Bursele AGORA

A treia etapă a concursului de programare Bursele Agora s-a încheiat. În cadrul acestui număr vă vom prezenta soluțiile problemelor de la această rundă, clasamentul etapei, precum și clasamentul general după primele trei etape.

A treia rundă

Pentru prima dată la această ediție a concursului nostru, unul dintre concurenți a reușit să obțină toate cele 300 de puncte puse în joc la o rundă. Performanța a fost obținută de *Silvestru-Cosmin Negrușeri*, din *Bistrița*.

Concitadinul său, *Adrian Cârcu*, ar fi putut obține și el punctajul maxim dar, datorită unei erori banale, a trebuit să se mulțumească doar cu 200 de puncte. Soluția sa de la problema *Cub* ar fi obținut 100 de puncte dacă datele de intrare ar fi fost citite din fișierul `CUBE.IN` (în loc de `CUBES.IN`) și rezultatele ar fi fost scrise în fișierul `CUBE.OUT` (în loc de `CUBES.OUT`).

Profităm de "ocazie" pentru a vă atenționa din nou că trebuie să fiți foarte atenți la specificațiile problemelor și la modul în care trebuie construită arhiva pe care o trimiteți.

Cele mai mari punctaje obținute la această rundă sunt:

1. Silvestru-Cosmin Negrușeri, Bistrița	*300*
2. Mugurel-Ionuț Andreica, București	*296*
3. Octavian-Daniel Dumitran, București	*264*
4. Adrian Balașko, Zalău	*252*
5. Radu Dondera, București	*248*
6. Bogdan Nicolae, Sibiu	*236*
7. Liviu-Cosmin Andreicuț, Bacău	*234*
8. Radu-Andrei Ștefan, Brașov	*231*
9. Csaba András, Oradea	*223*
10. Ștefăniță Fechete, Suceava	*214*
11. Andrei Giurgiu, București	*203*
12. Tiberiu Dăneț, București	*202*
13. Adrian-Nicolae Cârcu, Bistrița	*200*
13. Maximilian Machedon, București	*200*
13. Flaviu Pașca, Bistrița	*200*
16. Paul Rusu, Cluj-Napoca	*197*
17. Andrei Vancea, Cluj-Napoca	*184*
18. Victor-Marius Costan, București	*175*
19. Claudiu Raicu, Pitești	*170*
20. Csaba Pătcaș, Oradea	*163*
20. Andrei-Vasile Csibi, Cluj-Napoca	*163*

Cu excepția problemei *Scânduri,* la care au obținut punctajul maxim doar doi concurenți, la celelalte două probleme au existat o mulțime de participanți care au obținut 100 de puncte. Acest fapt demonstrează că, în general, problemele de la această rundă nu au fost foarte dificile, un punctaj de 200 de puncte fiind relativ ușor de obținut.

Datele de test folosite pentru testare, precum și variante ale ieșirilor corecte, pot fi descărcate folosind *link*-urile de pe paginile dedicate celor trei probleme. Adresele acestor pagini sunt:

- www.ginfo.ro/concurs/templu.shtml;
- www.ginfo.ro/concurs/scanduri.shtml;
- www.ginfo.ro/concurs/cub.shtml.

La mulți ani!

Andrei Dulceanu, Pașcani - 1 III 1985
Arcadie Crăcan, Iași - 2 III 1984
Radu Ceucă, Simeria - 3 III 1986
Ovidiu Pintican, Tg. Mureș - 5 III 1981
Alina Fețnic, Apahida - 5 III 1985
Andrei David, Râmnicu Vâlcea - 6 III 1982
Emil-Adrian Șpiac, Arad - 6 III 1985
Cezar-Florin Belghiru, Alexandria - 7 III 1984
Nicolae-Gheorghe Cojocaru, Nanov - 8 III 1984
József Csog, Sfântu Gheorghe - 8 III 1985
Emilian Miron, Bacău - 9 III 1985
Zoltán Szilágyi, Sfântu Gheorghe - 9 III 1985
Csaba Pătcaș, Oradea - 14 III 1983
Octavian-Daniel Dumitran, București - 15 III 1983
Ion Savin, Sighișoara - 17 III 1984
Bogdan-Vasile Hârjoc, Cugir - 18 III 1983
Andrei Paponiu, Drobeta-Turnu Severin - 18 III 1985
Silvia Copil-Crișan, Arad - 19 III 1985
Mihai Voinescu, Târgu-Jiu - 22 III 1982
Liviu-Laurențiu Iacob, București - 22 III 1984
Alexandru Munteanu, Roman - 22 III 1989
Andrei-Eugeniu Ioniță, Slatina - 25 III 1986

Runda 3, problema 1

TEMPLU

Această problemă a fost rezolvată corect de 19 dintre participanții la a treia rundă a concursului *Bursele Agora*, așadar soluția ei nu era foarte greu de găsit.

Problema este o adaptare a uneia dintre problemele de rezervă de la ediția 2001 a *Olimpiadei Internaționale de Informatică*, al cărui autor este ***Janne Kujala***.

Enunțul original era următorul: *se consideră un șir de numere binare*; *folosind acest șir se construiește o matrice în care prima linie este dată de șirul considerat, iar fiecare dintre următoarele linii sunt obținute prin permutarea la stânga cu o poziție a elementelor din linia precedentă*; *în continuare, liniile matricei obținute sunt sortate lexicografic* (*valoarea* 0 *precede valoarea* 1); *dându-se elementele de pe ultima coloană a matricei obținute după sortare, să se determine elementele de pe prima linie a acestei matrice*.

Soluția pe care o vom prezenta în continuare este preluată din cartea *IOI '01 Competition*, care a apărut după terminarea olimpiadei.

Algoritmul este foarte simplu și constă în trei pași:

- se numără elementele cu valoarea 0 și elementele cu valoarea 1 de pe ultima coloană și, pe baza rezultatului numărării, se construiește prima coloană a matricei; în continuare vom presupune că avem p elemente cu valoarea 0 și q elemente cu valoarea 1.
- se creează un șir a astfel:
  - pentru $i \le p$, elementul a_i va avea ca valoare linia pe care se află cel de-al i-lea element cu valoarea 0 de pe ultima coloană;
  - pentru $j \le q$, elementul a_{p+j} va avea ca valoare linia pe care se află cel de-al j-lea element cu valoarea 1 de pe ultima coloană;
- începând cu prima linie, se parcurg liniile folosind vectorul construit a (de la linia k se trece la linia a_k) și sunt reconstituite elementele de pe prima linie pe baza valorilor de pe ultima coloană.

În cele ce urmează vom demonstra corectitudinea algoritmului prezentat. Numin ***succesor*** al unui șir, șirul obținut printr-o permutare cu o poziție la stânga a elementelor șirului.

Pentru a demonstra corectitudinea algoritmului, va trebui să arătăm că șirul a reprezintă exact această relație de succesiune între șiruri. După demonstrarea acestui fapt, este evident că al treilea pas al algoritmului reconstituie corect elementele de pe prima linie.

Considerăm liniile (din matricea obținută după sortare) care încep cu un element care are valoarea 0. Deoarece, aceste șiruri sunt ordonate în matricea sortată, după efectuarea unei permutări la stânga cu o poziție, șirurile obținute (care se termină cu un element care are valoarea 0) sunt și ele sortate.

Aceeași proprietate este valabilă și pentru liniile care încep cu un element care are valoarea 1.

Deoarece aceste proprietăți sunt echivalente cu relația de succesiune, corectitudinea algoritmului este demonstrată.

Vom demonstra în continuare că timpul de execuție al algoritmului este unul liniar. Pentru aceasta vom arăta că fiecare dintre cei trei pași se efectuează în timp liniar.

Numărarea elementelor de pe ultima coloană se realizează printr-o singură parcurgere, deci liniaritatea este evidentă.

Inițializarea tabloului a necesită doar o singură parcurgere a acestuia și a ultimei coloane, deci timpul este liniar și în acest caz.

Cel de-al treilea pas se execută și el în timp liniar deoarece la fiecare tranziție se determină câte un element de pe prima linie a matricei sortate.

Codul sursă al programului este disponibil pentru *download* la adresa **www.ginfo.ro/concurs/templu.shtml**.

Au rezolvat corect:

Adrian Airinei, Bacău
Csaba András, Oradea
Mugurel-Ionuț Andreica, București
Liviu-Cosmin Andreicuț, Bacău
Adrian Balașko, Zalău
Adrian-Nicolae Cârcu, Bistrița
Victor-Marius Costan, București
Andrei-Vasile Csibi, Cluj-Napoca
Radu Dondera, București
Octavian-Daniel Dumitran, București
Ștefăniță Fechete, Suceava
Andrei Giurgiu, București
Maximilian Machedon, București
Silvestru-Cosmin Negruțeri, Bistrița
Bogdan Nicolae, Sibiu
Flaviu Pașca, Bistrița
Paul Rusu, Cluj-Napoca
Radu-Andrei Ștefan, Brașov
Andrei Vancea, Cluj-Napoca

Runda 3, problema 2

SCÂNDURI

Aceasta a fost cea mai dificilă problemă propusă spre rezolvare la cea de-a treia rundă a ediției 2001/2002 a concursului *Bursele Agora*. Doar doi dintre concurenții care au trimis soluții au reușit să obțină 100 de puncte; este vorba de ***Cosmin-Silvestru Negruşeri*** și ***Adrian Cârcu***, ambii din *Bistrița*. Foarte aproape de a obține punctajul maxim a fost ***Mugurel-Ionuț Andreica*** din *București*; el a obținut 96 de puncte.

Problema ***Scânduri*** face parte din categoria problemelor *NP*-complete, deci găsirea unui algoritm polinomial pentru rezolvarea ei ar însemna demonstrarea celebrei probleme cu care se confruntă cei care studiază algoritmica din punct de vedere teoretic: ***P ≡ NP***.

Conform așteptărilor, nu a fost găsită nici o soluție polinomială, deci nici unul dintre concurenți nu poate, deocamdată, intra în posesia premiului de un milion de dolari care va fi oferit celui care va găsi o soluție polinomială pentru o problemă *NP*-completă sau va demonstra că o astfel de soluție nu poate exista.

În continuare vom prezenta modul în care va trebui construit algoritmul exponențial de rezolvare a acestei probleme care să se încadreze în limita de timp impusă.

Pentru a determina soluția vom folosi metoda *backtracking*. Restricțiile impuse în enunț (cel mult cinci scânduri cu aceeași lungime și cel mult cinci bucăți cu aceeași lungime) permit scrierea de programe care să găsească soluția în mai puțin de cinci secunde.

Pentru a micșora cât mai mult posibil spațiul de căutare a soluțiilor va trebui să determinăm câte posibilități de reconstituire a unei scânduri există. Operația poate fi realizată foarte simplu în timp pătratic. Ulterior, vom ordona scândurile în funcție de numărul posibilităților de reconstituire.

Apoi, la fiecare pas al algoritmului care folosește metoda *backtracking* vom considera scândurile rămase în ordinea din șirul sortat. După alegerea a două bucăți care, prin lipire, duc la obținerea unei scânduri, numărul posibilităților de reconstituire se schimbă. Noile valori pot fi obținute în timp liniar. Va urma o reordonare a șirului care trebuie realizată în timp liniar-logaritmic.

În momentul în care am reconstituit toate scândurile, vom afișa soluția.

O implementare a algoritmului prezentat, care optimizează cât mai mult posibil viteza cu care sunt realizate operațiile de la fiecare pas, ar fi obținut toate cele 100 de puncte.

Dacă nu se reușea găsirea unui algoritm performant, puteau fi folosite diverse metode pentru obținerea a cât mai multe puncte.

Datorită faptului că se acordau punctaje parțiale în cazul în care erau reconstituite cel puțin trei sferturi dintre scânduri, este evident faptul că, de fiecare dată când se găsește o soluție mai bună (cu mai multe scânduri reconstituite) decât cea mai bună găsită anterior, atunci aceasta este păstrată. La expirarea intervalului de timp admis pentru execuția programului va fi afișată cea mai bună soluție găsită până în acel moment.

Chiar și unele versiuni mai puțin performante ale algoritmului care folosește metoda *backtracking* ar fi găsit soluțiile pentru testele în care numărul scândurilor era mic. Pentru un număr mai mare de scânduri, în general, puteau fi reconstituite trei sferturi dintre acestea fără ca aceasta să creeze dificultăți deosebite.

O altă posibilitate de a reconstitui cât mai multe puncte era folosirea unor algoritmi probabiliști.

O variantă era de a se încerca reconstituirea unei scânduri, folosind bucățile date (cele două bucăți sunt alese aleator, dar prin lipirea lor trebuie neapărat să fie obținută o scândură), de a elimina bucățile folosite după care se reconstituie altă scândură folosind bucățile rămase etc. În momentul în care nu mai pot fi reconstituite scânduri algoritmul se reia. Procesul se repetă până la expirarea timpului de execuție admis. În final, este aleasă cea mai bună soluție obținută.

Algoritmul are șanse de a reconstitui toate scândurile doar dacă numărul acestora este mic. Pentru valori medii ale numărului de scânduri, există șanse destul de mari să fie reconstituite trei sferturi dintre acestea. În cazul în care numărul scândurilor este mare, șansele de a reconstitui trei sferturi dintre acestea sunt destul de mici.

O altă abordare este folosirea unor metode euristice. Acestea pot combina principiile folosite de algoritmul cu metoda *backtracking* și algoritmul probabilist. Totuși, punctajele obținute de concurenții care au folosit astfel de metode au fost chiar mai mici decât cele obținute de cei care au ales strategiile aleatoare.

Exista și posibilitatea implementării unui algoritm genetic. Chiar dacă această abordare era dificil de implementat, ea putea duce la obținerea punctajului maxim.

Codul sursă al programului care folosește metoda *backtracking* este disponibil pentru *download* la adresa **www.ginfo.ro/concurs/scanduri.shtml**.

Runda 3, problema 3

CUB

Aceasta a fost cea mai simplă problemă a rundei a treia, 20 dintre participanți obținând punctajul maxim. Alți 10 concurenți au obținut peste 85 de puncte.

Problema ***Cub*** putea fi rezolvată foarte ușor folosind metoda *programării dinamice*.

Considerăm, pe rând, literele cuvântului care trebuie căutat (fie acesta *s*). La fiecare pas *x* vom putea determina numărul de apariții al cuvântului format din primele *x* caractere ale cuvântului dat.

Pentru aceasta, la fiecare pas vom calcula valorile a_{ijk}, unde a_{ijk} indică numărul de apariții ale cuvântului format din primele *x* litere ale șirului dat, astfel încât elementul din cubulețul de coordonate (*i*, *j*, *k*) a cubului să fie s_x.

Pentru toate căsuțele care nu conțin litera s_x valoarea a_{ijk} va fi 0.

Evident, la primul pas vom avea $a_{ijk} = 1$ pentru toate pozițiile pe care se găsește valoarea s_1 și $a_{ijk} = 0$ pentru celelalte poziții.

La următorul pas, pentru fiecare element care conține valoarea s_2 vom număra elementele vecine care au valoarea 1 și vom atribui elementului corespunzător valoarea obținută. Practic, se poate spune că vom aduna toate valorile elementelor vecine.

La al treilea pas, pentru fiecare element care conține valoarea s_3 vom calcula valoarea corespunzătoare însumând valorile elementelor vecine care corespund unor cubulețe în care se află litera s_2.

Se observă că, practic, și la al doilea pas se poate realiza o operație similară: pentru toate elementele corespunzătoare unui cubuleț în care se află litera s_2, se însumează valorile elementelor vecine care corespund unor cubulețe care conțin litera s_1.

În general, pentru toate elementele corespunzătoare unui cubuleț în care se află litera s_i, se însumează valorile elementelor vecine care corespund unor cubulețe care conțin litera s_{i-1} (pentru $i > 1$).

Soluția este dată de suma elementelor corespunzătoare cubulețelor în care se află litera s_n, unde *n* este lungimea cuvântului dat.

Datorită faptului că literele cuvântului sunt distincte, nu apar probleme suplimentare la calcularea valorilor elementelor. Dacă literele s-ar fi putut repeta, în cazul a două litere consecutive identice, ar fi putut apărea erori. Soluția ar fi fost dată de folosirea unui tablou tridimensional suplimentar care ar fi păstrat valorile calculate la iterația anterioară.

Au rezolvat corect:

Mugurel-Ionuț Andreica, București
Adrian Balașko, Zalău
Cristian Băicoianu, Ploiești
Vlad Dascălu, Bacău
Tiberiu Dănet, București
Radu Dondera, București
Octavian-Daniel Dumitran, București
Ștefan Gedo, București
Andrei Giurgiu, București
Andrei Homescu, Târgu-Jiu
Quoc Vinh Ly, Vietnam
Maximilian Machedon, București
Silvestru-Cosmin Negrușeri, Bistrița
Bogdan Nicolae, Sibiu
Sorin Otescu, Brașov
Flaviu Pașca, Bistrița
Csaba Pătcaș, Oradea
Alpár-Ferenc Perini, Sfântu Gheorghe
Mihail-Cosmin Pițigoi-Rada, Dr. Tr. Severin
Constantin Răuțu, Botoșani

Datorită faptului că numărul de apariții ale cuvântului este de cel mult 2.000.000.000, fiecare element al tabloului tridimensional necesită patru bytes pentru a fi reprezentat. Dimensiunea maximă a laturii cubului este 39, deci cantitatea de memorie necesară pentru a păstra elementele este de $39^3 \cdot 4 = 237276$ bytes ≈ 232 KB.

Pentru a păstra literele din cubulețe avem nevoie de alți $39^3 = 59319$ bytes ≈ 58 KB. Așadar, dimensiunea totală a structurilor de date pe care le folosim este de aproximativ 290 KB, deci se încadrează în limita de 300 KB specificată în regulamentul concursului.

Numărul de parcurgeri ale tabloului tridimensional este egal cu lungimea *n* a șirului dat. Ordinul de complexitate al unei parcurgeri este $O(k^3)$, unde *k* este lungimea laturii cubului. Ca urmare, ordinul de complexitate al algoritmului descris este $O(n \cdot k^3)$.

Codul sursă al programului este disponibil pentru *download* la adresa **www.ginfo.ro/concurs/cub.shtml**.