CONCURSUL de programare AGORA

S-a încheiat și ultima rundă a Concursului de Programare Agora. În cadrul acestei număr vă vom prezenta soluțiile problemelor de la runda a șasea, clasamentul final al concursului, precum și detalii referitoare la etapa finală la care participă primii zece clasați.

Cele trei probleme ale ultimei runde au fost relativ dificile, dovadă fiind numărul mic de concurenți care au reușit să obțină puncte. De obicei, vă prezentam și un clasament al rundei care conținea numele și punctajele celor mai buni 50 de concurenți la runda respectivă. De această dată am avut doar 49 de participanți al căror punctaj să fie diferit de 0, motiv pentru care ne limităm la prezentarea celor care au obținut primele 25 de punctaje. Aceștia sunt:

1. Mugurel Ionuț Andreica, București	*300*
1. Cosmin-Silvestru Negruseri, Bistrița	*300*
3. Maximilian Machedon, București	*275*
4. Liviu Lalescu, Craiova	*243*
5. Mihai Pătrașcu, Craiova	*233*
6. Radu Vătavu, Suceava	*217*
7. Vencel Bors, Oradea	*216*
8. Csaba András, Oradea	*213*
9. Valentin Bișa, Lugoj	*211*
10. Ciprian Baicu, Drobeta Turnu Severin	*200*
10. Adrian Cârcu, Bistrița	*200*
10. George Drumea, București	*200*
10. Octavian-Daniel Dumitran, București	*200*
10. Flaviu Pașca, Bistrița	*200*
10. Mihai Stroe, București	*200*
10. Alexandru-Emilian Susu, București	*200*
17. Csaba Pătcaș, Oradea	*189*
18. Dan Ghinea, București	*186*
19. Valentin Pop, Carei	*172*
20. Adrian Balasko, Zalău	*164*
21. Ciprian Cană, Vaslui	*152*
22. Cristian Toth, Lugoj	*151*
23. Horea Coroiu, Cluj-Napoca	*143*
24. Bogdan Nicolae, Sibiu	*137*
25. Claudiu Gruia, București	*122*

Despre probleme

Problemele de la ultima rundă au fost puțin mai dificile decât cele de la primele cinci.

Prima dintre ele era foarte simplă, fiind un exemplu clasic de aplicare a metodei *divide et impera*.

Cea de-a doua a fost, probabil, cea mai dificilă problemă propusă spre rezolvare în cadrul *Concursului de Programare Agora*. Era destul de greu de găsit algoritmul de rezolvare, dar cea mai dificilă parte a rezolvării o reprezenta implementarea eficientă a acesteia.

Ultima problemă a fost mai simplă. Cea mai "comodă" rezolvare a sa consta în folosirea "metodei constantelor".

Etapa finală

Primii zece clasați în urma celor șase etape ale concursului vor putea participa la faza finală care se va desfășura la Cluj-Napoca pe data de 26 mai 2001.

La această etapă, pot participa ca invitați (în afara concursului) și alți participanți la concursul organizat de revista noastră. Aceștia vor trebui să suporte cheltuielile de cazare și transport și, evident, trebuie să înștiințeze organizatorii că doresc să se deplaseze la *Cluj*.

Bilanț

După încheierea concursurilor se realizează, de obicei, bilanțuri. Vă prezentăm câteva date statistice referitoare la *Concursul de Programare Agora*.

Au fost 402 participanți înscriși, dar doar 176 dintre ei au reușit să obțină puncte. Printre concurenți s-au numărat și 86 de elevi sau studenți din afara României. La concurs au participat doar 22 de fete și numai 12 dintre ele au obținut puncte. 132 de concurenți au reușit să rezolve corect cel puțin o problemă și 46 dintre ei au obținut punctaj maxim (300 de puncte) la cel puțin una dintre runde.

Runda 6, problema 1

Inversiuni

Această problemă a fost rezolvată corect de către 28 dintre cei care au participat la a șasea rundă a ***Concursului de Programare Agora***. Problema era foarte cunoscută, ea fiind propusă spre rezolvare la numeroase alte concursuri.

Cea mai simplă idee de rezolvare este de a lua în considerare toate perechile (i, j) pentru care $i < j$ și a verifica dacă $a_i > a_j$.

Este evident că această metodă folosește $n \cdot (n - 1) / 2$ comparări, deci ordinul de complexitate al algoritmului este $O(n^2)$.

Chiar și un program care implementa un astfel de algoritm s-ar fi încadrat în limita de timp admisă dacă nu efectua operații suplimentare inutile. Au existat câțiva concurenți care au obținut 100 de puncte folosind această metodă de rezolvare.

Există și un algoritm de rezolvare, bazat pe metoda *divide et impera*, care are ordinul de complexitate $O(n \cdot \log n)$.

Etapa divide
Se împarte șirul de numere în două subșiruri de lungimi aproximativ egale. Primul subșir va conține prima jumătate a numerelor, iar al doilea cea de-a doua jumătate.

Etapa stăpânește
Se apelează recursiv algoritmul pentru cele două subșiruri formate și se determină numărul de inversiuni din fiecare dintre ele.

Etape combină
Numărul de inversiuni din șir va fi egal cu suma numărului de inversiuni din cele două subșiruri la care se adaugă numărul de inversini formate de elemente din primul subșir cu elemente din al doilea subșir.

Pentru fiecare element din al doilea subșir va trebui să determinăm numărul de elemente din primul subșir care sunt mai mari decât el. Pentru aceasta vom ordona cele două subșiruri (această operație poate fi realizată pe parcursul apelurilor recursive succesive) și vom realiza o simplă operație de interclasare. În momentul în care vom adăuga la șirul interclasat un element din al doilea subșir, vom ști câte elemente mai mari decât el se află în primul subșir.

Practic, folosim algoritmul *mergesort* de ordonare a unui șir de numere naturale. Singura operație suplimentară pe care o efectuăm este numărarea inversiunilor.

Versiunea oficială a rezolvării acestei probleme este următoarea:

Au rezolvat corect:

Csaba András, Mugurel Ionuț Andreica, Ciprian Baicu, Adrian Balasko, Valentin Bișa, Vencel Bors, Lucian Bucur, Ciprian Cană, Adrian Cârcu, Horea Coroiu, George Drumea, Daniel Dumitran, Claudiu Gruia, Mihai Hărănguș, Alexandru Ivan, Liviu Lalescu, Maximilian Machedon, Mihail-Cosmin Pițigoi-Rada, Silvestru Negruseri, Bogdan Nicolae, Flaviu Pașca, Pătcaș Csaba, Mihai Pătrașcu, Lucian-Daniel Stanciu, Mihai Stroe, Alexandru Susu, Cristian Toth, Radu Vătavu

Listing INVERS.CPP

```
#include <stdio.h>

long n,nr,huge a[10001],huge b[10001];

void ReadData(void){
  FILE *f=fopen("INVERS.IN","rt");
  fscanf(f,"%d",&n);
  for (int i=0;i<n;i++)
    fscanf(f,"%ld",&a[i]);
  fclose(f);
}

void MergeSort(int x, int y){
  if (x==y) return;
  int m=(x+y)/2,i=x,j=m+1,p=0;
  MergeSort(x,m);
  MergeSort(m+1,y);
  while (i<=m || j<=y)
    if ((a[j]<a[i] && j<=y) || i>m)
      b[x+p++]=a[j++],nr+=m-i+1;
    else b[x+p++]=a[i++];
  for (i=x;i<=y;i++) a[i]=b[i],b[i]=0;
}

void WriteSolution(void){
  FILE *f=fopen("INVERS.OUT","wt");
  fprintf(f,"%ld",nr);
  fclose(f);
}

void main(void){
  ReadData();
  MergeSort(0,n-1);
  WriteSolution();
}
```

Rețea

Această problemă a fost rezolvată corect doar de doi participanți la cea de-a șasea rundă a ***Concursului de Programare Agora*** și anume ***Cosmin-Silvestru Negruseri*** din *Bistrița* și ***Mugurel Ionuț Andreica*** din *București*. Au mai fost concurenți care au rezolvat corect problema, dar nu au respectat formatul fișierului de ieșire. Cea mai frecventă eroare care a apărut a fost separarea coordonatei orizontale de cea verticală prin virgulă în loc de spațiu.

Dintre cei 43 de concurenți care au trimis o soluție pentru această problemă doar șase au obținut peste 50 de puncte, iar 18 dintre ei nu au obținut nici un punct.

Problema se reduce la determinarea unui flux maxim într-o rețea de transport în care nodul sursă este conectat la punctele din care pornesc drumurile, iar nodul destinație este conectat la punctele de pe marginea rețelei.

Există și alte soluții echivalente care folosesc diferite metode de rezolvare. Totuși, în principiu, acești algoritmi de rezolvare a acestei probleme sunt echivalenți cu algoritmul de determinare a fluxului maxim într-o rețea de transport.

Vă prezentăm în continuare soluția propusă de ***Cosmin-Silvestru Negruseri***, concurentul clasat pe primul loc după cea de-a șasea etapă a concursului nostru.

Listing NET.PAS

```
Program Retea;

type  matrix=array[0..101,0..101] of Byte;
      imatrix=array[0..101,0..101] of Integer;
      isir=array[1..20000] of Integer;
      sir=array[1..20000] of Byte;

const  dir:array[1..4,1..2] of Shortint=
                 ((-1,0),(0,-1),(1,0),(0,1));
       invers:array[1..4] of Shortint=
                                  (3,4,1,2);

var a:array[0..4] of matrix;
    at:array[1..2] of ^imatrix;
    vv:array[1..10,1..10] of Byte;
    x,y,z:^sir;
    t:^isir;
    lx,ly,p:array[1..400] of Byte;
    n,m,h,t1,t2,i,j,last,aux,pas:Longint;
    bool:Boolean;
    fis:Text;

procedure finish;
begin
  Assign(fis,'NET.OUT');
  Rewrite(fis);
  Writeln(fis,0);
  Close(fis);
  Halt
end;
function min(a1,a2:Integer):Integer;
begin
  if a1>a2 then
    min:=a2
  else min:=a1
end;

procedure qsort(l,r:Integer);
var i,j,mid:Integer;
begin
  i:=l;
  j:=r;
  mid:=p[(l+r) shr 1];
  repeat
    while p[i]<mid do
      Inc(i);
    while p[j]>mid do
      Dec(j);
    if i<=j then
    begin
      aux:=p[i];
      p[i]:=p[j];
      p[j]:=aux;
      aux:=lx[i];
      lx[i]:=lx[j];
      lx[j]:=aux;
      aux:=ly[i];
      ly[i]:=ly[j];
      ly[j]:=aux;
      Inc(i);
      Dec(j)
    end
  until i>j;
```

```
  if l<j then
    qsort(l,j);
  if i<r then
    qsort(i,r)
end;

procedure readdata;
begin
  Assign(fis,'NET.IN');
  Reset(fis);
  Readln(fis,n,m);
  if m>4*n-4 then
  begin
    Close(fis);
    finish
  end;
  for i:=1 to m do
    Readln(fis,lx[i],ly[i]);
  Close(fis);
  for i:=1 to m do
    p[i]:=min(lx[i]-1,min(ly[i]-1,
                    min(n-lx[i],n-ly[i])));
  qsort(1,m)
end;

procedure add(v,ii,jj,kk:Integer);
begin
  if (ii<1) or (ii>n) or (jj<1) or (jj>n) or
                   (at[kk]^[ii,jj]=pas) then
    Exit;
  at[kk]^[ii,jj]:=pas;
  inc(t2);
  x^[t2]:=ii;
  y^[t2]:=jj;
  z^[t2]:=kk;
  t^[t2]:=v;
  if (ii=1) or (jj=1) or (ii=n) or (jj=n)
    then bool:=true
end;

procedure expand(v,i,j,k:Integer);
begin
  if k=1 then
  begin
    if a[0,i,j]=0 then
      add(v,i,j,2);
    if a[3,i-1,j]=1 then
      add(v,i-1,j,2);
    if a[1,i+1,j]=1 then
      add(v,i+1,j,2);
    if a[4,i,j-1]=1 then
      add(v,i,j-1,2);
    if a[2,i,j+1]=1 then
      add(v,i,j+1,2)
  end
```

```
  else
    begin
      if a[3,i,j]=0 then
      begin
        add(v,i+1,j,1);
        if bool then
          Exit
      end;
      if a[1,i,j]=0 then
      begin
        add(v,i-1,j,1);
        if bool then
          Exit
      end;
      if a[4,i,j]=0 then
      begin
        add(v,i,j+1,1);
        if bool then
          Exit
      end;
      if a[2,i,j]=0 then
      begin
        add(v,i,j-1,1);
        if bool then
          Exit
      end;
      if a[0,i,j]=1 then
        add(v,i,j,1)
    end
end;

procedure lee_it(ii,jj:Byte);
var i:Integer;
begin
  bool:=false;
  h:=1;
  t1:=1;
  x^[1]:=ii;
  y^[1]:=jj;
  z^[1]:=1;
  t^[1]:=0;
  repeat
    t2:=t1;
    for i:=h to t1 do
    begin
      expand(i,x^[i],y^[i],z^[i]);
      if bool then
        Break
    end;
    if bool then
      Break;
    h:=t1+1;
    t1:=t2;
  until h>t1
end;
```

```
procedure drum(vvv:Integer);
var i,j,i1,j1,dire,v:Integer;
begin
  v:=vvv;
  i:=x^[v];
  j:=y^[v];
  while t^[v]<>0 do
  begin
    v:=t^[v];
    i1:=i;
    j1:=j;
    i:=x^[v];
    j:=y^[v];
    if (i=i1) and (j=j1) then
      if a[0,i,j]=1
        then a[0,i,j]:=0
      else a[0,i,j]:=1;
    else
      begin
        for dire:=1 to 4 do
          if (i+dir[dire,1]=i1) and
                    (j+dir[dire,2]=j1) then
            Break;
        if a[invers[dire],i1,j1]=1 then
          a[invers[dire],i1,j1]:=0
        else
          begin
            a[dire,i,j]:=1;
            a[invers[dire],i1,j1]:=0
          end
      end
  end
end;

procedure work;
var i,j:Integer;
begin
  New(x);
  New(y);
  New(z);
  New(t);
  New(at[1]);
  New(at[2]);
  Fillchar(at[1]^,Sizeof(at[1]^),0);
  Fillchar(at[2]^,Sizeof(at[2]^),0);
  for pas:=1 to m do
    if (lx[pas]<>1) and (lx[pas]<>n) and
          (ly[pas]<>1) and (ly[pas]<>n) then
    begin
      for j:=1 to m do
        at[1]^[lx[j],ly[j]]:=pas;
      lee_it(lx[pas],ly[pas]);
      if h>t1 then finish;
      drum(t2)
    end;
  last:=pas;
  Dispose(x);
  Dispose(y);
  Dispose(z);
  Dispose(t);
  Dispose(at[1]);
  Dispose(at[2])
end;

procedure writedata;
var k,i,j,nr,k1:Integer;
begin
  Assign(fis,'NET.OUT');
  Rewrite(fis);
  for k:=1 to m do
  begin
    nr:=1;
    i:=lx[k];
    j:=ly[k];
    while a[0,i,j]=1 do
    begin
      for k1:=1 to 4 do
        if a[k1,i,j]=1 then
        begin
          i:=i+dir[k1,1];
          j:=j+dir[k1,2];
          Break
        end;
      Inc(nr)
    end;
    Writeln(fis,nr);
    i:=lx[k];
    j:=ly[k];
    Write(fis,'(',i,' ',j,')');
    while a[0,i,j]=1 do
    begin
      for k1:=1 to 4 do
        if a[k1,i,j]=1 then
        begin
          i:=i+dir[k1,1];
          j:=j+dir[k1,2];
          Write(fis,' (',i,' ',j,')');
          Break
        end
    end;
    Writeln(fis)
  end;
  Close(fis)
end;

Begin
  readdata;
  work;
  writedata
End.
```

Numere prime

Această problemă a fost rezolvată corect de 17 dintre cei care au participat la a șasea rundă a *Concursului de Programare Agora*. Problema era dificilă dacă se încerca scrierea unui algoritm de verificare a primalității numerelor, dar relativ simplă dacă se folosea *metoda constantelor*.

Vom furniza ca soluții cele mai mici numere de n cifre care sunt prime. Acestea pot fi obținute prin diferite metode. Trebuie remarcat faptul că aceste numere au forma $10^{n-1} + k$, unde k este o valoarea relativ mică.

Se știe că există aproximativ $\ln n$ numere prime mai mici decât un număr n.

Este evident că există aproximativ $(n - 1) \cdot \ln 10$ numere prime mai mici decât 10^{n-1}.

Vom încerca să aproximăm valoarea maximă k în cel mai defavorabil caz (100 de numere prime cu 100 de cifre).

Numărul de numere prime cuprinse între n și $n + k$ este de aproximativ $k / \ln n$.

Putem deduce destul de ușor că valoarea maximă a lui k este de aproximativ $10000 \cdot \ln 10$.

Așadar valorile k pot fi stocate folosind constante de patru bytes.

Puteau fi scrise și programe care memorau în întregime numerele prime care constituiau soluțiile. Apar probleme datorită numărului mare de constante care trebuie memorate. Din această cauză codul sursă depășea dimensiunea de 64 KB. Mediul *Borland Pascal 7.0* nu permite compilarea programelor care depășesc această dimensiune, dar mediul *Borland C++ 3.1* poate compila și astfel de surse dacă sunt specificate opțiunile de compilare adecvate.

Versiunea oficială a rezolvării acestei probleme are următoarea formă.

Listing PRIME.CPP

```
#include <stdio.h>

const long
  digits_2[]={1,3},
  digits_3[]={1,3,7},
  ...

void main(void){
  FILE *f=fopen("PRIME.IN","rt");
  int i,m,n;
  fscanf(f,"%d%d",&n,&m);
  fclose(f);
  f=fopen("PRIME.OUT","wt");
  switch (n){
    case 1: fprintf(f,"3\n");
            break;
    case 2: for (i=0;i<m;i++)
              fprintf(f,"1%01ld\n",
                              digits_2[i]);
            break;
    case 3: for (i=0;i<m;i++)
              fprintf(f,"1%02ld\n",
                              digits_3[i]);
            break;
    ...
    case 100: for (i=0;i<m;i++)
                fprintf(f,"1%099ld\n",
                              digits_100[i]);
  }
  fclose(f);
}
```

Au rezolvat corect:

Csaba András, Oradea
Mugurel Ionuț Andreica, București
Ciprian Baicu, Drobeta Turnu Severin
Bors Vencel, Oradea
Adrian Cârcu, Bistrița
George Drumea, București
Octavian-Daniel Dumitran, București
Dan Ghinea, București
Liviu Lalescu, Craiova
Maximilian Machedon, București
Cosmin-Silvestru Negruseri, Bistrița
Flaviu Pașca, Bistrița
Mihai Pătrașcu, Craiova
Valentin Pop, Carei
Mihai Stroe, București
Alexandru-Emilian Susu, București
Radu Ștefan, Brașov