Zadaci sa rešenjima Državno takmičenje 2010.

zadatak: Sumarum

Đurica je pronašao *n* karata poređanih u niz. Na kartama su zapisani celi brojevi. Datom nizu karata *A*, Đurica dodeluje vrednost *f(A)* koja je jednaka sumi razlika vrednosti na uzastopnim kartama. Formalno,
$f(A) = \Sigma(A_{k+1} - A_k),\ k = 1 .. n - 1$
gde A_i predstavlja vrednost na *i*-toj karti u nizu.
Đurica iz niza želi da izbaci najviše *K* karata. Izbacivanjem nekih *m* karata, $m \leq K$, dobija nov niz karata kojem ponovo računa vrednost na opisani način. Od svih mogućih odabira vrednosti *m* i svih mogućih odabira *m* karata koje će izbaciti, njega zanima onaj niz karata koji će imati najveću vrednost. Pomozite Đurici i recite mu koliko je ta najveća vrednost. Primetimo da Đurica ni u jednom momentu ne menja raspored karata datih na početku.

Ulaz.

(Ulazni podaci se učitavaju iz datoteke **sumarum.in**.) U prvom redu nalaze se celi brojevi *n* ($2 \leq n \leq 500.000$) i *K* ($0 \leq K \leq n - 2$). U narednom redu se učitavaju celi brojevi A_i ($-1.000.000 \leq A_i \leq 1.000.000$), u *i*-tom redu broj A_i.

Izlaz.

(Izlazne podatke upisati u datoteku **sumarum.out**) U prvom i jedinom redu ispisati jedan ceo broj koji predstavlja najveću vrednost niza koju Đurica može da dobije od početnog izbacivanjem najviše *K* karata.

Primer 1.

sumarum.in	sumarum.out
11 4 1 7 2 5 3 8 2 3 6 5 5	5

Objašnjenje.

Jedno optimalno rešenje je da se izbacimo 3 karte iz niza. Karte koje treba izbaciti imaju na sebi vrednost 5.

Primer 2.

sumarum.in	sumarum.out
3 1 10 9 8	-1

fajl: sumarum.cpp

```
/*
  Autor: Slobodan Mitrovic
*/
#include <iostream>
#include <cstdio>
#include <fstream>
#include <vector>
#include <cmath>
#include <time.h>
#define ffor(_a,_f,_t) for(int _a=(_f),__t=(_t);_a<__t;_a++)
#define all(_v) (_v).begin() , (_v).end()
#define sz size()
#define pb push_back
#define SET(__set, val) memset(__set, val, sizeof(__set))
#define FOR(__i, __n) ffor (__i, 0, __n)

using namespace std;

const int MAXN = 500000;

int mmax[MAXN], mmin[MAXN], a[MAXN];

int main(){
  FILE *fin;
  FILE *fout;
  fin = fopen("sumarum.in", "r");
  fout = fopen("sumarum.out", "w");

  int n, k;
  fscanf(fin, "%d %d", &n, &k);
```

```
  FOR (i, n)
    fscanf(fin, "%d", &a[i]);

  mmax[n - 1] = a[n - 1];
  for (int i = n - 2; i > -1; i--)
    mmax[i] = max(a[i], mmax[i + 1]);


  mmin[0] = a[0];
  ffor (i, 1, n)
    mmin[i] = min(a[i], mmin[i - 1]);

  int idxLeft = 0, idxRight = n - k - 1, ret = -(1 << 30);
  FOR (i, k + 1){
    ret = max(ret, mmax[idxRight] - mmin[idxLeft]);
    idxLeft++;
    idxRight++;
  }

  fprintf(fout, "%d\n", ret);
  fclose(fin);
  fclose(fout);
  return 0;
}
```

fajl: sumarum.pas

```
(* Autor: Slobodan Mitrovic *)
program sumarum;
var
  mmax, mmin, a : array [0 .. 499999] of longint;
  n, k, i, idxLeft, idxRight, ret : longint;
  f : text;

function max(x, y : longint) : longint;
begin
  if (x > y) then
    max := x
  else
    max := y;
end;

function min(x, y : longint) : longint;
begin
  if (x < y) then
    min := x
  else
    min := y;
end;

begin
  assign(f, 'sumarum.in');
  reset(f);
  readln(f, n, k);
  for i := 0 to n - 1 do
    read(f, a[i]);

  mmax[n - 1] := a[n - 1];
  for i := n - 2 downto 0 do
    mmax[i] := max(a[i], mmax[i + 1]);

  mmin[0] := a[0];
  for i := 1 to n - 1 do
    mmin[i] := min(a[i], mmin[i - 1]);
```

```
  idxLeft := 0;
  idxRight := n - k - 1;
  ret := -1000000000;
  for i := 0 to k do
    begin
      ret := max(ret, mmax[idxRight] - mmin[idxLeft]);
      inc(idxLeft);
      inc(idxRight);
    end;

  dec(idxRight);
  dec(idxLeft);
  assign(f, 'sumarum.sol');
  rewrite(f);
  writeln(f, ret);
  close(f);
end.
```

zadatak: Operacije

Duletu je bilo jako dosadno na času matematike (ne zato što ga gradivo ne interesuje već zato što je većinu prešao kada se pripremao za takmičenje) tako da je rešio da se malo poigra. Naime, rekao je svom drugu da napiše veliki prirodni broj *n* na papiru i tvrdio da može samo uz pomoć operacija: dodavanja broja 1 datom broju, oduzimanja broja 1 od datog broja i njegovim deljenjem sa 2 (ukoliko je paran) - dobiti broj 0. Kako njegovo rešenje mora stati na jednom papiru, Dule mora da nađe najmanji broj operacija koje dovode broj n na 0.
Kako je Dule uvideo da ovo nije toliko jednostavan problem, zamolio vas je da napišete program koji nalazi traženi minimalni broj operacija.

Ulaz.

(Ulazni podaci se učitavaju iz datoteke **mino.in**.) U prvom i jednom redu ulazne datoteke nalazi se prirodni broj *n*. Broj *n* neće imati više od 1000 cifara. Broj neće imati vodećih nula.

Izlaz.

(Izlazne podatke upisati u datoteku **mino.out**) U prvi i jedini red izlazne datoteke upisati minimalni broj operacija potrebnih da se prirodni broj *n* svede na nulu. Operacije su dodavanje ili oduzimanje jedinice i deljenje sa dva, ukoliko je broj paran.

Primer 1.

mino.in	mino.out
5	4

Objašnjenje.

Minimalni broj operacija potrebnih da se broj 5 dovede na nulu je 4. Jedini algoritam koji sadrži 4 operacija je:

$$5 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 0$$

Primer 2.

mino.in	mino.out
30	7

fajl: mino.cpp

```
/*
  Author: Andreja Ilic, PMF Nis
  e-mail: ilic_andrejko@yahoo.com
*/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define MAX_N 1005

// d - broj pamtimo u nizu d po ciframa u obrnutom redosledu
// n - broj trenutnih cifara u broj
int n, d[MAX_N], sol;

  // Unos podataka
```

```
void input()
{
  FILE *in = fopen ("mino.in", "r");
  char tmp [MAX_N];
  fscanf (in, "%s", &tmp);
  n = strlen(tmp);
  for (int i = 0; i < n; i++)
    d [n - i] = (int)(tmp [i] - '0');
  fclose(in);
}

void output()
{
  FILE *out = fopen ("mino.out", "w");
  fprintf (out, "%d\n", sol);
  printf ("%d\n", sol);
  fclose(out);
}

// Funckija koja deli trenurni broj sa 2
void divide(int c)
{
  int tmp = 0;
  for (int i = n; i >= 1; i--)
  {
    tmp = tmp * 10 + d [i];
    d [i] = tmp / c;
    tmp = tmp % c;
  }
  if (d [n] == 0)
    n--;
}

// Metoda koja trenutnom broju dodaje 1
void add()
{
  d [1] = d [1] + 1;
  int index = 1;
  while (d [index] == 10)
  {
    d [index] = 0;
    d [++index]++;
  }
}

// Metoda koja trenutnom broju oduzima 1
void sub()
{
  d [1] = d [1] - 1;
  int index = 1;
  while ((d [index] == -1) && (index < n))
  {
    d [index] == 9;
    d [++index]--;
  }
  if (d [n] == 0)
    n--;
}

void printN()
{
  printf ("%d: ", sol);
  for (int i = n; i >= 1; i--)
    printf("%d", d [i]);
  printf ("\n");
}
```

```
int solve()
{
  //printN();

  // [1] Ispitivanje kraja rekurzije
  if (n == 1)
  {
    switch (d [1])
    {
      case 0:
        return 0;
      case 1:
        return 1;
      case 3:
        return 3;
      default:
        break;
    }
  }

  int tmp = d [2] * 10 + d [1];

  // [2] Ukoliko je broj deljiv sa 2, delimo ga
  if (tmp % 4 == 2)
  {
    divide(2);
    return (1 + solve());
  }

  if (tmp % 4 == 0)
  {
    divide(4);
    return (2 + solve());
  }

  // [3] Ispitujemo da li treba dodati 1 ili oduzeti 1
  if ((tmp + 1) % 4 == 0)
    add();
  else
    sub();

  return (1 + solve());
}

int main()
{
  input();
  d [n + 1] = 0;
  sol = solve();
  output();

  return 0;
}
```

zadatak: Mrav Mika

Mrav Mika je vredan mrav i mnogo voli da kopa rupe i pravi tunele. Toliko voli to da radi, da je iskopao čak n ($5 \leq n \leq 1.000$) rupa, koje su međusobno povezane sa m ($1 \leq m \leq 10.000$) tunela. Međutim, mrav Mika nije znao zlatno pravilo kopača rupa i tunela. Ono nalaže da je mogućnost podele rupa u dve grupe, tako da između grupa postoji bar $\lfloor m / 2 \rfloor$ tunela, neophodan uslov za stabilnost podzemnog sistema a samim tim i dobijanje dozvole za korišćenje tunela. Inspekcija je došla kod Mike da proveri da li se pridržavao ovog pravila, a kako Mika ipak želi da svoje tunele pusti u rad, zanima ga da li je ono možda slučajno zadovo eno. Kako rupa i tunela ima previše, ovo je pretežak zadatak za Mikin mali mozak te vas je zamolio da mu pomognete. Dakle, potrebno je da nađete traženu podelu rupa u dve grupe (svaka rupa mora pripadati tačno jednoj grupi) ili da mu saopštite da neće moći da koristi tunele.

Ulaz.
(Ulazni podaci se učitavaju iz datoteke **mika.in**.) U prvom redu nalaze se brojevi *n* i *m*. Rupe su označene redom brojevima od 1 do *n*. U narednih m redova nalaze se po dva broja *a* i *b*, i oni označavauj da između rupa označenih brojevima *a* i *b* postoji direktan tunel. Između dve rupe može da postoji najviše jedan direktan tunel.
Izlaz.
(Izlazne podatke upisati u datoteku **mika.out**) Ukoliko nije moguće napraviti takvu podelu rupa, ispisati -1. U suprotnom, u prvom redu ispisati koliko ima rupa u prvoj i koliko u drugoj grupi, i potom u naredna dva reda redne brojeve rupa koje pripadaju prvoj, odnosno drugoj grupi.
Primer 1.

mika.in	mika.out
8 10 1 2 1 5 1 6 5 6 2 6 3 8 3 7 3 4 4 8 7 8	4 4 5 6 7 8 1 2 3 4

Primer 2.

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

fajl: mika.cpp

```
#include <iostream>
#include <cstdio>
using namespace std;

const long MaxN = 1001;

long n, m;

long susedCnt[MaxN];
long sused[MaxN][MaxN]; // broj cvorova je mali pa mozemo i ovako da pamtimo listu suseda

bool grupa[MaxN];   // da li pripada prvoj grupi?
long grupaCnt[MaxN]; // koliko cvor gadja unutar svoje grupe
bool dobar[MaxN]; // da li je cvor u listi dobrih ili losih cvorova

// lista suseda zadovoljenih i nezadovoljenih cvorova
long next[MaxN];
long prev[MaxN];

// pocetak liste
int goodStart, badStart;


char fileIn[] = "mika.in";
char fileOut[] = "mika.out";


void ubaci(int &list, int k)
```

```
{
  if (list == -1)
  {
    // prazna lista
    list = k;
    prev[k] = -1;
    next[k] = -1;
  }
  else
  {
    // na pocetak
    prev[list] = k;
    next[k] = list;
    prev[k] = -1;
    list = k;
  }
}

void izbaci(int &list, int k)
{
  if (list == k)
  {
    list = next[list];
    if (list != -1)
      prev[list] = -1;
  }
  else
  {
    next[prev[k]] = next[k];
    if (next[k] != -1)
      prev[next[k]] = prev[k];
  }
}


// dodaje nov cvor u odgovarajucu listu
void novCvor(int k)
{
  if (grupaCnt[k] > susedCnt[k] / 2)
  {
    // los cvor
    dobar[k] = false;
    ubaci(badStart, k);
  }
  else
  {
    // dobar cvor
    dobar[k] = true;
    ubaci(goodStart, k);
  }
}


void transfer()
{
  int v = badStart;

  // posle prebacivanja, cvor postaje dobar
  dobar[v] = true;
  izbaci(badStart, v);
  ubaci(goodStart, v);


  for (int i = 0; i < susedCnt[v]; i++)
  {
    int w = sused[v][i];
    if (grupa[w] == grupa[v])
```

```
    {
      // bili ista grupa, vise nisu
      grupaCnt[w]--;
      grupaCnt[v]--;

      // mozda je postao dobar
      if (!dobar[w] && (grupaCnt[w] <= susedCnt[w] / 2))
      {
        // izbacujemo ga iz liste losih, i guramo u listu dobrih
        dobar[w] = true;
        izbaci(badStart, w);
        ubaci(goodStart, w);
      }
    }
    else
    {
      // postali su ista grupa
      grupaCnt[w]++;
      grupaCnt[v]++;

      // mozda je cvor w postao los
      if (dobar[w] && (grupaCnt[w] > susedCnt[w] / 2))
      {
        // izbacujemo ga iz liste dobrih, i guramo u listu losih
        dobar[w] = false;
        izbaci(goodStart, w);
        ubaci(badStart, w);
      }
    }
  }

  // promeni grupu
  grupa[v] = !grupa[v];
}


int main()
{
  FILE *f = fopen(fileIn, "r");
  fscanf(f, "%ld %ld", &n ,&m);

  for (int i = 0; i < n; i++)
  {
    grupaCnt[i] = 0;
    susedCnt[i] = 0;

    // inicijalna grupa
    if (i < n/2) grupa[i] = true;
    else grupa[i] = false;
  }

  for (long i = 0; i < m; i++)
  {
    long a, b;
    fscanf(f, "%ld %ld", &a, &b);
    a--; b--;

    sused[a][susedCnt[a]++] = b;
    sused[b][susedCnt[b]++] = a;

    if (grupa[a] == grupa[b])
    {
      grupaCnt[a]++;
      grupaCnt[b]++;
    }
  }
  fclose(f);
```

```
  goodStart = -1;
  badStart = -1;
  for (int i = 0; i < n; i++)
    novCvor(i);

  // prebacujemo iz jedne grupe u drugu, dokle god mozemo
  while (badStart != -1)
  {
    transfer();
  }

  int grupa1 = 0, grupa2 = 0;
  for (int i = 0; i < n; i++)
  {
    if (grupa[i])
      grupa1++;
    else
      grupa2++;
  }

  f = fopen(fileOut, "w");
  fprintf(f, "%d %d\n", grupa1, grupa2);

  for (int i = 0; i < n; i++)
    if (grupa[i]) fprintf(f, "%d ", (i+1));
  fprintf(f, "\n");
  for (int i = 0; i < n; i++)
    if (!grupa[i]) fprintf(f, "%d ", (i+1));
  fprintf(f, "\n");
  fclose(f);

  return 0;
}
```

zadatak: Robotići

Robotić WALL-E se još od Okružnog takmičenja usamljen igra na deponiji smeća i to mu već pomalo postaje dosadno. Kako bi stao na put dokolici, priredio je veliku žurku na deponiji na koju je pozvao sve svoje metalne drugare sa fejsbuka. Robotska žurka se odvija standardno: Svaki robotić zauzme jedno polje matrice koja predstavlja deponiju i zamisli jedan od četiri moguća pravca u kom želi da načini korak. Potom se robotići istovremeno pomere, svaki za tačno jedan korak u svom željenom pravcu. Tako se nađu na novim pozicijama i žurka se završava (sve što je dobro traje kratko).
Međutim, odaziv je bio veoma velik i na žurci se napravila prilična gužva. Zbog toga nisu svi robotići u stanju da načine željeni korak. Da bi robotić uspeo da ode na polje na koje je zamislio, ni jedan drugi robotić se ne sme naći na tom polju nakon odigranog koraka. To znači da od svih robotića koji žele da dođu na neko polje samo jedan može da uspe u tome, i to samo ako robotić koji je prethodno bio na tom polju uspe da ga napusti. U toku koraka robotići se mogu mimoilaziti bez problema, ali po izvršenju koraka na svakom polju se sme nalaziti najviše jedan robotić.
WALL-E želi da mu žurka koliko-toliko uspe i da usreći što više svojih drugara. On treba da odabere najveći mogući broj robotića koji će moći da naprave korak, dok će svi ostali morati da ostanu na svojim pozicijama. Koliko najviše robotića može napraviti korak?
Ulaz.
(Ulazni podaci se učitavaju iz datoteke **robotici.in**.) U prvom redu nalaze se dva broja m i n ($1 \leq m, n \leq 200$), dimenzije matrice koja predstavlja deponiju. U svakom od sledećih m redova nalazi se n razmakom razdvojenih brojeva koji predstavljaju polja. Brojevi označavaju šta se nalazi na odgovarajućem polju i imaju sledeća značenja:

1. 0 - prazno polje
2. 1 - robotić koji želi da napravi korak na desno
3. 2 - robotić koji želi da napravi korak na gore
4. 3 - robotić koji želi da napravi korak na levo
5. 4 - robotić koji želi da napravi korak na dole

Nijedan robotić neće poželeti da napusti deponiju za vreme žurke (tj. napravi korak van matrice).
Izlaz.
(Izlazne podatke upisati u datoteku **robotici.out**) Na izlaz ispisati samo jedan broj - koliko najviše robotića moće da načini korak.
Primer 1.

robotici.in	**robotici.out**
4 5 0 0 0 0 0 0 1 1 4 0 0 2 0 1 0 0 0 0 0 0	5

Primer 2.

robotici.in	**robotici.out**
4 3 0 0 0 0 4 0 0 2 0 0 0 0	2

fajl: robotici.cpp

```
#include <iostream>
#include <cstdio>
using namespace std;

const int MaxN = 201;

int move[][2] = { {0, 1}, {-1, 0}, {0, -1}, {1, 0} };

int n, m;
int smer[MaxN][MaxN];
int inv[MaxN][MaxN];

long lvl[MaxN][MaxN];

long marker[MaxN][MaxN];
long markID;


long najdublje(int i, int j)
{
  // markiraj
  lvl[i][j] = -1;

  long najduboko = 0;
  for (int k = 0; k < 4; k++)
  {
    if (inv[i][j] & (1<<k))
      najduboko = max(najduboko, 1 + najdublje(i + move[k][0], j + move[k][1]));
  }
  return najduboko;
}

long ciklus(int i, int j, int dubina)
{
  // markiraj
  lvl[i][j] = dubina;
  marker[i][j] = markID;

  int in = i + move[smer[i][j] - 1][0], jn = j + move[smer[i][j] - 1][1];
  if (lvl[in][jn] == 0)
    return ciklus(in, jn, dubina + 1);
  else if (marker[in][jn] != markID)
```

```
    return 0;
  else
    return dubina - lvl[in][jn] + 1;
}

int main()
{
  scanf("%d %d", &n, &m);
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      scanf("%d", &smer[i][j]);

  // obrnuto
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
    {
      inv[i][j] = 0;
      if (j < m - 1 && smer[i][j+1] == 3)
        inv[i][j] |= 1;

      if (i > 0 && smer[i-1][j] == 4)
        inv[i][j] |= 2;

      if (j > 0 && smer[i][j-1] == 1)
        inv[i][j] |= 4;

      if (i < n - 1 && smer[i+1][j] == 2)
        inv[i][j] |= 8;
    }


  memset(lvl, 0, sizeof(lvl));
  long longestPath = 0;

  // drvece
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
    {
      // koren drveta
      if (smer[i][j] == 0)
      {
        long deep = najdublje(i, j);
        longestPath += deep;
      }
    }

  // ciklusi
  memset(marker, 0, sizeof(marker));
  markID = 1;
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
    {
      // nije markiran, trazimo ciklus
      if (lvl[i][j] == 0 && smer[i][j] != 0)
      {
        long cycle = ciklus(i, j, 0);
        longestPath += cycle;
        markID++;
      }
    }

  printf("%ld\n", longestPath);
  return 0;
}
```

fajl: robotici.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;


class Cell {
  int y, x;

  public Cell(int y, int x) {
    this.y = y;
    this.x = x;
  }
}


public class Robotici {
  static final int[] dy = new int[] {0, 0, -1, 0, 1};
  static final int[] dx = new int[] {0, 1, 0, -1, 0};
  String fileIn, fileOut;

  int ny, nx;
  int[][] dir;
  int[][] color;
  int[][] rank;
  int maxDancers;


  Cell paint(int startY, int startX, int colorOld, int colorNew) {
    int l = 0;
    int y = startY;
    int x = startX;

    while (dir[y][x] != 0 && color[y][x] == colorOld) {
      color[y][x] = colorNew;
      l++;
      int d = dir[y][x];
      y += dy[d];
      x += dx[d];
    }

    int r = l + rank[y][x];
    y = startY;
    x = startX;

    for (int i = 0; i < l; i++) {
      rank[y][x] = r--;
      int d = dir[y][x];
      y += dy[d];
      x += dx[d];
    }

    return new Cell(y, x);
  }


  void solve() {
    color = new int[ny][nx];
    rank = new int[ny][nx];

    int currentColor = 1;
    for (int y = 0; y < ny; y++)
      for (int x = 0; x < nx; x++)
        if (dir[y][x] != 0 && color[y][x] == 0) {
```

```
          Cell front = paint(y, x, 0, currentColor);
          int c = color[front.y][front.x];

          if (c == 0)                                          // 1) naisli smo na slobodno
polje
            color[front.y][front.x] = currentColor;
          else
            if (c == currentColor)  {                       // 2) napravili smo petlju, rep
blokira
              paint(front.y, front.x, currentColor, -1);
              paint(y, x, currentColor, -2);
            } else {
              if (c < 0)                                    // 3) udarili smo u petlju ili
blokirani lanac, blokiramo
                paint(y, x, currentColor, -2);
              else                                          // 4) pridruzujemo se postojecem
lancu koji ne blokira
                paint(y, x, currentColor, c);
            }

          currentColor++;
        }

    maxDancers = 0;

    int[] maxColorRank = new int[currentColor];
    for (int y = 0; y < ny; y++)
      for (int x = 0; x < nx; x++) {
        int c = color[y][x];
        if (c == -1)
          maxDancers++;                                        // u petlji se svi krecu
        if (c > 0)
          if (rank[y][x] > maxColorRank[c])                    // u ostalim grupama samo
najduzi lanac
            maxColorRank[c] = rank[y][x];
      }

    for (int maxRank : maxColorRank)
      maxDancers += maxRank;
  }


  void readInput() throws FileNotFoundException {
    Scanner in = new Scanner(new File(fileIn));

    ny = in.nextInt();
    nx = in.nextInt();
    dir = new int[ny][nx];
    for (int y = 0; y < ny; y++)
      for (int x = 0; x < nx; x++)
        dir[y][x] = in.nextInt();

    in.close();
  }


  void writeOutput() throws IOException {
    PrintWriter out = new PrintWriter(fileOut);
    out.print(maxDancers);
    out.close();
  }


  public Robotici(String fileIn, String fileOut) {
    this.fileIn = fileIn;
    this.fileOut = fileOut;
```

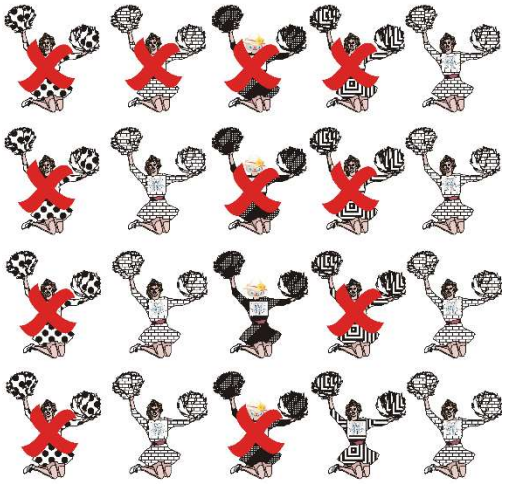
```
    try {
      readInput();
      solve();
      writeOutput();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }


  public static void main(String[] args) {
    new Robotici("robotici.in", "robotici.out");
  }
}
```

zadatak: Progsice

Cica je vođa grupe progsica - čirlidersica koje su se specijalizovale za zabavljanje publike na Državnom takmičenju iz programiranja. Tokom poluvremena ove prestižne manifestacije progsice izvode svoj performans.
Svaka progsica obučena je u kostim određene boje. Cica i ekipa zamislile su da se, kao vrhunac nastupa, poređaju u liniju duž pravca sever-jug, a da potom neke od njih napuste arenu (moguće je i da nijedna ne ode) i to tako da publika sa istočne tribine, gledajući preostale progsice, vidi isti redosled boja kakav vidi i publika sa zapadne tribine. Tim povodom vas mole za pomoć: pitaju na koliko načina mogu ostvariti svoju zamisao.
Ulaz.
(Ulazni podaci se učitavaju iz datoteke **progsice.in**.) U prvom redu ulazne datoteke nalazi se broj progsica, n ($1 \leq n \leq 5.000$). U drugom redu dat je poredak progsica, onako kako ih vidi publika s istočne tribine, u momentu kada neke od njih treba da napuste arenu. Poredak je dat u formi stringa dužine n, pri čemu i-ti karakter u stringu označava boju kostima i-te progsice (naravno, isti karakteri uvek označavaju istu boju, a različiti karakteri različitu).
Izlaz.
(Izlazne podatke upisati u datoteku **progsice.out**) U prvi i jedini red izlazne datoteke upisati ostatak pri deljenju broja načina da progsice ostvare svoju zamisao brojem 1.000.000.007.
Primer 1.

progsice.in	**progsice.out**
5 patka	8

Objašnjenje.
Na slici su prikazane sve mogućnosti (prekrižene su one progsice koje treba da napuste arenu). Slika prikazuje pogled s istočne tribine, a uočava se da i publika sa zapadne vidi isti redosled boja. Primetiti da se ne ubraja mogućnost kada sve progsice napuste teren (tj., mora ostati bar jedna da nastavi zabavljanje publike).



fajl: progsice.cpp

```
/*
  Autor: Slobodan Mitrovic
*/
#include <iostream>
#include <cstdio>
#include <fstream>
#include <vector>
#include <cmath>
#include <time.h>
#define ffor(_a,_f,_t) for(int _a=(_f),__t=(_t);_a<__t;_a++)
#define all(_v) (_v).begin() , (_v).end()
#define sz size()
#define pb push_back
#define SET(__set, val) memset(__set, val, sizeof(__set))
#define FOR(__i, __n) ffor (__i, 0, __n)

using namespace std;

const int MAXN = 5000, MOD = 1000000007;

char str[MAXN + 10];

int fastMod(int val){
//  return val % MOD;
  if (val > MOD)
    return val - MOD;
  return val;
}


int fastestDP[MAXN + 1][MAXN + 1];

int fastestSolution(int n){
  SET(fastestDP, 0);
  int *cur;
  for (int i = n - 1; i > -1; i--){
    fastestDP[i][i] = 1;
    for (int j = i + 1; j < n; j++){
      cur = &fastestDP[i][j];
      *cur = fastMod(fastestDP[i][j - 1] + fastestDP[i + 1][j]);
      *cur = fastMod((*cur) - fastestDP[i + 1][j - 1] + MOD);
      if (str[i] == str[j])
        *cur = fastMod((*cur) + 1 + fastestDP[i + 1][j - 1]);
```

```
    }
  }
  return fastMod(fastestDP[0][n - 1]);
}

int main(){
  FILE *fin;
  FILE *fout;


  fin = fopen("progsice.in", "r");
  fout = fopen("progsice.sol", "w");

  int n;
  fscanf(fin, "%d\n", &n);
  FOR (i, n)
    fscanf(fin, "%c", &str[i]);

  fprintf(fout, "%d\n", fastestSolution(n));
  fclose(fin);
  fclose(fout);
  return 0;
}
```

fajl: progsice.pas

```
var i,n,k:integer;
    a:ansistring;
    palin:array[1..5000,1..5000] of longint;
    f:text;
begin
 assign(f,'progsice.in');
 reset(f);
 readln(f,n);
 readln(f,a);
 close(f);
 for i:=1 to n-1 do
   begin
    palin[i,i]:=1;
    if a[i]=a[i+1] then palin[i,i+1]:=3
                   else palin[i,i+1]:=2;
   end;
 palin[n,n]:=1;
 for k:=1 to n-1 do
   for i:=1 to n-k do
     begin
      if a[i]=a[i+k] then palin[i,i+k]:=palin[i,i+k-1]+palin[i+1,i+k]+1
                     else palin[i,i+k]:=palin[i,i+k-1]+palin[i+1,i+k]-palin[i+1,i+k-1];
      if palin[i,i+k]>1000000007 then palin[i,i+k]:=palin[i,i+k]-1000000007
                                 else if palin[i,i+k]<0 then
palin[i,i+k]:=palin[i,i+k]+1000000007;
     end;
 assign(f,'progsice.out');
 rewrite(f);
 writeln(f,palin[1,n]);
 close(f);
end.
```