CLASA a V-a – PROBA PRACTICĂ

TIP SUBIECT – PROGRAMARE

Problema SIR

Time limit: 0.1s Memory limit: 64MB
Input: sir.in
Output: sir.out

Enunț

Mare matematician, Micul Gates studiază următorul șir de numere: 1^n, 3^n, 5^n, 7^n, 9^n, 11^n….

Cerință

1. Să se determine cifra unităților numărului de pe poziția k din șir.
2. Să se determine suma cifrelor unităților primelor k numere din șir.

Date de intrare

Pe prima linie a fișierului de intrare sir.in se găsește numărul c al cerinței, care poate fi doar 1 sau 2. Pe a doua linie se găsesc două numere naturale nenule, n și k, separate printr-un spațiu, cu semnificația din enunț

Date de ieșire

Pe prima linie a fișierului de ieșire sir.out se va găsi valoarea determinată conform cerinței.

Restricții și precizări

- 1≤n,k≤1 000 0001;
- Pentru cerința 1, se acordă 32 de puncte;
- Pentru cerința 2, se acordă 68 de puncte. Pentru 12 puncte, avem n=1.

Descrierea soluției

Șirul de numere este alcătuit din: 1^n, 3^n, 5^n, 7^n, 9^n, 11^n….

Elementele șirului sunt de forma: $(2*k-1)^n$

Aceasta înseamnă că:

- Pentru k = 1, elementul va fi 1^n (adică 1^n),
- Pentru k = 2, elementul va fi 3^n (adică 3^n),
- Pentru k = 3, elementul va fi 5^n (adică 5^n),

și așa mai departe….

De asemenea, se observă că ultima cifră a numerelor din șir ridicate la puterea n se repetă din 5 în 5 poziții, deci vom afla ce cifră a unităților are numărul care este ridicat la puterea n în șir, pe poziția k în funcție de restul împărțirii lui k la 5. Apoi vom determina cifra unităților astfel:

- Dacă cifra unităților numărului din șir este egală cu 1, atunci cifra unităților numărului 1^n este mereu egală cu 1.
- Dacă cifra unităților numărului din șir este 3, atunci cifra unităților lui 3^n urmează un model ciclic. Pentru rezolvare, analizăm cifrele unităților puterilor succesive ale lui 3

și se observă că cifra unităților lui 3^n urmează un ciclu de 4 valori: 3, 9, 7, 1. Pentru a găsi cifra unităților lui 3^n, trebuie doar să verifici restul împărțirii lui n la 4:

- Dacă n mod 4=1, cifra unităților este 3.
- Dacă n mod 4=2, cifra unităților este 9.
- Dacă n mod 4=3, cifra unităților este 7.
- Dacă n mod 4=0, cifra unității este 1.

De exemplu: Pentru 3^5, 5 mod 4= 1, deci cifra unităților este 3; pentru 3^7, 7 mod 4=3, deci cifra unităților este 7.

- Dacă cifra unităților numărului din șir este egală cu 5, atunci cifra unităților numărului 5^n este mereu egală cu 5.
- Cifra unităților lui 7^n, de asemenea, se repetă. Observăm acest lucru analizând puterile succesive ale lui 7 și deducem că cifra unităților lui 7^n poate fi: 7, 9, 3, 1. Pentru a găsi cifra unităților lui 7^n, trebuie să verifici restul împărțirii lui n la 4:
  - Dacă n mod 4=1, cifra unităților este 7.
  - Dacă n mod 4=2, cifra unităților este 9.
  - Dacă n mod 4=3, cifra unităților este 3.
  - Dacă n mod 4=0, cifra unităților este 1.
- Cifra unităților lui 9^n, de asemenea, se repetă. Observăm acest lucru analizând puterile succesive ale lui 9 și deducem că cifra unităților lui 9^n poate fi: 9 sau 1. Pentru a găsi cifra unităților lui 9^n, trebuie să verifici restul împărțirii lui n la 2:
  - Dacă n mod 2=1, cifra unităților este 9.
  - Dacă n mod 2=0, cifra unităților este 1.

Cu toate observațiile de mai sus putem rezolva cele două cerințe.
Un exemplu de rezolvare este mai jos:

```
#include <fstream>
#include <iostream>
using namespace std;
ifstream fin("sir.in");
ofstream fout("sir.out");
int cerinta, n, i,suma,k,u;
int main()
{  fin>>cerinta;
  fin>>n>>k;
  if(cerinta==1)
    {if(k%5==1) u=1;
     else if(k%5==2) {if(n%4==1) u=3;
                      else if(n%4==2) u=9;
                        else if(n%4==3) u=7;
                          else u=1;
                     }
       else if(k%5==3) u=5;
         else if(k%5==4) {if(n%4==1) u=7;
                          else if(n%4==2) u=9;
```

```
                                else if(n%4==3) u=3;
                                    else u=1;
                            }
                        else {if(n%2==1) u=9;
                            else u=1;
                           }
      fout<<u<<'\n';
     }
   else
   { suma=0;
    for(i=1;i<=k;i++)
      {if(i%5==1) u=1;
       else if(i%5==2) {if(n%4==1) u=3;
                    else if(n%4==2) u=9;
                        else if(n%4==3) u=7;
                            else u=1;
                    }
          else if(i%5==3) u=5;
              else if(i%5==4) {if(n%4==1) u=7;
                           else if(n%4==2) u=9;
                               else if(n%4==3) u=3;
                                   else u=1;
                           }
                       else {if(n%2==1) u=9;
                           else u=1;
                          }
        suma=suma+u;
       }
      fout<<suma<<'\n';
      }
}
```

PROBLEMA NUMERE

Time limit: 0.1s Memory limit: 64MB
Input: numere.in
Output: numere.out

Cerință

Micul Gates are de rezolvat o problemă la informatică. El primește un șir format din nnn numere naturale și trebuie să răspundă următoarelor cerințe:

1. Să se determine câte numere din șir au exact două cifre.
2. Să se determine a k-a cifră care apare în șirul de numere. Dacă nu există, se afișează mesajul *nu exista*. Numerotarea cifrelor se face de la stânga la dreapta, începând cu poziția 1.
3. Să se determine cel mai mare număr care se poate forma din cel mult m cifre consecutive aflate în numerele din șirul dat, parcurgând cifrele de la stânga la dreapta.

Date de intrare

Pe prima linie a fișierului de intrare numere.in se găsește numărul c al cerinței, care poate fi doar 1, 2 sau 3.

Dacă cerința este 1, pe a doua linie se găsește numărul natural nenul n iar pe a treia linie se găsesc n numere naturale, separate prin câte un spațiu.

Dacă cerința este 2, pe a doua linie se găsesc numerele n și k separate prin câte un spațiu, cu semnificația din enunț. Pe a treia linie se găsesc cele n numere naturale, separate prin câte un spațiu.

Dacă cerința este 3, pe a doua linie se găsesc numerele n și m separate prin câte un spațiu, cu semnificația din enunț. Pe a treia linie se găsesc cele n numere naturale, separate prin câte un spațiu.

Date de ieșire

Pe prima linie a fișierului de ieșire numere.out se va găsi valoarea determinată conform cerinței.

Restricții și precizări

- 1≤n≤100 0001;
- Numerele din șir sunt numere naturale cu cel mult 9 cifre fiecare.
- 1≤k≤900 000;
- 1≤m≤91;
- Pentru cerința 1, se acordă 16 puncte;
- Pentru cerința 2, se acordă 52 puncte (pentru 8 puncte, numerele din șir au o singură cifră);
- Pentru cerința 3, se acordă 32 puncte, dintre care:
  - m=1 pentru 4 puncte;
  - Pentru alte 28 puncte, m>=2 si m<=9.

Exemplul 1

numere.in	*numere.out*
1 4 23 4567 12 345	2

Explicație

Cerința este 1.

Observăm ca sunt două numere din șir care au două cifre (23 și 12).

Exemplul 2

numere.in	*numere.out*
2 3 10 23 4567 12345	4

Explicație

Cerința este 2.

Parcurgând numerele din șir, cifră cu cifră, de la stânga spre dreapta, întâlnim în ordine cifrele 2 3 4 5 6 7 1 2 3 4 5. Cea de-a 10-a cifră în acest șir este 4.

Exemplul 3

numere.in	*numere.out*
3 3 4 23 4507 12345	7123

Explicație

Cerința este 3.

Parcurgând numerele din șir, cifră cu cifră, de la stânga spre dreapta, se pot obține mai multe numere de cel mult 4 cifre, astfel: 2345, 3450, 4507, 5071, 712, 7123, 1234, 2345. Dintre acestea, cel mai mare număr de cel mult 4 cifre consecutive în șirul dat este: 7123.

Descrierea solutiei:

Pentru cerința 1: este suficient să parcurgem numerele din șir, să le citim și să verificăm dacă au două cifre.

Pentru cerința 2: se citesc cele n valori și pentru fiecare valoare se calculează câte cifre are. Se adună aceste valori, iar atunci când suma depășește k, se caută ultima cifră a numărului care completează suma până la k. Dacă nu există o astfel de valoare, se afișează "nu exista".

Pentru cerința 3: Vom forma, dacă este posibil, primul număr care are m cifre, parcurgând cifrele numerelor din șir, de la stânga la dreapta. Apoi, dacă este posibil, se adaugă următoarea cifră la sfârșitul numărului format anterior și se elimină cea mai din stânga cifră a numărului format anterior. Pentru fiecare număr construit se verifică dacă noul număr format este maximul.

O variantă de rezolvare este mai jos:

```
#include <fstream>
using namespace std;
ifstream cin("numere.in");
```

```
ofstream cout("numere.out");
int n, k, x, cx, cifra, cerinta, nrc, tc, poz,u,ok,p, pp, ctnr, ctcf,numar, maxnr,cp,m,
gasit;
int main()
{
  int i;
  cin>>cerinta;
  if(cerinta==1)
    {cin>>n;
     for(i=1;i<=n;i++)
       {cin>>x;
        if(x>=10 && x<=99) ctnr++;
       }
     cout<<ctnr<<'\n';
    }
  else
  if(cerinta==2)
    {cin>>n>>k;
     for(i=1;i<=n;i++)
       {
         cin>>x;
         cx=x;
         nrc=0;
         if(x==0) nrc=1;
         while(cx>0)
           {
             nrc++;
             cx=cx/10;
           }
         if(nrc+tc<k) tc=tc+nrc;
         else
           {k=tc+nrc-k+1;
            poz=0;
            while(poz<k)
              {  u=x%10;
                 poz++;
                 x=x/10;
              }
            cout<<u;
            ok=1;
            break;
           }

       }
```

```
  if(ok==0) cout<<"nu exista"<<'\n';
  }
else
  {
    cin>>n>>m;
    ctnr=0;
    ctcf=0;
    numar=0;
    maxnr=0;
    p=1;
    for(i=1;i<=m;i++) p=p*10;
    cp=p;
    while(ctcf<m && ctnr<=n)
      {cin>>x;
       cx=x;
       ctnr++;
       nrc=0;
       pp=1;
       if(x==0 && numar!=0) {nrc=1;pp=10;}
       else while(x>0)
            {nrc++;
             x=x/10;
             pp=pp*10;
            }
       if(ctcf+nrc<=m)   {numar=numar*pp+cx;
                          ctcf=ctcf+nrc;
                          p=p/pp;
                          }
       else {numar=numar*p+cx/(pp/p);
             ctcf=m;
             pp=pp/p;
             cx=cx%pp;
            }   }

    maxnr=numar;
    while(pp>0)
          {pp=pp/10;
           if(pp>0)
             {numar=numar%(cp/10)*10+cx/pp;
              cx=cx%pp;
              if(numar>maxnr) maxnr=numar;
             }
          }
   for(i=ctnr+1;i<=n;i++)
     {cin>>x;
```

```
            cx=x;
            nrc=0;
            pp=1;
            if(x==0) pp=10;
            else while(x>0)
                    {nrc++;
                     x=x/10;
                     pp=pp*10;
                    }

            while(pp>0)
                {pp=pp/10;
                 if(pp>0)
                    {numar=numar%(cp/10)*10+cx/pp;
                     cx=cx%pp;
                     if(numar>maxnr) maxnr=numar;
                    }
                }
             }
          cout<<maxnr<<'\n';
          }
       return 0;
}
```

CLASA a VI-a – PROBA PRACTICĂ

TIP SUBIECT – PROGRAMARE

PROBLEMA NUMERE

Time limit: 0.1s Memory limit: 64MB
Input: numere.in
Output: numere.out

Cerință

Micul Gates are de rezolvat o problemă la informatică. El primește un șir format din nnn numere naturale și trebuie să răspundă următoarelor cerințe:

4. Să se determine câte numere din șir au exact două cifre.
5. Să se determine a k-a cifră care apare în șirul de numere. Dacă nu există, se afișează mesajul *nu exista*. Numerotarea cifrelor se face de la stânga la dreapta, începând cu poziția 1.
6. Să se determine cel mai mare număr care se poate forma din cel mult m cifre consecutive aflate în numerele din șirul dat, parcurgând cifrele de la stânga la dreapta.

Date de intrare

Pe prima linie a fișierului de intrare numere.in se găsește numărul c al cerinței, care poate fi doar 1, 2 sau 3.
Dacă cerința este 1, pe a doua linie se găsește numărul natural nenul n iar pe a treia linie se găsesc n numere naturale, separate prin câte un spațiu.
Dacă cerința este 2, pe a doua linie se găsesc numerele n și k separate prin câte un spațiu, cu semnificația din enunț. Pe a treia linie se găsesc cele n numere naturale, separate prin câte un spațiu.
Dacă cerința este 3, pe a doua linie se găsesc numerele n și m separate prin câte un spațiu, cu semnificația din enunț. Pe a treia linie se găsesc cele n numere naturale, separate prin câte un spațiu.

Date de ieșire

Pe prima linie a fișierului de ieșire numere.out se va găsi valoarea determinată conform cerinței.

Restricții și precizări

- 1≤n≤100 0001;
- Numerele din șir sunt numere naturale cu cel mult 9 cifre fiecare.
- 1≤k≤900 000;
- 1≤m≤91;
- Pentru cerința 1, se acordă 16 puncte;
- Pentru cerința 2, se acordă 52 puncte (pentru 8 puncte, numerele din șir au o singură cifră);
- Pentru cerința 3, se acordă 32 puncte, dintre care:
    - m=1 pentru 4 puncte;
    - Pentru alte 28 puncte, m>=2 si m<=9.

Exemplul 1

numere.in	*numere.out*
1 4 23 4567 12 345	2

Explicație

Cerința este 1.

Observăm ca sunt două numere din șir care au două cifre (23 și 12).

Exemplul 2

numere.in	*numere.out*
2 3 10 23 4567 12345	4

Explicație

Cerința este 2.

Parcurgând numerele din șir, cifră cu cifră, de la stânga spre dreapta, întâlnim în ordine cifrele 2 3 4 5 6 7 1 2 3 4 5. Cea de-a 10-a cifră în acest șir este 4.

Exemplul 3

numere.in	*numere.out*
3 3 4 23 4507 12345	7123

Explicație

Cerința este 3.

Parcurgând numerele din șir, cifră cu cifră, de la stânga spre dreapta, se pot obține mai multe numere de cel mult 4 cifre, astfel: 2345, 3450, 4507, 5071, 712, 7123, 1234, 2345. Dintre acestea, cel mai mare număr de cel mult 4 cifre consecutive în șirul dat este: 7123.

Descrierea solutiei:

Pentru cerința 1: este suficient să parcurgem numerele din șir, să le citim și să verificăm dacă au două cifre.

Pentru cerința 2: se citesc cele n valori și pentru fiecare valoare se calculează câte cifre are. Se adună aceste valori, iar atunci când suma depășește k, se caută ultima cifră a numărului care completează suma până la k. Dacă nu există o astfel de valoare, se afișează "nu exista".

Pentru cerința 3: Vom forma, dacă este posibil, primul număr care are m cifre, parcurgând cifrele numerelor din șir, de la stânga la dreapta. Apoi, dacă este posibil, se adaugă următoarea cifră la sfârșitul numărului format anterior și se elimină cea mai din stânga cifră a numărului format anterior. Pentru fiecare număr construit se verifică dacă noul număr format este maximul.

O variantă de rezolvare este mai jos:

```
#include <fstream>
using namespace std;
ifstream cin("numere.in");
ofstream cout("numere.out");
int n, k, x, cx, cifra, cerinta, nrc, tc, poz,u,ok,p, pp, ctnr, ctcf,numar, maxnr,cp,m,
gasit;
int main()
{
  int i;
  cin>>cerinta;
  if(cerinta==1)
    {cin>>n;
     for(i=1;i<=n;i++)
       {cin>>x;
        if(x>=10 && x<=99) ctnr++;
       }
    cout<<ctnr<<'\n';
    }
  else
  if(cerinta==2)
    {cin>>n>>k;
     for(i=1;i<=n;i++)
       {
          cin>>x;
          cx=x;
          nrc=0;
          if(x==0) nrc=1;
          while(cx>0)
            {
              nrc++;
              cx=cx/10;
            }
          if(nrc+tc<k) tc=tc+nrc;
          else
            {k=tc+nrc-k+1;
             poz=0;
             while(poz<k)
               {  u=x%10;
                  poz++;
                  x=x/10;
               }
             cout<<u;
             ok=1;
```

```
            break;
           }

     }

   if(ok==0) cout<<"nu exista"<<'\n';
   }
else
   {
     cin>>n>>m;
     ctnr=0;
     ctcf=0;
     numar=0;
     maxnr=0;
     p=1;
     for(i=1;i<=m;i++) p=p*10;
     cp=p;
     while(ctcf<m && ctnr<=n)
        {cin>>x;
         cx=x;
         ctnr++;
         nrc=0;
         pp=1;
         if(x==0 && numar!=0) {nrc=1;pp=10;}
         else while(x>0)
               {nrc++;
                x=x/10;
                pp=pp*10;
               }
         if(ctcf+nrc<=m)   {numar=numar*pp+cx;
                            ctcf=ctcf+nrc;
                            p=p/pp;
                            }
         else {numar=numar*p+cx/(pp/p);
               ctcf=m;
               pp=pp/p;
               cx=cx%pp;
              }   }

     maxnr=numar;
     while(pp>0)
           {pp=pp/10;
            if(pp>0)
              {numar=numar%(cp/10)*10+cx/pp;
               cx=cx%pp;
```

```
                    if(numar>maxnr) maxnr=numar;
                   }
                }
          for(i=ctnr+1;i<=n;i++)
            {cin>>x;
             cx=x;
             nrc=0;
             pp=1;
             if(x==0) pp=10;
             else while(x>0)
                    {nrc++;
                     x=x/10;
                     pp=pp*10;
                    }

             while(pp>0)
                 {pp=pp/10;
                  if(pp>0)
                     {numar=numar%(cp/10)*10+cx/pp;
                      cx=cx%pp;
                      if(numar>maxnr) maxnr=numar;
                     }
                 }
           }
        cout<<maxnr<<'\n';
        }
    return 0;
}
```

PROBLEMA SOPHIE

Limită de timp: 1s Limită de memorie: 64MB
Intrare: sophie.in
Ieșire: sophie.out

Cerință

Un număr i este numit număr Sophie Germain dacă i este prim și 2·i+1 este, de asemenea, prim. Se poate generaliza acest concept astfel: i este considerat număr k-Sophie Germain dacă i este prim și i·k+1 este, de asemenea, prim, pentru un număr k natural nenul, k>2.

1. Dându-se un număr natural nenul n, să se calculeze câte numere Sophie Germain există între 1 și n.
2. Dându-se două numere naturale nenule n și k, să se calculeze câte numere k-Sophie Germain există între 1 si n.

Date de intrare

Pe prima linie a fișierului de intrare sophie.in se găsește un număr natural c, care poate să fie doar 1 sau 2, reprezentând numărul cerinței de rezolvat.
Dacă cerința este c=1, pe a doua linie se găsește un număr natural nenul n, cu semnificația din enunț.
Dacă cerința este c=2, pe a doua linie se găsesc două numere naturale nenule n și k, separate printr-un spațiu, cu semnificația din enunț.

Date de ieșire

În fișierul sophie.out se va afișa un număr natural care reprezintă numărul cerut.

Restricții și precizări

- 1≤n,k≤106;
- Pentru prima cerință (c=1) se acordă 40 puncte.
- Pentru a doua cerință (c=2) se acordă 60 puncte. Dintre acestea, pentru 15 puncte, k este impar. Pentru alte 15 puncte, $n \cdot k \leq 10^6$.

Exemplul 1

sophie.in	*sophie.out*
1 10	3

Explicație Cerința este 1.
Numerele Sophie Germain între 1 și 10 sunt:

- 2: 2 este prim, iar 2·2+1=5, 5 este prim.
- 3: 3 este prim, 3·2+1=7, 7este prim.
- 5: 5 este prim, iar 5·2+1=11, 11 este prim.

Exemplul 2

sophie.in	*sophie.out*
2 10 20	3

Explicație Cerința este 2.
Singurele numere care respectă condiția între 1 și 10 sunt:

- 2: 2 este prim, iar 2·2+1=5, 5 este prim.
- 3: 3 este prim, 3·2+1=7, 7este prim.
- 5: 5 este prim, iar 5·2+1=11, 11 este prim.

Descrierea soluției:

Pentru a rezolva problema legată de numerele Sophie Germain și k-Sophie Germain, trebuie să parcurgem următorii pași:

Folosim Ciurul lui Eratostene pentru a determina toate numerele prime până la n, sau 2 * n + 1.
Pentru cerința 1, verificarea condiției Sophie Germain:

- Pentru fiecare număr prim i în intervalul [1, n], verificăm dacă 2 * i + 1 este și el prim.
- Dacă ambele condiții sunt îndeplinite, numărul i este un număr Sophie Germain.

Pentru cerința 2, verificarea condiției k-Sophie Germain:

- Pentru fiecare număr prim i în intervalul [1, n], verificăm dacă k * i + 1 este și el prim. Pentru aceasta verificare, deoarece atât k și i sunt mari, verificarea primalității nu se poate face direct prin ciurul lui Eratostene, dar se pot folosi numerele prime extrase de acolo.
- Dacă ambele condiții sunt îndeplinite, numărul i este un număr k-Sophie Germain.

Ciurul lui Eratostene este o metodă eficientă pentru a genera numere prime și este potrivită pentru intervale mari de numere. De asemenea, problema necesita cateva optimizări ale ciurului pentru 100 de puncte.

VARIANTA DE PUNCTAJ MAXIM (100 PNCT)

```
#include <fstream>
using namespace std;
ifstream f("sophie.in");
ofstream g("sophie.out");
int n, k,cnt, cerinta, m;
const int MAX=2000005;
bool ciur[MAX],ok;
int prim[MAX];
int i,j;
int main()
{f>>cerinta;
 ciur[0]=ciur[1]=1;
 for(i=2;i*i<MAX;i++)
   if(ciur[i]==0)
      for (j=i*i; j<MAX; j+=i) ciur[j]=1;
 if(cerinta==1)
   {f>>n;
    for (int i=2;i<=n; i++)
      if (ciur[i]==0 && ciur[i*2+1]==0) cnt++;
    g<<cnt;
   }
```

```
else
   {f>>n>>k;
    m=0;
    for(i=2;i<MAX;i++)
     {if(ciur[i]==0)
        {prim[m]=i;
         m++;
         }
     }
   for(i=2;i<=n;i++)
      {if(ciur[i]==0)
           {long long pereche=1LL*i*k+1;
            ok=0;
            if(pereche<MAX)
              {if(ciur[pereche]==1) ok=1;} ///se gaseste perechea direct in vectorul de
numere prime mici
            else
              {
              for(j=0; j<m && 1LL*prim[j]*prim[j]<=pereche;j++) ///verific daca i*k+1 este
nr prim
                   if(pereche%prim[j]==0) {ok=1; break;}
              }
            if(ok==0) {///g<<i<<" ";
                      cnt++;
                      }
            }
      }
    g<<cnt;
   }
  return 0;
}
```

VARIANTA PUNCTAJ PARTIAL (75PNCT)

```
#include <fstream>
using namespace std;
ifstream f("sophie.in");
ofstream g("sophie.out");
bool Prim(long long x)
{
   if (x<2) return 0;
   if (x==2) return 1;
   if (x%2==0) return 0;
   for (int d=3;d*d<=x;d=d+2)
      if (x%d==0) return 0;
   return 1;
}
int n, k,cnt, cerinta;
int main()
{f>>cerinta;
 if(cerinta==1)
   {f>>n;
    for (int i=2;i<=n; i++)
      if (Prim(i) && Prim(i*2+1)) cnt++;
    g<<cnt;
   }
 else
    {f>>n>>k;
     for (int i=2;i<=n; i++)
       if (Prim(i) && Prim(1LL*i*k+1)) cnt++;
     g<<cnt;
    }
   return 0;
}
```

CLASA a VII-a și CLASA a VIII-a – PROBA PRACTICĂ

TIP SUBIECT – PROGRAMARE

PROBLEMA JOC

Limită de timp: 1s
Limită de memorie: 64MB
Intrare: joc.in Ieșire: joc.out

Cerință

Micul Gates este din ce în ce mai priceput în ceea ce privește numerele prime, astfel încât domnul profesor de matematică îi propune următorul joc: se consideră un număr construit prin alipirea primelor k numere prime, în ordinea crescătoare a acestora. Ajută-l tu să câștige jocul, rezolvând următoarele cerințe:

1. Câte numere impare au fost alipite șirului?
2. Care este suma numerelor care au fost alipite?
3. Se numerotează pozițiile cifrelor numărului format din alipirea celor prime, începând de la 1. Pentru p poziții date de profesor, Micul Gates trebuie să determine numărul prim din care provine cifra aflată pe fiecare dintre aceste poziții.

Date de intrare

Pe prima linie a fișierului de intrare joc.in se găsește numărul c, care poate fi doar 1, 2 sau 3 și corespunde cerinței care trebuie rezolvată.
Dacă cerința este 1 sau 2, pe a doua linie din fișierul joc.in se află un număr natural nenul k, reprezentând numărul de numere prime care sunt alipite.
Dacă cerința este 3, pe a doua linie se află două numere naturale nenule, k– numărul de numere prime care vor fi alipite și p – numărul de poziții date de profesor, separate prin câte un spațiu. Pe a treia linie se află p numere naturale reprezentând câte o poziție poz în numărul format prin alipirea celor k numere prime.

Date de ieșire

Dacă cerința este c=1, pe prima linie din fișierul joc.out se găsește un număr corespunzător răspunsului la cerința 1.
Dacă cerința este c=2, pe prima linie din fișierul joc.out se găsește un număr corespunzător răspunsului la cerința 2.
Dacă cerința este c=3, fișierul joc.out va conține pe prima linie p numere separate prin câte un spațiu, fiecare număr reprezentând numărul prim din care provine cifra aflată la poziția poz, conform cerinței.

Restricții și precizări

- $1 \le k < 50\,000$;
- $1 \le p \le 100\,000$;
- $1 \le poz \le$ lungimea numărului format;
- Pentru 15 puncte, cerința este c=1;

- Pentru alte 25 de puncte, cerința este c=2;
- Pentru alte 60 de puncte, cerința este c=3. Dintre acestea, pentru 30 de puncte k≤1000;
- Se garantează că există mereu o soluție

Exemplul 1

joc.in	*joc.out*
1 3	2

Explicație

Cerința este 1, k=3, deci se alipesc primele 3 numere prime (2, 3 și 5). Dintre acestea, două sunt impare.

Exemplul 2

joc.in	*joc.out*
2 3	10

Explicație

Cerința este 2, k=3, deci se alipesc primele 3 numere prime (2, 3 și 5). Suma lor este: 2+3+5=10

Exemplul 3

joc.in	joc.out
3 5 3 1 5 6	2 11 11

Explicație

Cerința este 3, k=5, p=3.
Primele 5 numere prime sunt: 2, 3, 5, 7, 11.
Numărul construit este: 235711.

- Cifra de pe poziția 1 este 2, care provine din 2.
- Cifra de pe poziția 5 este 1, care provine din 11.
- Cifra de pe poziția 6 este 1 și provine din 11.

Exemplul 4

joc.in	*joc.out*
3 6 2 8 3	13 5

Explicație

Cerința este 3, k=6, p=2.
Primele 6 numere prime sunt: 2, 3, 5, 7, 11 și 13.
Numărul construit este: 23571113.

- Cifra de pe poziția 8 este 3, care provine din 13.
- Cifra de pe poziția 3 este 5, care provine din 5.

Descrierea soluției:

Cerința 1 :

La această cerință, trebuie să determinăm câte numere impare au fost folosite la crearea șirului. Toate numerele prime, cu excepția lui **2**, sunt impare.

- Astfel, dintre primele k numere prime, k - 1 dintre ele sunt impare.
  Complexitate timp: O(1), deoarece nu facem nicio iterație.
  Complexitate memorie: O(1).

Cerința 2 :

Aici trebuie să calculăm suma primelor k numere prime.

- Folosim un vector în care reținem primele k numere prime, generat anterior cu Ciurul lui Eratostene.
- Iterăm prin primele k elemente și le adunăm într-o variabilă sum de tip long long, pentru a evita depășirea limitei maxime a tipului int.
  Complexitate timp: O(k * log log (k)), deoarece trebuie să aflăm primele k numere prime, k<=50000,
  Complexitate memorie: O(k).

Cerința 3 :

În această cerință trebuie să determinăm, pentru p poziții date, numărul prim din care provine cifra de la fiecare poziție.

- Folosind vectorul cu numere prime pe care l-am folosit și la cerința 2, parcurgem primele k numere prime și stocăm pentru fiecare cifră numărul prim corespunzător fiecărei poziții.
- Pentru fiecare poziție poz citită, se afișează numărul prim din care provine cifra respectivă.
- Deoarece lungimea șirului este limitată (fiecare număr prim are maxim 5-6 cifre), operațiile sunt eficiente.
  Complexitate timp: O(k * log log (k) + p)
  Complexitate memorie: O(k)

O varianta de rezolvare:

Problema JOC

```
#include <bits/stdc++.h>
#define nmx 700000
using namespace std;
ifstream f ("joc.in");
ofstream g ("joc.out");
int c,k,p,pr[50005],leng[300005];
bool er[nmx+5];
```

```
void ciur()
{
   er[0]=er[1]=1;
   for (int i=4; i<=nmx; i+=2)
      er[i]=1;
   for (int i=3; i*i<=nmx; i+=2)
      if (!er[i])
         for (int j=i*i; j<=nmx; j+=2*i)
            er[j]=1;
}
int main()
{  f>>c>>k;
   if (c==1)
   {
      g<<k-1<<'\n';
      return 0;
   }
   ciur();
   int ct=0;
   for (int i=1; i<=nmx; i++)
      if (er[i]==0)
      {
         pr[++ct]=i;
         if (ct>=k) break;
      }
   if (c==2)
   {
      long long sum=0;

      for (int i=1; i<=k; i++)
         sum+=1LL*pr[i];

      g<<sum<<'\n';
      return 0;
   }
   if (c==3)
   {
      int x,poz;
      ct=0;
      for (int i=1; i<=k; i++)
      {
         x=pr[i];
         while (x)
         {
            leng[++ct]=pr[i];
```

```
            x/=10;
          }
        }
        f>>p;
        for (int i=1; i<=p; i++)
        {
          f>>poz;
          g<<leng[poz]<<' ';
        }
        g<<'\n';
        return 0;
    }
}
```

PROBLEMA STALPI

Limită de timp: 1s
Limită de memorie: 64MB
Intrare: stalpi.inIeșire: stalpi.out

Cerință

În ciudatul oraș al Micului Gates, primăria a instalat pe o stradă n stâlpi pentru iluminat, dispusi în linie, la distanțe egale între ei, de 1 metru. Fiecare stâlp este identificat unic prin poziția sa în șir, de la 1 la n. Primul stâlp se află la poziția 1 (la începutul străzii), al doilea stâlp la poziția 2 și așa mai departe, până la poziția n, unde se află ultimul stâlp (unde se termină strada). Ciudățenia constă în faptul că stâlpii au înălțimi diferite, fiecare stâlp asigurând o iluminare a zonei în care se află pe o distanță egală în stânga și în dreapta sa cu înălțimea stâlpului. De exemplu, dacă un stâlp are înălțimea x, el va asigura iluminarea pe o distanță egală cu x metri în stânga și x metri în dreapta sa (inclusiv poziția pe care se află). Înălțimea fiecărui stâlp este cunoscută.

Micul Gates primește de la primărie sarcina de a răspunde următoarelor cerințe:

1. Care este cea mai mare diferență de înălțime între doi stâlpi consecutivi în șir?
2. Care este numărul minim de stâlpi care trebuie păstrați, astfel încât strada să fie complet iluminată?

Date de intrare

Fișierul de intrare stalpi.in conține:

- pe prima linie o valoare 1 sau 2 reprezentând numărul cerinței care trebuie rezolvată;
- pe a doua linie un număr întreg n, reprezentând numărul de stâlpi;
- pe a treia linie n numere naturale nenule separate printr-un spațiu, unde al i-lea număr reprezintă înălțimea stâlpului de la poziția i.

Date de ieșire

Pe prima linie a fișierului de ieșire stalpi.out se va găsi un singur număr reprezentând răspunsul în conformitate cu cerința care trebuie rezolvată.

Restricții și precizări

- $2 \leq n \leq 10^6$;
- Pentru cerința 1 se acordă 32 de puncte;
- Pentru cerința 2 se acordă 68 de puncte. Dintre acestea, pentru 24 de puncte, $n \leq 10^3$;
- Pentru toate testele, $1 \leq h_i \leq 10^3$ pentru oricare i între 1 și n.

Exemplul 1

stalpi.in	*stalpi.out*
1 3 5 10 8	5

Explicație: Diferența maximă dintre doi stâlpi vecini este 5 (dintre stâlpul 1 si stâlpul 2).

Exemplul 2

stalpi.in	*stalpi.out*
2 10 1 2 1 1 3 1 2 1 1 1	3

Explicație
Cel mai mic număr de stâlpi de care avem nevoie este 3.
1 **2** 1 1 **3** 1 2 1 1 **1** (stâlpul de pe poziția 2, cel de pe poziția 5 și cel de pe poziția 10)

Exemplul 3

stalpi.in	*stalpi.out*
2 10 1 1 1 1 1 1 1 1 1 1	4

Explicație: pentru șirul 1 1 1 1 1 1 1 1 1 1, avem nevoie de 4 stâlpi.
Primul stâlp ales acoperă pozițiile de la 1 la 3, inclusiv acestea.
Al doilea stâlp ales acoperă pozițiile de la 4 la 6, inclusiv acestea.
Al treilea stâlp ales acoperă pozițiile de la 7 la 9, inclusiv acestea.
Al patrulea stâlp ales acoperă pozițiile de la 9 la 10, inclusiv acestea.

Descrierea soluției:

Cerința 1: Pentru primul subpunct, se reține numerele în vectorul h[i] și se calculează maximul dintre două valori successive în vector, h[i + 1] , h[i]), pentru oricare 2 <= i <= n;

- **Complexitate timp:** O(n)
- **Complexitate memorie:** O(n).

Cerința 2 :

Solutie posibilă 1 :

- Pentru al doilea subpunct, pornim de la primul stâlp și încercăm să acoperim întreaga linie folosind cât mai puțini stâlpi. Pornim un proces în care extindem acoperirea.
- La fiecare pas, considerăm ca am acoperit de la 1 pana la R. Încercăm să extindem acoperirea cât mai mult posibil înainte de a adăuga un alt stâlp.
- Intervalul acoperit de stâlpul i, cu înălțimea h[i] este intervalul închis [i - h[i], i + h[i]]
- Căutăm următorul stâlp din zona lui R astfel încât, dacă acest stâlp acoperă intervalul închis, capătul sau din dreapta este cat mai îndepărtat.
- Deoarece h[i] <= 1000, nu este nevoie sa verificam foarte mulți stâlpi candidat.

Solutie posibila 2:

- Intervalul acoperit de stâlpul i, cu inaltime h[i] este [i - h[i], i + h[i]]
- Sortam după capătul din stânga al intervalului : i - h[i].
- Zona acoperită este de la 1 până k + h[k], unde k este ultimul interval ales.. Căutăm intervalul j > k, astfel incat j - h[j] <= h[k] si j + h[j]maximal.
- **Complexitate timp:** O(n * log(n))
- **Complexitate memorie:** O(n)
- Se poate folosi o structura avansată de date pentru a găsi cel mai bun candidat din toți j + h[j] în aceeași complexitate

O soluție posibilă poate fi:

```
#include <fstream>
#include <iostream>
#include <cmath>
using namespace std;
int n, i, c, x, y, dr, st, ct, mx, p;
struct stalp
{
   int st, dr;
}v[1000001];

ifstream f("stalpi.in");
ofstream g("stalpi.out");
int main()
{
   f >> c;
   if(c == 1)
   {
      f >> n;
      f >> y;
      for(i = 2; i <= n; i ++)
```

```
    {
        f >> x;
        mx = max(mx, abs(x - y));
        y = x;
    }
    g << mx;
  }
  else
  {
    f >> n;
    for(i = 1; i <= n; i ++)
    {
        f >> x;
        v[i].st = i - x;
        v[i].dr = i + x;
    }

    st = -1;
    while(st < n - 1)
    {
        mx = 0;
        if(v[st].dr >= n) break;
        for(i = p; i <= min(n, p + 1002); i ++)
            if(v[i].st <= v[st].dr || v[i].st == v[st].dr + 1)
                if(v[i].dr >= mx) mx = v[i].dr, p = i;
        st = p;
        ct ++;
    }
    g << ct;
  }
  return 0;
}
```