Zadaci sa rešenjima Okružno takmičenje 2007.

zadatak: Vlada

U dalekoj zemlji Bajtoviji, održali su se redovni parlamentarni izbori. Nakon što je izborna komisija proglasila rezultate, potrebno je formirati vladu. Svakoj od stranaka koje su prešle cenzus je dodeljen odredeni broj mesta za poslanike u parlamentu. Vladu može formirati jedna stranka ili koalicija stranaka, s tim što ukupan broj poslanika koje stranka, odnosno koalicija, poseduje mora biti viši od polovine svih mesta u parlamentu. Kompanija koja se bavi predviđanjima političkih dogadaja želi da zna na koliko se načina može formirati postizborna koalicija.

Ulaz:

(Ulazni podaci se nalaze u datoteci **vlada.in**) U prvom redu datoteke se nalazi prirodan broj *N*, broj stranaka koje su prešle cenzus. U narednih *N* redova se nalazi *N* brojeva (u svakom redu po jedan), svaki pokazuje koliko je koja stranka dobila mesta u skupštini

Izlaz:

(Izlazne podatke upisati u datoteku **vlada.out**) U prvi i jedini red datoteke treba ispisati broj načina da se formira vladajuća koalicija.

Ograničenja:

- $1 \leq N \leq 20$
- brojevi mesta koje su stranke dobile u parlamentu nisu veći od 200,
- vremensko ograničenje za izvršavanje programa je 1 s.

Primer 1:

vlada.in	**vlada.out**
4 10 20 30 40	7

Primer 2:

vlada.in	**vlada.out**
2 50 51	

fajl: vlada.cpp

```
#include <cstdlib>
#include <fstream>
#include <iostream>

using namespace std;

ifstream fin("vlada.p9.in");
ofstream fout("vlada.p9.out");
int n;
const int MAXN = 20;
int p[MAXN];
bool in[MAXN];
int sum;

void input() {
     fin >> n;
     sum = 0;
     for (int i = 0; i < n; i++) {
         fin >> p[i];
         sum += p[i];
     }
}
long solve(int s) {
```

```
     if (s < n) {
                 in[s] = false;
                 long total = solve(s+1);
                 in[s] = true;
                 total += solve(s+1);
                 return total;
     }
     int sum1 = 0;
     for (int i = 0; i < n; i++)
         if (in[i])
            sum1 += p[i];
     if (sum1 > sum/2)
        return 1;
     else
         return 0;
}
int main(int argc, char *argv[])
{
    input();
    for (int i = 0; i < n; i++)
        in[i] = false;
    long res = solve(0);
    cout << n;
    fout << res << "\n";
    fout.close();
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

zadatak: Birački spisak

Izborna komisije opštine Bajtograd je dobila dopis od Republičke Izborne Komisije o novom načinu rasporedivanja birača, po kome birači dobijaju svoj broj koji odredjuje biračko mesto na osnovu njihove pozicije u sortiranom biračkom spisku za celu opštinu. Nažalost, ovaj dopis im je stigao nedelju dana pre izbora nakon što su oni već poslali pozive na glasanje sa brojevima dodeljenim po starom spisku koji nije bio sortiran već su birači dopisivani i brisani onim redom kojim su dobijali i gubili biračko pravo. Da bi uštedeli, članovi OIK/a su se setili da ne moraju svima da pošalju nove pozive već samo onim biračima čiji se broj u starom i novom spisku razlikuje. Pomozite im da izračunaju koliko poziva treba da pošalju.

Ulaz:

(Ulazni podaci se nalaze u datoteci **spisak.in**) Ulaz predstavlja stari spisak birača. U prvom redu se nalazi broj birača n, a u sledećih n redova se nalaze imena i prezimena birača, po jedno u svakom redu. Biraču u i/tom redu dodeljen je broj i. Imena i prezimena su sastavljena od malih slova engleskog alfabeta i razdvojeni tačno jednim razmakom.

Izlaz:

(Izlazne podatke upisati u datoteku **spisak.out**) U prvom redu izlazne datoteke ispisati broj birača kojima treba poslati nove pozive.

Ograničenja:

- broj birača je manji od 1000,
- dužina imena i prezimena je manja od 25 slova,
- vremensko ograničenje za izvršavanje programa je 1 s.

Napomena:

Novi spisak je sortiran leksikografski, prvo po imenima pa po prezimenima. Ako su A i B neke dve osobe sa spiska onda se odredjuje prvo mesto na kome se razlikuju njihova imena (odnosno prvo mesto na kome se razlikuju prezimena, ako su imena ista). Ako su s_A i s_B slova koja se nalaze na tim mestima u rečima *A* i *B*, redom, onda se proveri koje je od ta dva slova pre u engleskom alfabetu. Za onu reč u kojoj se nalazi slovo koje je pre u alfabetu, kažemo da je manja i stoji pre druge (veće) reči u sortiranom spisku. Dozvoljeno je postojanje više osoba sa istim imenom i prezimenom.

Primer 1:

spisak.in	**spisak.out**
4	2

```
ana popovic
nikola peric
nemanja tomic
marko djordjevic
```

Primer 2:

spisak.in	**spisak.out**
9 ana popovic nemanja tomic marko djordjevic nikola peric marko djordjevic ana popovic nikola peric sanja popovic marko djordjevic	5

Objašnjenje:

Birači kojima se ne šalju novi pozivi su: `ana popovic` (broj 1), `marko djordjevic` (broj 3), `marko djordjevic` (broj 5) i `nikola peric` (broj 7). Primetimo da bi ovaj spisak nakon leksikografskog uređivanja imao sledeći izgled: `ana popovic`, `ana popovic`, `marko djordjevic`, `marko dordjevic`, `marko djordjevic`, `nemanja tomic`, `nikola peric`, `nikola peric`, `sanja popovic`.

fajl: spisak.cpp,

```
#include <stdio.h>
#include <string.h>

bool less(char *a,char *b)
{
  int i=0;
  while ((a[i]!=0) && (b[i]!=0) && (a[i]==b[i]))
    i++;
  return a[i]<b[i];
}
void sort(char **a, int n)
{
  int i;
  char temp[30];
  bool ok = false;
  while (!ok)
  {
    ok=true;
    for (i=0;i<n-1;i++)
      if (less(a[i+1],a[i]))
      {
        ok=false;
        strcpy(temp,a[i]);
        strcpy(a[i],a[i+1]);
        strcpy(a[i+1],temp);
      }
}
}
bool equal(char *a,char *b)
{
  int i=0;
  while ((a[i]!=0) && (b[i]!=0) && (a[i]==b[i]))
    i++;
  return a[i]==b[i];
}
int count(char **a,char **b,int n)
{
  int k=0,i;
  for (i=0;i<n;i++)
```

```
    if (equal(a[i],b[i]))
      k++;
  return k;
}
int main()
{
  FILE *f;
  char **a,**b;
  a = new char*[1000];
  for (int i=0;i<1000;i++) a[i] = new char[30];
  b = new char*[1000];
  for (int i=0;i<1000;i++) b[i] = new char[30];
  int i,n;
  f = fopen("spisak.in","r");
  fscanf(f,"%d",&n);
  fgets(a[0],30,f);
  for (i=0;i<n;i++)
    fgets(a[i],30,f);
  fclose(f);
  for (i=0;i<n;i++)
    strcpy(b[i],a[i]);
  sort(b,n);
  f = fopen("spisak.out","w");
  fprintf(f,"%d\n",n-count(a,b,n));
  fclose(f);
  return 0;
}
```

zadatak: Gaus

Profesor Đurić u poslednje vreme ima velikih problema sa svojim nestašnim đacima. Da bi ih smirio, odlučio je da im da, na prvi pogled, mnogo težak zadatak. U prošlosti su se često primenjivale takve mere smirivanja, i omiljen zadatak je bio sabiranje prvih 1000 brojeva. *A*li, otkako je mali Gaus našao način da brzo izračuna taj zbir, profesori su morali da promene zadatak. I tako je prof. Đurić smislio sledeće: daće đacima nenegativne cele brojeve A i B, i tražiće od njih da mu kažu koliko brojeva iz intervala $[A,B]$ (tj. svi brojevi veći ili jednaki od A i manji ili jednaki od B) ima paran zbir cifara (npr, zbir cifara broja 1234 je 1 + 2 + 3 + 4 = 10, dakle paran broj). Međutim, među đacima se nalazi i mali Draganče koji, poput malog Gausa, želi da što pre reši taj zadatak i nastavi da pravi probleme prof. Đuriću. Kako Draganče nije uspeo da nađe rešenje zadatka, pomoć je potražio od njegovih drugova, mladih programera.

Ulaz:
(Ulazni podaci se nalaze u datoteci **gaus.in**) U prvom redu tekstualne datoteke zapisani su brojevi A i B, odvojeni jednim razmakom.

Izlaz:
(Izlazne podatke upisati u datoteku **gaus.out**) U izlaznu datoteku ispisati samo jedan broj - koliko ima brojeva iz intervala $[A,B]$ takvih da im je zbir cifara paran broj.

Ograničenja:

- $0 \leq A \leq B \leq 2^{30}$
- vremensko ograničenje za izvršavanje programa je 1 s.

Primer 1:

gaus.in	**gaus.out**
5 15	5

Primer 2:

gaus.in	**gaus.out**
16 20	3

fajl: gaus.cpp

```
// Gaus, Okruzno 2007
#include <cstdio>
```

```
#include <vector>
using namespace std;

vector<int> cifra;

long count(long n, bool paran, bool manje){
  if (n==-1){
    if (paran) return 1;
    else return 0;
  }
  else{
    long ret=0;

    if (manje){
      ret += 5*count(n-1,paran,true) + 5*count(n-1,!paran,true);
    }
    else{
      if (cifra[n]%2==0) ret += count(n-1, paran, false);
      else ret += count(n-1,!paran,false);

      if (cifra[n]>0){
        int p=1 + (cifra[n]-1)/2;
        int np=(1 + cifra[n]-1)/2;

        ret += p * count(n-1,paran,true);
        ret += np * count(n-1,!paran,true);
      }
    }

    return ret;
  }
}


int main(){
  long A,B;

  FILE *f;
  f=fopen("gaus.10.in","r");
  fscanf(f,"%ld %ld",&A,&B);
  fclose(f);

  long res=0;

  while (B>0){
    cifra.push_back(B%10);
    B/=10;
  }
  res+=count(cifra.size()-1,true,false);

  cifra.clear();
  A--;
  while (A>0){
    cifra.push_back(A%10);
    A/=10;
  }
  res-=count(cifra.size()-1,true,false);


  f=fopen("gaus.10.out","w");
  fprintf(f,"%ld\n",res);
  fclose(f);

  return 0;
}
```

zadatak: Zecovi

Na poljani se nalazi *n* zečeva i *n* zečica. Svi oni stoje duž jedne linije tako da svi zečevi gledaju desno niz liniju, a sve zečice levo niz liniju. Oni bi želeli da se podele u parove tako da u svakom paru bude jedna zečica i jedan zec koji mogu međusobno da se vide. Zec vidi zečicu ako je ona bilo gde desno od njega, a zečica vidi zeca ako je on bilo gde levo od nje. Od vas se traži da izbrojte na koliko načina oni to mogu uraditi tako da se svaki zec odnosno zečica nađe u tačno jednom paru. Pošto taj broj može biti veoma velik, nađite samo koliki ostatak daje taj broj pri deljenju sa 10007.

Ulaz:
(Ulazni podaci se nalaze u datoteci **zecovi.in**) U prvom redu zapisan je broj *n*. U drugom redu zapisano je tačno 2n simbola (bez razmaka) od kojih ima *n* znakova > koji označavaju zečeve (gledaju desno) i *n* znakova < koji označavaju zečice (gledaju levo).

Izlaz:
(Izlazne podatke upisati u datoteku **zecovi.out**) U prvom redu ispisati samo jedan broj - ostatak pri deljenju sa 10007 broja mogućih načina da se zečevi podele u parove.

Ograničenja:

- $1 \leq n \leq 100000$
- vremensko ograničenje za izvršavanje programa je 1 s.

Primer 1:

zecovi.in	zecovi.out
3 >><>><	4

Objašnjenje:
Ako zečeve i zečice označimo brojem koji odgovara poziciji na kojoj se nalaze, četiri moguća načina formiranja parova su:
{(1, 6), (2, 3), (4, 5)},
{(1, 3), (2, 5), (4, 6)},
{(1, 3), (2, 6), (4, 5)} i
{(1, 5), (2, 3), (4, 6)}

Primer 2:

zecovi.in	zecovi.out
8 >>>>>>><<<<<<<<	292

Objašnjenje:
Ukupan broj načina je 40320, što pri deljenju sa 10007 daje ostatak 292

fajl: zecovi.pas

```
const
  fin = 'zecovi.in';
  fout = 'zecovi.out';


var
  res : Longint;


  procedure ReadInputAndSolve();
  var
    f : Text;
    n, i, z : Longint;
    ch : Char;
  begin
    Assign(f, fin);
    Reset(f);
```

```
    Readln(f, n);

    res := 1;
    z := 0;
    for i := 1 to 2*n do
    begin
      Read(f, ch);
      if ch = '>' then
        Inc(z)
      else
      begin
        res := (res * z) mod 10007;
        Dec(z);
      end;
    end;

    Close(f);
  end;


  procedure WriteOutput();
  var
    f : Text;
  begin
    Assign(f, fout);
    Rewrite(f);
    Writeln(f, res);
    Close(f);
  end;


begin
  ReadInputAndSolve();
  WriteOutput;
end.
```

zadatak: Filmovi

Kako se mali Draganče iz trećeg zadatka nije proslavio u matematici i programiranju, roditelji su mu smanjili džeparac. Zato je on rešio da iskoristi svoju veliku kolekciju DVD filmova i odlučio da prodaje filmove po niskim cenama. Svaki film se nalazi na jednom DVD-u, a na Dragančetovom hard disku može stati najviše k filmova. Procedura rezanja je sledeća: ukoliko se traženi film nalazi na hard disku, Draganče odmah počinje sa rezanjem; u suprotnom on mora da nađe odgovarajući DVD i presnimi ga na hard disk. Ako na disku nema slobodnog prostora, on mora da obriše neki film. Traženje DVD-a i presnimavanje iziskuje puno vremena, i zato Draganče želi da smanji taj broj. Na početku je njegov disk prazan.
On je napravio spisak poručenih filmova i zna tačno redosled n kupaca koji dolaze da nasnime omiljeni film. Draganče je uspeo da minimizira broj prebacivanja filmova na HDD (a samim tim i čekanje kupaca). Da li i vi možete da izračunate koliko će najmanje puta Draganče ipak morati da presnimi neki film na hard disk?
Ulaz:
(Ulazni podaci se nalaze u datoteci **filmovi.in**) U prvom redu datoteke se nalaze dva prirodna broja n i k. Broj n predstavlja broj naručenih filmova, a broj k je broj filmova koji može da stane na disku. U sledećih n redova nalaze se redni brojevi filmova $a[i]$ koje kupci uzimaju, poređani po vremenu dolaska
Izlaz:
(Izlazne podatke upisati u datoteku **filmovi.out**) U prvom i jedinom redu štampati minimalan broj presnimavanja DVD-a na hard disk.
Ograničenja:

- $1 \leq n \leq 10000$
- $1 \leq k \leq 500$
- $1 \leq a[i] \leq 10000$
- vremensko ograničenje za izvršavanje programa je 1 s.

Primer 1:

filmovi.in	filmovi.out
2 5 1 2 2 4 1	3

Objašnjenje:
lj Kako je hard disk na početku prazan, Draganče mora da presnimi film sa rednim brojem 1. Zatim, mora da presnimi film broj 2. Sledeći kupac naručuje film koji se već nalazi na disku, tako da ga Draganče odmah nareže. Naredni kupac traži film 4, tako da Draganče briše film broj 2 i presnimava preko film broj 4. Poslednji film koji se traži je broj 1, tako da Draganče ne mora da ga traži, jer se na hard disku nalaze filmovi 4 i 1.

Primer 2:

filmovi.in	filmovi.out
3 10 2 3 2 1 5 2 4 5 3 2	

fajl: filmovi.cpp

```
/*
  Niz a [] -  redni brojevi filmova
  Niz p [] -  niz indeksa, tako da je niz a [p [i]] sortiran
  Niz q [] -  indeksi filmova koji su trenutno na disku
  Niz next [] -  sledeci index sa jednakim rednim brojem, ili n ako takav ne postoji
  n -      broj filmova
  k -      kapacitet diska
*/
/*
#include <iostream>
#include <fstream>
#define MaxN 10001
#define MaxK 501

using namespace std;
int a [MaxN], p [MaxN], next [MaxN], q [MaxK];
int n, k, sol;

void input ()
{
  ifstream in ("filmovi.in");
  in >> n >> k;
  for (int i = 0; i < n; i++)
    in >> a [i];
  in.close ();
}

void output ()
{
  ofstream out ("filmovi.out");
  out << sol << endl;
  out.close();
```

```
}

void quicksort (int l, int r)
{
  int i, j, pivota, pivoti, tmp;

  i = l;
  j = r;
  tmp = (l + r) / 2;
  pivota = a [p [tmp]];
  pivoti = p [tmp];
  do
  {
    while ((i <= r) && ((a [p [i]] < pivota) || ((a [p [i]] == pivota) && (p [i] <
pivoti))))
      i++;
    while ((j >= l) && ((a [p [j]] > pivota) || ((a [p [j]] == pivota) && (p [j] >
pivoti))))
      j--;
    if (i <= j)
    {
      tmp = p [i];
      p [i] = p [j];
      p [j] = tmp;
      i++;
      j--;
    }
  }
  while (i < j);
  if (i < r)
    quicksort (i, r);
  if (l < j)
    quicksort (l, j);
}

void solve ()
{
  int i, m, j, ind, max, found;

  sol = 0;
  for (i = 0; i < n; i++)
    p [i] = i;
  quicksort (0, n - 1);
  for (i = 0; i < n; i++)
    if ((i + 1 < n) && (a [p [i]] == a [p [i + 1]]))
      next [p [i]] = p [i + 1];
    else
      next [p [i]] = n;
  m = 0;
  for (i = 0; i < n; i++)
  {
    found = 0;
    ind = 0;
    max = -1;
    for (j = 0; j < m; j++)
      if (a [q [j]] == a [i])
      {
        q [j] = i;
        found = 1;
        break;
      }
      else
      {
        if (max < next [q [j]])
        {
          max = next [q [j]];
          ind = j;
```

```
        }
      }
    if (found == 0)
    {
      if (m < k)
      {
        q [m] = i;
        m++;
      }
      else
        q [ind] = i;
      sol++;
    }
  }
}

int main ()
{
  input ();
  solve ();
  output ();
  return 0;
}
*/
```