ispitni centar
PRAVA
MJERA
ZNANJA

DRŽAVNO TAKMIČENJE 2017.

ŠIFRA UČENIKA

SREDNJA ŠKOLA

PROGRAMIRANJE

UKUPAN BROJ OSVOJENIH BODOVA

Test pregledala/pregledao

.....

.....

Podgorica, ... 20..... godine

Uputstva takmičarima

Ovo takmičenje sastoji se od rješavanja **3 problemska zadatka** u vremenu od **4 sata** (240 minuta). Zadatke je potrebno rješavati u jednom od sljedećih programskih jezika: Pascal, C, C++ ili Java. Takmičari koji koriste Pascal moraju programirati u programskom alatu **FreePascal ili TurboPascal.** Takmičari u C-u i C++-u moraju koristiti programske alate **CodeBlocks**, **DJGPP, DevCpp ili GCC**. Za programski jezik Java predviđena je upotreba platforme **Eclipse**. Dozvoljeno je koristiti editor po izboru i pomoću navedenih alata prevoditi izvorni kod u izvršnu datoteku.

Tokom takmičenja **ne smijete komunicirati** ni sa jednom osobom, osim dežurne osobe takmičenja. To znači da morate **raditi samostalno** i **ne smijete koristiti Internet**. Takođe, zabranjena je upotreba bilo kakvih ranije napisanih programa ili dijelova programa.

Po isteku vremena predviđenog za takmičenje, na desktopu u folderu sa imenom ***Takmicenje2017*** moraju se nalaziti datoteke sa snimljenim izvornim kôdovima rješenja. Nakon takmičenja, komisija će testirati vaša rješenja na ranije izabranim test podacima i dodijeliti vam određeni broj bodova. Na kraju svakog zadatka dati su primjeri test podataka. Ti primjeri služe da bi vam tekst zadataka bio što je moguće jasniji te za provjeru formata ulaza i izlaza, a ne služe za provjeru ispravnosti vašeg programa. Ako vaš program radi na tim primjerima, to **nije garancija** da će raditi na službenim podacima za testiranje.

Zadaci ne nose jednak broj bodova. Lakše i brže rješivi zadaci nose manje bodova, dok teži nose više bodova. Svaki test podatak u nekom zadatku nosi jednak broj bodova. Ukupan broj bodova na nekom zadatku jednak je zbiru bodova test podataka koji se poklapaju sa službenim rješenjem. Ukupan broj bodova jednak je zbiru bodova na svim zadacima.

Sve informacije o zadacima (ime zadatka, vremensko i memorijsko ograničenje, način bodovanja) možete naći na uvodnoj stranici s naslovom *Zadaci*. Ako vam nije jasno nešto u vezi načina organizacije ovog takmičenja, odmah postavite pitanje dežurnom da vam to razjasni.

Tokom cijelog takmičenja možete postavljati pitanja dežurnom u vezi zadataka. Dozvoljena su pitanja **koja razjašnjavaju nejasnoće u tekstu zadatka**. Ne smijete postavljati pitanja u vezi rješavanja zadataka. Prije nego postavite pitanje, pročitajte još jednom zadatak, jer je moguće da ste u prethodnom čitanju preskočili dio teksta zadatka.

VAŽNO za C/C++!

Glavni program (glavna funkcija) **mora** biti deklarisan kao: `int main(void) { ... }`.

Program mora završiti svoje izvođenje naredbom `return 0;` unutar funkcije `main` ili naredbom `exit(0);`.

Zabranjeno je koristiti biblioteke <conio.h> i <cconio>, kao i sve funkcije deklarisane u ovim bibliotekama (npr. clrscr(); getch(); getche(); i sl.). Zabranjeno je koristiti i **sve** sistemske (nestandardne) biblioteke.

Zabranjeno je koristiti funkcije itoa() i ltoa() jer one ne postoje u standardu jezika C/C++. Umjesto tih funkcija možete koristiti funkciju sprintf() deklarisanu u <stdio.h> i <cstdio>, koja ima i veće mogućnosti primjene,

Dozvoljeno je koristiti sve ostale standardne biblioteke (koje su dio jezika), uključujući i STL (Standard Template Library) u jeziku C++.

VAŽNO za Pascal!

Program **mora** regularno završiti svoje izvođenje naredbom end. unutar glavnog programa ili naredbom halt;.

Zabranjeno je koristiti bilo kakve biblioteke, a posebno biblioteku crt, tj. zabranjeno je u programu imati direktivu uses. To znači da u programu ne smije biti naredbi clrscr() i readkey().

Nepoštovanje ovih pravila ili nepridržavanje formata izlaznih podataka rezultiraće nepovratnim gubitkom bodova. Nemojte štampati ništa što se u zadatku ne traži, kao npr. poruke tipa 'Rjesenje je:' ili 'Unesite brojeve' i slično!

Srećno i uspješno takmičenje!

Zadaci

Zadatak	Zadatak1	Zadatak2	Zadatak3
Izvorni kôd	zadatak1.java zadatak1.pas zadatak1.c zadatak1.cpp	zadatak2.java zadatak2.pas zadatak2.c zadatak2.cpp	zadatak3.java zadatak3.pas zadatak3.c zadatak3.cpp
Memorijsko ograničenje	64 MB	256 MB	64 MB
Vremensko ograničenje (po test podatku)	1 sekunda	2 sekunde	2 sekunde
Broj test podataka	10	10	10
Broj bodova (po test podatku)	3	3.5	3.5
Ukupno bodova	**30**	**35**	**35**

Napomena: Program u C-u i C++-u treba kompajlirati sa sljedećim opcijama: *–O2 –lm –static*, a program u Pascalu sa *–O1 –XS*.

Zadatak 1 – Kodiranje

Dat je niz sastavljen od cifara od '0' do '9'. Kodiramo ga tako da uzastopne cifre zamijenimo malim slovima engleske abecede na sljedeći način: 10 → 'a', 11 → 'b', 12 → 'c', … , 34 → 'y', 35 → 'z'. Na primjer, u nizu "1234" možemo zamijeniti prve dvije cifre i dobiti "c34". Moguće je i "23" zamijeniti sa "n" pa se dobija "1n4". Ako zamijenimo prve dvije i posljednje dvije cifre dobijamo "cy".
Napisati program koji izračunava koliko različitih nizova možemo dobiti opisanim kodiranjem, ako se za kodiranje koristi prvih n malih slova engleske abecede. Na primjer, za n = 5, koristimo slova 'a', 'b', 'c', 'd' i 'e'. Za n = 0, ne koristimo slova pri kodiranju.

Ulazni podaci

Prvi red ulaza sadrži cio broj n ($0 \le n \le 26$). Drugi red sadrži dati niz (najviše 50 cifara).

Izlazni podaci

U jedini red izlaza štampati jedan cio broj – traženi broj nizova.

Test primjeri

Ulaz	Izlaz
0 10237	1
1 10237	1
26 10237	4

Rješenje:

Rješenje za oko 50% bodova: Rekurzivno rješenje, po sljedećem obrascu:

```
int f(string s)
{
      c1 = f(s bez prvog simbola)
      Ako prva dva simbola iz s mogu biti zamjenjena slovom,
            računamo c2  =  f(s bez prve dvije cifre)
      vratiti c1+c2
}
// rekurzivno rjesenje
#include<iostream>
#include<cstdlib>
using namespace std;

string s;
int m;
```

```
int f(string s)
{
  if(s=="") return 1;
  int c=f(s.substr(1));
  if(s.size()>1)
  {
   string s2=s.substr(0,2);
   if(s2.size()==2)
   {
    int v=atoi(s2.c_str());
    if((10<=v)&&(v<=m))
     c=c+f(s.substr(2));
   }
  }
  return c;
}

int main()
{
  cin >> m;
  m=9+m;
  cin >> s;
  int r=f(s);
  cout << r << endl;
}
```

Rješenje za 100%: Memoizacija za vrijednosti f(s). Moguće je npr, koristiti kontejner map iz STL.

```
#include<iostream>
#include<cstdlib>
#include<map>
using namespace std;

typedef long long int INT;

string s;
int m;

map<string,INT> d;

INT f(string s)
{
  if(s=="") return 1;
  if(d.count(s)>0) return d[s];
  INT c=f(s.substr(1));
  if(s.size()>1)
  {
   string s2=s.substr(0,2);
   if(s2.size()==2)
   {
    int v=atoi(s2.c_str());
    if((10<=v)&&(v<=m))
```

```
      c=c+f(s.substr(2));
    }
   }
   d[s]=c;
   return c;
}

int main()
{
   cin >> m;
   m=9+m;
   cin >> s;
   INT r=f(s);
   cout << r << endl;
}
```

Zadatak 2 – Svemirski brojevi

Vedran je osmislio novu igru. Na pravouganoj tabli sa N redova i M kolona raspoređeno je nekoliko svemirskih brodova, pri čemu brod zauzima tačno jedno polje. Svaki od brodova može da se kreće u jednom, ranije definisanom smjeru, brzinom jedno polje u sekundi. Brod se kreće ili dok ne izađe sa table (kada se igra završava) ili dok se ne sudari sa drugim brodom na tabli. Pri sudaru, brod nestaje a kretanje nastavlja brod koji je udaren. Na početku, nijedan od brodova se ne kreće. Možete izabrati jedan od brodova sa table i pokrenuti ga, pa se zatim igra odvija na gore opisani način. U zavisnosti od izabranog broda, igra može trajati različit broj sekundi. Odredite koliko najduže može trajati igra.

Ulazni podaci

Prvi red ulaza sadrži dva cijela broja **N** i **M** – broj redova i broj kolona table ($1 \leq$ **N**, **M** ≤ 100). Sljedećih **N** redova sadrže po **M** simbola iz skupa {'<', '^', '>', 'v', '.'}. Strelice označavaju brod i njegov smjer kretanja (redom: lijevo, gore, desno i dolje), a tačka označava prazno polje kroz koje brod samo prolazi. Garantuje se da će na tabli biti bar jedan brod.

Izlazni podaci

Jedini red izlaza sadrži jedan cio broj – koliko sekundi traje najduža igra za dati raspored na tabli.

Test primjeri

Ulaz	Izlaz
5 6 vv.^>> .^.<>. >>.^>v .^v>.. ^^...<	21

Pojašnjenje test primjera

Ako izaberemo brod iz gornjeg lijevog ugla, tada će igra trajati 16 sekundi:

1. Prvi brod ide dolje 2 sekunde , do sudara sa (2, 0);
2. Drugi brod (iz polja (2,0)) ide desno 1 sekundu, do sudara sa (2, 1);
3. Treći brod ide desno 2 sekunde, do sudara sa (2, 3);
4. Četvrti brod ide gore 1 sekundu, do sudara sa (1, 3);
5. Peti brod ide lijevo 2 sekunde, do sudara sa (1, 1);
6. Šesti brod ide gore 1 sekundu, do sudara sa (0, 1);

7. Sedmi brod ide dolje 3 sekunde, do sudara sa (3, 1). Primjetite da su brodovi (1, 1) i (2, 1) nestali u sudarima pa su ta polja prazna;
8. Osmi brod ide gore 4 sekunde, dok ne izađe sa table, pa se igra završava.

Optimalno bi bilo da izaberete polje (2, 3), kada bi igra trajala 21 sekundu.

Rješenje:
Za 80% bodova dovoljno je pokrenuti simulaciju kretanja brodova iz svakog polja tabele. Složenost je $O(N^2 * M^2 * \max(N, M))$.

Za 100% bodova potebno je smanjiti složenost na $O(N^2 * M^2)$. Za svako polje tabele posmatrajmo njena 4 susjeda (gore, desno, dolje, lijevo). Kada brod udari u polje, mi znamo koji su mu susjedni brodovi (ignorišemo prazna polja). Sada za dati brod možemo za O(1) da stignemo sljedećeg broda (ili da ustanovimo da izlazimo sa table), pa je složenost jedne simulacije O(N * M), a ukupna složenost je $O(N^2 * M^2)$.

Za 80% bodova:

```
#include <cstdio>

const int MAX = 104;
FILE *in = stdin, *out = stdout;

int numRows, numCols;
char board[MAX][MAX], a[MAX][MAX];
int dir[4][2] = { {0, -1}, {-1, 0}, {0, +1}, {+1, 0} };

int getType(char ship) {
    if (ship == '<') return 0;
    if (ship == '^') return 1;
    if (ship == '>') return 2;
    if (ship == 'v') return 3;
    return -1;
}

int simulate(int row, int col) {
    for (int i = 0; i < numRows; i++)
        for (int c = 0; c < numCols; c++)
            a[i][c] = board[i][c];

    int moves = 0, type = -1;
    while (row >= 0 && row < numRows && col >= 0 && col <
numCols) {
        moves++;
        if (a[row][col] != '.') {
            type = getType(a[row][col]);
            a[row][col] = '.';
        }
```

```
        row += dir[type][0], col += dir[type][1];
    }
    return moves;
}

int eval() {
    int ans = 0;
    for (int srow = 0; srow < numRows; srow++) {
        for (int scol = 0; scol < numCols; scol++) {
            if (board[srow][scol] != '.') {
                int cur = simulate(srow, scol);
                ans = ans < cur ? cur : ans;
            }
        }
    }
    return ans;
}

int main(void) {
    fscanf(in, "%d %d", &numRows, &numCols);
    for (int row = 0; row < numRows; row++)
        fscanf(in, "%s", board[row]);
    fprintf(out, "%d\n", eval());
    return 0;
}
```

Za 100% bodova:

```
#include <cstdio>
#include <cmath>
#include <stdlib.h>

const int MAX = 102;
FILE *in = stdin, *out = stdout;

int numRows, numCols;
char board[MAX][MAX];
int links[MAX][MAX][4], cur[MAX][MAX][4];
int dir[4][2] = { {0, -1}, {0, +1}, {-1, 0}, {+1, 0} };

int getType(char ship) {
    if (ship == '<') return 0;
    if (ship == '>') return 1;
    if (ship == '^') return 2;
    if (ship == 'v') return 3;
    return -1;
}

int simulate(int row, int col) {
    int moves = 0;
    for (int i = 0; i < numRows; i++)
        for (int c = 0; c < numCols; c++)
            for (int d = 0; d < 4; d++)
                cur[i][c][d] = links[i][c][d];
```

```
    while (true) {
        int type = getType(board[row][col]);

        if (cur[row][col][type] == -1) {
            if (type == 0) moves += col + 1;
            if (type == 1) moves += numCols - col;
            if (type == 2) moves += row + 1;
            if (type == 3) moves += numRows - row;
            break;
        }

        // Brisemo tekuci brod i update veza
        for (int d = 0; d < 4; d++) {
            if (cur[row][col][d] != -1) {
                int nrow = cur[row][col][d] >> 10;
                int ncol = cur[row][col][d] & ((1 << 10) - 1);
                cur[nrow][ncol][d ^ 1] = cur[row][col][d ^ 1];
            }
        }

        // Sljedeca pozicija
        int nrow = cur[row][col][type] >> 10;
        int ncol = cur[row][col][type] & ((1 << 10) - 1);
        moves += abs(row - nrow) + abs(col - ncol);
        row = nrow, col = ncol;
    }
    return moves;
}

int eval() {
    // Lista susjeda
    for (int row = 0; row < numRows; row++) {
        for (int col = 0; col < numCols; col++) {
            for (int d = 0; d < 4; d++) {
                links[row][col][d] = -1;
                int crow = row + dir[d][0], ccol = col +
dir[d][1];
                while (crow >= 0 && crow < numRows && ccol >= 0
&& ccol < numCols) {
                    if (board[crow][ccol] != '.') {
                        links[row][col][d] = (crow << 10) |
ccol;
                        break;
                    }
                    crow += dir[d][0], ccol += dir[d][1];
                }
            }
        }
    }

    int ans = 0;
    for (int row = 0; row < numRows; row++) {
        for (int col = 0; col < numCols; col++) {
            if (board[row][col] != '.') {
                int cur = simulate(row, col);
```

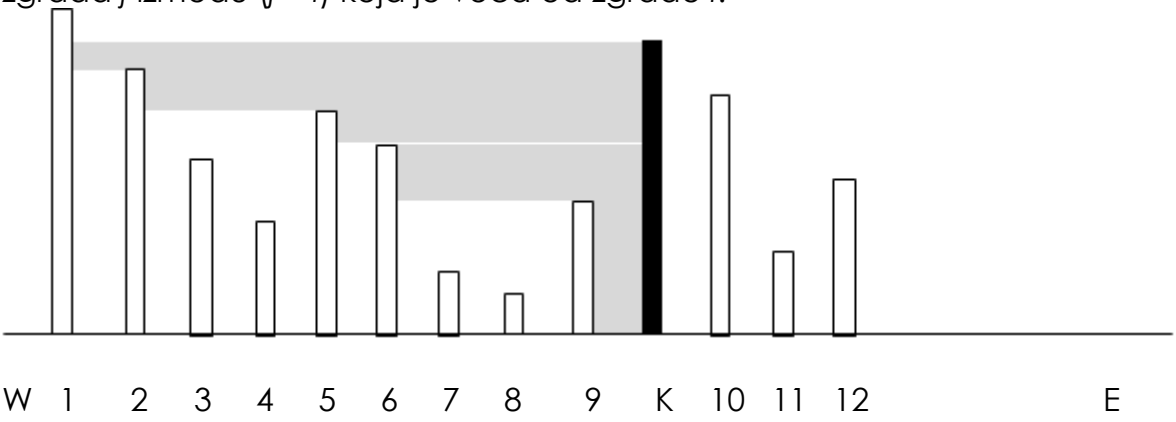
```
                ans = ans < cur ? cur : ans;
            }
        }
    }
    return ans;
}

int main(void) {
    fscanf(in, "%d %d", &numRows, &numCols);
    for (int row = 0; row < numRows; row++)
        fscanf(in, "%s", board[row]);

    fprintf(out, "%d\n", eval());
    return 0;
}
```

Zadatak 3 – Zgrade

Grad X ima N zgrada, poređanih u red od zapada ka istoku I numerisanih redom brojevima od 1 do do N. Svaka zgrada ima različitu visinu od ostalih - cio broj h_1, h_2, …, h_N. Gradska uprava planira da izgradi kulu, u istom redu sa ostalim zgradama. Ona može biti prije prve zgrade, između dvije postojeće zgrade ili iza posljednje zgrade u redu. Sa kule biće emitovane poruke građanima. Kula mora biti visine H, i ta visina je različita od visina svih ostalih zgrada. Kula može emitovati signal samo ka zapadu. Signali su pravolinijski i paralelni sa podlogom i prostiru se cijelom visinom kule. Dakle, možemo zamisliti da kula emituje neprekidni opseg signala čija je širina jednak visini kule. Svaka zgrada prima signale preko prijemnika koji je na vrhu zgrade. Zgrada prima signal ako bar jedan zrak dopire do prijemnika. Drugim riječima, zgrada *i* će primiti signal ako: zgrada *i* je zapadno od kule; *i* nije veća od kule i ne postoji zgrada *j* između (*j* > *i*) koja je veća od zgrade *i*.



Na slici iznad, zgrade koje će primiti signal su 2, 5, 6 i 9.

Napišite program koji će, na osnovu visina zgrada i visine kule, odrediti maksimalan broj zgrada koje će primiti signal, ako se kula postavi na optimalnu poziciju.

Ulazni podaci

Prvi red ulaza sadrži dva cijela broja N i H razdvojena blankom – broj zgrada i visinu kule($1 \le N \le 1\,000\,000$; za 30% testova važiće $N \le 1000$; $1 \le H, h_1, h_2, \ldots, h_N \le 10^9$). Drugi red sadrži N cijelih brojeva, razdvojenih sa po jednim blankom, koji predstavljaju visine zgrada u redu, od prve do N-te.

Izlazni podaci

U jedini red izlaza štampati jedan cio broj – maksimalan broj zgrada koje će primiti signal, ako se kula izgradi na optimalnoj poziciji.

Test primjeri

Ulaz	Izlaz
12 180 200 170 130 90 150 140 40 30 100 160 50 110	5

Pojašnjenje test primjera

Optimalna pozicija kule je između zdrada 8 i 9. Signal će doći do zgrada 2, 5, 6, 7 i 8.

Rješenje:

Osnovna ideja je da se kula postavi na sva moguća mjesta u redu i da se za svaku poziciju kule prebroji koliko zgrada će primiti signal. Koristićemo frazu kula se postavlja „neposredno iza pozicije m“, koja označava da se kula postavlja između zgrada m i m+1 ili iza zgrade N.

Rješenje složenosti *O(N²)*
Pretpostavimo da je kula neposredno iza pozicije *m*. Prolazimo kroz zgrade, od kule ka zgradi 1. Prolazak se prekida ili dok ne prođemo kroz sve zgrade, ili dok ne naiđemo na zgradu čija je visina veća od visine kule. Ovo rješenje vrijedi 20% bodova.

Rješenje složenosti *O(N²)*, primjenom steka
Pretpostavimo da je kula neposredno iza pozicije *m*. Sad prolazimo kroz zgrade, od 1 do *m*. Zgrada *j* „pokriva“ zgradu *i* (*j*>*i*), ako je $h_j > h_i$ i između njih nema zgrade koja je viša od zgrade *i*. Sada se zadatak svodi na maksimalan broj zgrada koje kula pokriva sa tekuće pozicije.
Koristimo stek koji čuva one zgrade koje do tog trenutka nisu pokrivene od drugih zgrada. Kada stignemo do zgrade *j*, sa vrha steka skidamo sve zgrade čija je visina manja od visine zgrade *j*, i zatim stavljamo *j* na stek. Ovo rješenje i dalje vrijedi 20% bodova, ali daje ideju kako da dobijemo rješenje manje složenosti.

Rješenje složenosti *O(N)*, primjenom steka
Da li je potrebno da pregledamo sve zgrade od 1 do *m*, kada se pozicija kule promijeni? Neka je L_C broj zgrada koje pokriva kula. Opišimo kako se mijenja taj broj kada kulu postavimo na poziciju *m*. Obrađujući zgradu *m*, sa steka uklanjamo sve zgrade čija je visina manja od *m*. Za svaku zgradu koju uklanjamo sa steka i čija je visina manja od H, umanjujemo L_C za 1. Ako je visina *m* manja od H, tada L_C uvećavamo za 1. Vrijednost L_C se mijenja kada se zgrada postavlja na stek ili uklanja iz steka, pa kako se za svaku zgradu to radi tačno jednom, složenost je O(N).

```
#include <cstdio>
#include <stack>
```

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

const int nmax=1000000;
int towers[nmax+2];
int n;
int newtower;
int Lmax=0;
int Lc=0;


void input() {
  scanf("%d %d",&n, &newtower);
  for (int i=1;i<=n;i++){
    scanf("%d",&towers[i]);

  }
}
void calcL() {
  stack<int> st;
  st.push(INT_MAX);

  for ( int i = 1; i <= n; i++ )
  {
    while ( st.top() < towers[i] ) {
      if (st.top() < newtower) Lc--;
      st.pop();

    }
    st.push(towers[i]);
    if (newtower>towers[i]){
        Lc++;
        if (Lc>Lmax){Lmax=Lc;}
    }
  }
}

int main() {
int i;
input();
calcL();

printf("%d\n",Lmax);

return 0;
}
```