ispitni centar
PRAVA
MJERA
ZNANJA

DRŽAVNO TAKMIČENJE 2013.

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 ***Takmicenje2012*** 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!

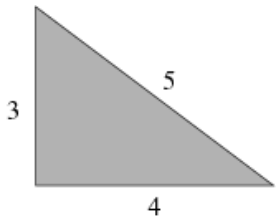
Zadaci

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**

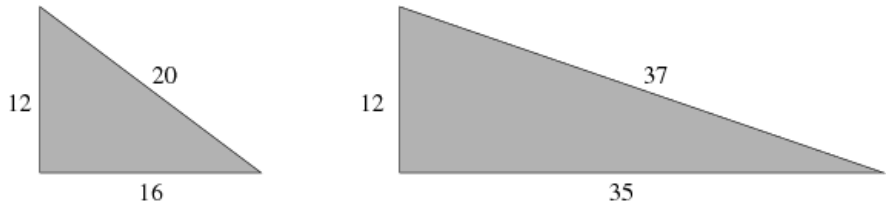
Napomena: Program u C-u i C++-u treba kompajlirati sa sljedećim opcijama: *–O2 –lm –static*, a program u Pascalu sa *–O1 –XS*.

Zadatak 1 – Pitagora

Svima je poznata Pitagorina teorema: u pravouglom trouglu sa dužinama kateta A i B i dužinom hipotenuze C, važi jednakost $A^2 + B^2 = C^2$. Poznato je da postoje pravougli trouglovi čije su sve stranice prirodni brojevi. Na primjer, jedini pravougli trougao čija je manja kateta dužine A=3 prikazan je na slici:



Za A=12 postoje dva trougla:



Za datu vrijednost A, koliko postoji prirodnih brojeva B>A takvih da su A i B katete nekog pravouglog trougla?

Ulazni podaci

Svaki red ulaza sadrži jedan cio broj A, takav da je $2 \le A < 1048576 = 2^{20}$. Kraj ulaza označen je redom koji sadrži broj 0.

Izlazni podaci

Za svaku vrijednost A, štampati koliko ima prirodnih brojeva B > A takvih da trougao sa katetatama A i B ima hipotenuzu čija je dužina prirodan broj.

Test primjeri

Ulaz	Izlaz
3 12 2 1048574 1048575 0	1 2 0 1 175

Napomena: Za vrijednosti $A \approx 2^{20}$, mogu postojati rješenja za koje je $B \approx 2^{39}$, pa je tada $C^2 > B^2 \approx 2^{78}$. Možete koristiti 64-bitne cijele brojeve tipa `long long` u jeziku C++ ili `long` u jeziku Java ili `Int64` u jeziku Pascal.

Nijedan od ovih tipova ne može tačno čuvati vrijednost C^2 za tako veliki broj.

Rješenje: Iz jednakosti $A^2 + B^2 = C^2$ dobijamo $A^2 = C^2 - B^2 = (C-B)(C+B)$. Umjesto da tražimo brojeve veće od A, tražićemo cjelobrojne vrijednosti $X = (C-B)$ i $Y = (C+B)$ takve da je $A^2 = XY$. Sada možemo redom pregledati sve brojeve $1 \le X < A$ da bi našli sve parove (X, Y). Za svaki takav par $X < Y$, važi $X=C-B$, $Y=C+B$, pa je $B = (Y-X)/2$ i $C = (Y+X)/2$. Samo moramo provjeriti da li je $(Y-X)/2$ prirodan broj i da li je veći od A.
Moguća optimizacija je da pregledamo samo cijele brojeve iz intervala $1 \le X < A(\sqrt{2}-1)$.

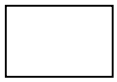
C++:

```
#include <iostream>
#include <fstream>
#include <cmath>
using namespace std;

int main() {
    while (true) {
        long long a;
        cin >> a;
        if (a == 0) break;

        long long count(0);
        for (long long x=1; x <= a/2; x++) {
            if (a*a % x == 0) {
                long long y = a*a / x;
                if ((y-x)%2 == 0) {
                    long long b = (y-x)/2;
                    if (b > a) {
                        count++;
                    }
                }
            }
        }
        cout << count << endl;
    }
}
```

Java:

```
import java.util.*;
import java.io.*;
import static java.lang.Math.*;

public class Zadatak1
{
   public static void main(String[] args) throws Exception {
      Scanner in = new Scanner(new System.in);
      int a  = in.nextInt();
      while (a  > 0) {
         long a2 = (long)a*a;
```

```
            int count = 0;
            for (long x = (long)((sqrt(2) - 1)*a); x > 0; x--)
               if (a2 % x == 0 && (a2/x - x) % 2 == 0)
                  count++;
            System.out.println(count);
            a = in.nextInt();
         }
      }
}
```



Zadatak 2 – Autobusi

U udaljenim oblastima regiona ABC nalazi se N sela. Između nekih sela postoje autobuske linije. Broj putnika nije veliki, pa autoprevozno preduzeće „ABC Trans" ima samo nekoliko linija dnevno. Marija i Aleksa stanuju u selu d i hitno moraju otići u selo v da bi pomogli baki koja priprema zimnicu. Smatramo da se Marija i Aleksa u trenutku 0 nalaze u selu d.

Ulazni podaci

U prvom redu ulaza nalazi se cio broj N, $1 \leq N \leq 100$ – broj sela u regionu. U drugom redu su dva cijela broja d i v, razdvojena jednim blankom – brojevi sela u kojim žive Marija i Aleksa i njihova baka. Treći red sadrži broj R, $1 \leq R \leq 10000$ – broj autobuskih linija. Sljedećih R redova opisuju autobuske linije. Svaka linija opisuje se sa 4 broja razdvojena sa po jednim blankom: broj sela iz kojeg polazi linija, vrijeme polaska, broj sela u kojem završava linija i vrijeme dolaska. Sva vremena su cijeli brojevi iz intervala [0, 10000]. Ako putnik stiže u neko selo u trenutku t, on može otići iz tog sela u trenutku t1, $t \leq t1$.

Izlazni podaci

U jedinom redu izlaza štampati jedan cio broj – minimalno vrijeme potrebno Mariji i Aleksi da dođu u selo v. Ako nije moguće iz sela d doći do sela v, štampati broj -1 (minus 1).

Test primjeri

Ulaz	Izlaz
3 1 3 4 1 0 2 5 1 1 2 3 2 3 3 5 1 1 3 10	5

Napomena: Marija i Aleksa mogu da krenu u trenutku 1 iz sela 1 i da u trenutku 3 budu u selu 2. Odmah imaju autobus za selo 3, i stižu u trenutku 5, što je traženo minimalno vrijeme.

Rješenje:

Kreirajmo graf čiji su čvorovi sela a grane autobuske linije. Težina grane je vrijeme dolaska u selo (npr.za liniju 1 0 2 5, grana je (1,2) sa težinom 5). Putanja u ovom grafu je niz grana takav da je težina puta do svakog čvora v na tom putu ne prelazi vrijeme koje odgovara sljedećoj grani. Na ovako konstruisanom grafu primjenimo Dijkstrin algoritam.

C++:

```
#include <iostream>
#include <cstdio>
#include <vector>

using namespace std;

const int INF = 2000000;
const int size = 101;
struct quad {int src, dep, dest, arr;}; // (polazna planeta, vrijeme
polaska, dolazna planeta, vrijeme dolaska)

int main()
{
      int n; // broj cvorova
      int p; // pocetni cvor
      int q; // zavrsni cvor
      int r; // broj planetarnih linija
      struct quad par;

      //... ucitavanje n ...
      cin >> n >> p >> q >> r;
    p--;
    q--;
      vector < vector < quad > > g (n);

      //... ucitavanje grafa ...
    for (int i=0; i<r; ++i)
    {
        cin >> par.src >> par.dep >> par.dest >> par.arr;
        par.src--; // indeksi krecu od 0
        par.dest--; // indeksi krecu od 0
        g[par.src].push_back(par);
    }

    vector<int> d (n, INF),  pr (n); // nizovi vremena (d) i roditalja
(pr)
    d[p] = 0;
      vector<char> u (n); // da li je cvor posjecen ili ne
      for (int i=0; i<n; ++i) {
            int v = -1;
            for (int j=0; j<n; ++j)
                  if (!u[j] && (v == -1 || d[j] < d[v]))
                        v = j;
            if (d[v] == INF)
                  break;
            u[v] = true; // sada je v minimalni cvor

            for (size_t j=0; j<g[v].size(); ++j) {
```

```
                  par = g[v][j];
                  //int to = g[v][j].first,
                  //    len = g[v][j].second;
                  if ((d[v] <= par.dep) && (d[par.dest]>par.arr)) {
                        d[par.dest] = par.arr;
                        pr[par.dest] = v;
                  }
            }
      }

    if (d[q] != INF)
        cout << d[q];
    else
        cout << -1;
    return 0;
}
```

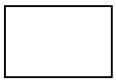
Java:

```
import java.util.Arrays;
import java.util.Scanner;
import java.util.ArrayList;

public class zadatak2 {

      public void run()
      {
            solve();
      }

      private static class Quad
      {
            int src, dep, dest, arr;; // (polazna planeta, vrijeme
polaska, dolazna planeta, vrijeme dolaska)
            public Quad(int a, int b, int c, int d)
            {
                  src = a;
                  dep = b;
                  dest = c;
                  arr = d;
            }
      }

      public void solve()
      {

            final int INF = 2000000; // podesiti ovu vrijednost na
zeljenu

            int n; // broj cvorova
            int p; // pocetni cvor
            int q; // zavrsni cvor
            int r; // broj planetarnih linija
            int a1 , b1, c1, d1; // pomocne promjenljive
            Quad par; // pomocna promjenljiva, za upisivanje u listu
susjedstva

            // ucitavanje prvog reda
            Scanner in = new Scanner(System.in);
```

```
            n = in.nextInt();
            p = in.nextInt();
            q = in.nextInt();
            r = in.nextInt();

            p--;
            q--;

            ArrayList<ArrayList<Quad>> g = new
ArrayList<ArrayList<Quad>>(n); // graf je predstavljen listama
susjedstva

            for (int i = 0; i<n; i++)
            {
                  g.add(new ArrayList<Quad>());
            }
            //... ucitavanje grafa ...
            for (int i=0; i<r; ++i)
            {
                  a1 = in.nextInt();
                  b1 = in.nextInt();
                  c1 = in.nextInt();
                  d1 = in.nextInt();
                  a1--; // nasi indeksi krecu od 0
                  c1--; // nasi indeksi krecu od 0
                  ArrayList<Quad> t = g.get(a1);
                  t.add(new Quad(a1,b1,c1,d1)); // g.get(a).add(new
Quad(a1,b1,c1,d1));
            }

            int [] d = new int[n]; // niz vremena
            int [] pr = new int[n]; // niz roditalja
            boolean[] u = new boolean[n]; // da li je cvor posjecen
ili ne
            Arrays.fill(d, INF);
            Arrays.fill(u, false);
            d[p] = 0;
            for (int i=0; i<n; ++i) {
                  int v = -1;
                  for (int j=0; j<n; ++j)
                        if (!u[j] && (v == -1 || d[j] < d[v]))
                              v = j;
                  if (d[v] == INF)
                        break;
                  u[v] = true; // sada je v minimalni cvor


                  for (int j=0; j<g.get(v).size(); ++j) {
                        par = g.get(v).get(j);
                        if ((d[v] <= par.dep) &&
(d[par.dest]>par.arr)) {
                              d[par.dest] = par.arr;
                              pr[par.dest] = v;
                        }
                  }
            }


            if (d[q] != INF)
                  System.out.println(d[q]);
```

```
		else
			System.out.println(-1);

	}

	public static void main(String[] args) {

		new zadatak2().run();
	}

}
```

Zadatak 3 – Vrtić



Jedan od alata za učenje slikanja u vrtiću je i pravougaonik dimenzija NxM centimetara, podijeljen na kvadratiće dimenzija 1x1, pri čemu su stranice kvadrata paralelene stranicama pravougaonika. Svaki kvadratić treba obojiti jednom do 26 boja, koje su označene malim slovima engleske abecede.
Vrtić organizuje prodajnu izložbu slika dobijenih na ovaj način. Cijena slike jednaka je broju „simetričnih" regiona na slici. Region slike je bilo koji pravougaonik koji možemo izrezati iz slike po granicama kvadratića. Region je simetričan ako se simetrijom u odnosu na centar pravougaonika dobija pravougaonik koji je isti kao i polazni. Na primjer, na slici

a	b	c
a	c	b

simetrični su svi regioni koji se sastoje od samo jednog kvadratića (ima ih 6) kao i sljedeća dva regiona:

b	c
c	b

a
a

Napišite program koji na osnovu zadate slike određuje njenu cijenu.

Ulazni podaci

U prvom redu ulaza su dva cijela broja N i M (1 ≤ N, M ≤ 100), razdvojena jednim blankom. U sljedećih N redova dati su stringovi koji se sastoje od M malih slova engleske abecede. Simbol i u redu j zadaje boju odgovarajućeg kvadratića slike.

Izlazni podaci

U jedinom redu izlaza štampati jedan cio broj – cijenu slike tj. broj simetričnih regiona na slici.

Test primjeri

Ulaz	Izlaz	Komentar
2 3 abc acb	8	Ovo je primjer iz teksta zadataka
2 3 ab cc ba	8	Simetrični regioni su 6 kvadratića 1x1, jedan region 1x2 (naime cc) i cijela slika

Ograničenja:

U prvoj grupi testova, koja vrijedi 30%, važi N, M ≤ 15.
U drugoj grupi testova, koja vrijedi 60%, važi N, M ≤ 50.

Rješenje: Prikažimo prvo rješenje za 30% bodova. Pregledajmo na slici sve regione (pravougaonike) redom i provjeravamo da li su simetrični. Provjera za simetričnost je složenosti O(MN), jer se mora pregledati polovina regiona. Pravougaonik biramo zadavanjem 4 tačke: gornje i donje granice (od 1 do N) i lijeve i desne granice (od 1 do M). Kako gornja granica ne smije preći donju, ukupan broj takvih tačaka je N(N-1)/2 tj. $O(N^2)$. Slično je za lijevu i desnu granicu $O(M^2)$, pa je ukupna složenost ovog algoritma $O(N^3M^3)$.
Rješenje za 60% bodova: Optimiziramo prethodno rješenje, u kojem smo provjeravali simetričnost regiona nezavisno od prethodnih regiona. Nek je A region čija je širina veća od 2. Ako iz A odrežemo lijevi i desni kraj, dobijamo pravouganik B čiji se centar poklapa sa centrom pravougaonika A. Dakle, simetrija B je neophodan uslov za simetričnost A, a ostaje još da provjerimo simetričnost slova iz krajnjih kolona u A. Složenost ovog algoritma je $O(N^3M^2)$, jer imamo provjeru po gornjoj i donjoj granici , a unutar toga postepeno uveličavamo širinu za O(N).
Rješenje za 100% bodova: Pokažimo kako je moguće provjeriti simetričnost regiona A za O(1). Posmatrajmo pravougaonike B i C dobijene iz A: B se dobija odrezivanjem krajnje lijeve i krajnje desne kolone iz A, dok se C dobija odrezivanjem krajnje gornje i krajnje donje vrste iz A. Da bi A bio simetričan neophodno je i dovoljno da su i B i C simetrični i da su suprotni ugaoni elementi u A jednaki. Sada svaki region možemo provjeriti za O(1), što daje ukupnu složenost $O(N^2M^2)$. Jedina teškoća je što moramo u svakom trenuku vremena moramo imati informaciju o simetričnosti B i C. Nije moguće pamtiti informaciju o svim pravougaonicima, zbog memorijskog ograničenja. Moguće je više načina da se to izbjegne:

- Nezavisna izračunavanja za svaki centar – memorija je kvadratna
- Ukupan broj regiona ne može biti veći od $(100*99)^2/2$, možemo ih numerisati i dobiti niz koji je 4 puta manji
- Koristiti niz bitova.

Postoji i još jedno rješenje za 100%, koje koristi polinomijalnu heš-funkciju i čija je složenost O(NM(N+M)).

C++:

```
#include <iostream>
using namespace std;
char b[13][100][100][100] = {{{{0}}}};
void place1(int x1, int x2, int y1, int y2)
{
      b[x1>>3][x2][y1][y2] += 1<<(x1&7);
      return;
}
bool get(int x1, int x2, int y1, int y2)
{
      if (x1 > x2 || y1 > y2) return 1;
      return (b[x1>>3][x2][y1][y2] & (1 << (x1 & 7)));
```

```
}
int main()
{
      int n, m;
      cin >> n >> m;
      string a[100];
      for (int i = 0; i < n; ++i)
      cin >> a[i];
      int ans = 0;
      for (int dx = 1; dx <= n; ++dx)
        for (int dy = 1; dy <= m; ++dy)
          for (int x = 0; x + dx <= n; ++x)
            for (int y = 0; y + dy <= m; ++y)
      if (get(x+1, x+dx-2, y, y+dy-1) && get(x, x+dx-1, y+1, y+dy-2)
      &&   a[x][y] == a[x+dx-1][y+dy-1] && a[x][y+dy-1] == a[x+dx-
      1][y])
      {
            place1(x, x+dx-1, y, y+dy-1);
            ans++;
      }
      cout << ans << endl;
      return 0;
}
```