzătoare (cel mult n + 1 astfel de stări), deci ordinul de complexitate al operațiilor efectuate pentru fiecare pereche este $O(n)$. Așadar, pentru cea de-a patra situație ordinul de complexitate al tuturor operațiilor este $O(10) \cdot O(n) \cdot O(n) = 10 \cdot O(1) \cdot O(n) \cdot O(n) = O(1) \cdot O(n) \cdot O(n) = O(n^2)$.

În momentul în care detectăm o posibilitate alternativă de deschidere a cifrului, va trebui să construim noua parolă. Operația se realizează în timp ***liniar*** deoarece noua parolă va conține cel mult $2 \cdot n + 1$ cifre.

În final, vom scrie parola găsită sau cuvântul `CORECT`, operație realizabilă în timp ***liniar*** (dacă a fost găsită o parolă alternativă) sau ***constant*** (dacă nu a fost găsită o astfel de parolă).

În concluzie, algoritmul de rezolvare a acestei probleme are ordinul de complexitate $O(n) + O(n) + O(n^3) + O(n^3) + O(n^2) + O(n) + O(n) = O(n^3)$.

Dificultate 32	
Algoritmi	★★☆☆☆
Structuri de date	★☆☆☆☆
Originalitate	★★☆☆☆
Cunoștințe	★★★★☆
Implementare	★★☆☆☆

http://www.ginfo.ro/revista/12_7/surse/zavor.cpp