ispitni centar
PRAVA
MJERA
ZNANJA

DRŽAVNO TAKMIČENJE 2016.

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 ***Takmicenje2016*** 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 – Kocka

Ana je na stolu pronašla običnu kockicu s brojevima 1-6, kao u igri „Čovječe ne ljuti se". Kockica izgleda kao na slici. Zbir brojeva na suprotnim stranama kockice uvijek je 7. Ana je postavila kockicu u gornje lijevo polje matrice sa **R** redova i **S** kolona. Okrenula je tako da se na gornjoj strani nalazi broj 1, a na desnoj broj 3.
Zatim počne kotrljati kockicu prema sljedećim pravilima:
1. Kotrlja kockicu udesno, sve dok ne dođe do posljednje kolone
2. Otkotrlja je jedno polje prema dolje (u sljedeći red)
3. Kotrlja kockicu ulijevo, sve dok ne dođe do prve kolone
4. Otkotrlja je jedno polje prema dolje (u sljedeći red)
Ove korake Ana ponavlja sve dok može, tj. dok god je moguće pomjeranje u sljedeći red. Za svako polje u matrici Ana zapiše broj koji se pojavljuje na gornjoj strani kockice u trenutku kada se kockica nađe na tom polju, i na kraju sabere sve zapisane brojeve. Pomozite Ani i napišite program koji će izračunati zbir zapisanih brojeva.

Ulazni podaci

U prvom i jedinom redu nalaze se dva prirodna broja **R** i **S** ($1 \leq$ **R**, **S** ≤ 100000), dimenzije matrice.

Izlazni podaci

U prvi i jedini red štampajte traženi zbir.

Bodovanje

U 50% test podataka brojevi **R** i **S** biće manji ili jednaki od 100.

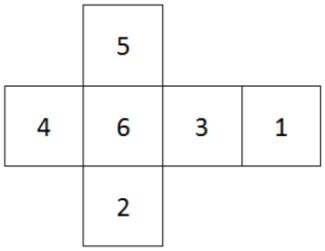
Test primjeri

Ulaz	Izlaz
3 2	19
3 4	42
737 296	763532

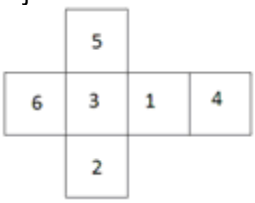
Objašnjenje prvog primjera: Zapisani brojevi su:

1	4
1	5
3	5

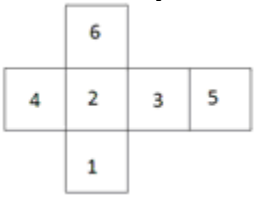
Rješenje:
Rastvorena kocka (ili mreža kocke) ima sljedeći oblik:



Ova mreža odgovara kocki na početnoj poziciji u matrici. Ako napravimo jedan potez udesno, brojevi 4-6-3-1 ciklično se zarotiraju ulijevo do 6-3-1-4 tj dobijamo mrežu:



Primjetite da kotrljanje ulijevo ima isti efekat kao tri kotrljanja udesno. Na isti način zaključimo šta se dobija kotrljanjem nadolje. Ako početnu kocku kotrljamo nadolje dobijamo sljedeću mrežu:



Polja 2-6-5-1 sa prve slike postaju 1-2-6-5 tj. imamo cikličnu rotaciju udesno.

Rješenje za 70%: Simuliramo kretanje kockice po matrici. Složenost je O(RS).

Rješenje za 100%: Izračunajmo zbir u jednom redu. Primjetimo da 4 kotrljanja kockice u redu daju zbir 14 (4+6+3+1=14) a kockica se vraća u početnu poziciju. Dakle zbir u redu je 14* X, gdje je X=(S-1) div 4, a preostali dio reda, koji ima najviše 3 kotrljanja, nađemo simulacijom. Složenost je O(R).

```
#include <cstdio>

using namespace std;

struct dice{
  int row[4];
  int up, down;

  dice(){
    row[0] = 4;
    row[1] = 6;
    row[2] = 3;
```

```
    row[3] = 1;
    up = 5;
    down = 2;
  }

  void roll_down(){
    int old_up = up;

    up = row[1];
    row[1] = down;
    down = row[3];
    row[3] = old_up;
  }

  void roll_right(){
    int old_left = row[0];

    for( int i = 0; i < 3; i++ )
      row[i] = row[i+1];
    row[3] = old_left;
  }

  void roll_left(){
    roll_right();
    roll_right();
    roll_right();
  }

  void roll( bool dir ){
    if( !dir ) roll_right();
    else roll_left();
  }

  int top(){ return row[3]; }
  int sum_row(){ return row[0] + row[1] + row[2] + row[3]; }
//uvijek 14
};

int main(){
  dice curr;

  int r, c;
  scanf( "%d %d", &r, &c );

  long long sol = 0;
  bool dir = 0;

  for( int row = 0; row < r; row++, dir = !dir ){
    sol += curr.top();

    int repeat = (c-1) / 4;
    sol += repeat * curr.sum_row();

    int rest = (c-1) % 4;
    for( int i = 0; i < rest; i++ ){
```

```
      curr.roll(dir);
      sol += curr.top();
    }

    curr.roll_down();
  }

  printf( "%I64d\n", sol );
  return 0;
}
```

Zadatak 2 – Sretni brojevi

Mirku je na času matematike postalo dosadno, pa je na papir napisao **N** velikih brojeva. Potom je primijetio da mu se neki parovi od tih brojeva sviđaju, a neki drugi parovi ne sviđaju. Za parove brojeva koji mu se sviđaju Mirko je osmislio i naziv. Za **dva broja** reći ćemo da su **srećni par** ako imaju **barem jednu zajedničku cifru,** ne obavezno na istoj poziciji. Pomozite Mirku i recite mu koliko na njegovom papiru ima srećnih parova.

Ulazni podaci

U prvom redu ulaza nalazi se prirodan broj **N** ($1 \leq \mathbf{N} \leq 1\ 000\ 000$). U sljedećih **N** redova nalazi se po jedan prirodan broj iz intervala $[1, 10^{18}]$. To su brojevi koje je Mirko napisao na papir; među njima neće biti jednakih.

Izlazni podaci

U jedini red izlaza štampati traženi broj srećnih parova.

Test primjeri

Ulaz	Izlaz
3 4 20 44	1
4 32 51 123 282	4

Rješenje:

Primijetite da je za svaki od unesenih brojeva važno samo koji skup cifara taj broj sadrži tj. nije bitan redoslijed ni broj pojavljivanja tih cifara. Unesenom broju pridružimo binarni niz od 10 nula ili jedinica koji predstavlja skup njegovih cifara: ako je npr. unesen broj 12421, tada će njemu odgovarati binarni niz 0110100000, jer u broju 12421 imao cifre 1, 2 i 4, pa su bitovi na tim pozicijama 1, dok su svi ostali jednaki 0. Ovakvih binarnih nizova ima najviše $2^{10} = 1024$. Za svaki od tih nizova, koje možemo predstaviti cijelim brojevima od 0 do 1023, sačuvamo u nizu kol koliko od unesenih brojeva odgovara upravo tom nizu. Lako se provjerava da li dva broja imaju bara jednu cifru – njihovi binarni nizovi

imaju bar jednu zajedničku jedinicu. Ako imaju, onda srećni par čine bilo koja dva broja koja generišu te binarne nizove, pa ukupnom broju dodajemo sljedeći sabirak: kol[prvi niz] * kol[drugi niz].
Preostaje još da prebrojimo parove koji generišu isti binarni niz a njih ima:
kol[niz] * (kol[niz] - 1) / 2

Složenost algoritma, u najgorem slučaju, je broj parova binarnih nizova dužine 10, tj $1024*1024 = 2^{20}$.

```
#include<cstdio>
#include<ctime>
#include<cstring>
#include<iostream>

using namespace std;

long long sol;
int kol[1<<10];

int main()
{
    int n; scanf("%d", &n);
    char buff[22];
    while(n--)
    {
              scanf("%s", buff);
              int s = strlen(buff);
              int mask = 0;
              for(int i=0; i<s; ++i)
                     mask |= (1<<(buff[i]-'0'));
              kol[mask]++;
    }
    for(int i=0; i<1024; i++)
            for(int j=i+1; j<1024; j++)
                    if( i & j ) sol += (long long)kol[i] *
kol[j];
    for(int i=0; i<1024; i++) sol += (long long)kol[i]*(kol[i]-
1)/2;
    cout << sol << endl;
    return 0;
}
```

Zadatak 3 – Smartphone

Pavle programira aplikaciju za pametni telefon koja će pomoći posjetiocima tržnog centra da pronađu najkraću šetačku putanju između dvije lokacije u centru. Ako zadamo trenutnu lokaciju posjetica i željenu lokaciju, aplikacija pokazuje mapu centra i na njoj najkraću putanju izraženu u metrima. Tržni centar ima N lokacija koje se nalaze na više spratova i koje su povezane pješačkim stazama, stepenicama, liftovima i pokretnim stepenicima (eskalatorma). Obratite pažnju da najkraća putanja u metrima može uključiti i upotrebu eskalatora u suprotnom smjeru. Aplikacija mjeri koliko je metara prepješačio posjetilac, pa različiti metodi kretanja imaju razne cijene izražene metrima:

- ako posjetilac hoda ili koristi stepenice, rastojanje se izražava euklidskim rastojanjem lokacija.
- Ako koristi lift, rastojanje je 1 metar, jer u liftu nema pješačenja. Jedan lift povezuje tačno dvije lokacije. Lift povezuje iste lokacije koje se nalaze na različitim spratovima. Npr. ako postoje tri sprata i lift je na poziciji (1,2) na svakom spratu, tada se to u ulaznim podacima opisuje sa: (0; 1; 2) - (1; 1; 2), (1; 1; 2) - (2; 1; 2) i (0; 1; 2) - (2; 1; 2). Ako na nekim mapama lift ne povezuje sve spratove, tada neke od veza neće postojati u ulaznim podacima.
- Eskaltor ima dvostruku namjenu
    - Pomjeranje od A do B (u pravom smjeru) – rastojanje je 1, jer napravimo nekoliko koraka a dalje se kreće eskalator
    - Pomjeranje od B do A (u suprotnom smjeru) – rastojanje je euklidsko rastojanje pomnoženo sa 3,

Najkraća šetnja mora uključivati samo neke od navednih lokacija. Sve lokacije su međusobno povezane bar jednom putanjom.

Ulazni podaci

Ulazni podaci sadrže opis mape tržnog centra i određeni broj parova lokacija za koje treba odrediti najkraće rastojanje.

U prvom redu ulaza nalazi se dva prirodna broja N i M ($1 \leq N \leq 200$, $N-1 \leq M \leq 1000$) – broj lokacija u centru i broj veza između lokacija. Lokacije su numerisane brojevima od 0 do N-1. Sljedećih N redova opisuju lokacije, po jednu u redu, i sadrže po tri cijela broja: sprat i x i y koordinatu lokacije na spratu. Rastojanje između spratova je 5 metara, a x i y koordinata se izražavaju u metrima.

Sljedećih M redova opisuju direktne veze između lokacija u centru. Svaka veza je definisana brojevima lokacija i tipom kretanja (jedan od sljedećih: walking, stairs, lift ili escalator). Za lokacije na istom spartu tip kretanja je walking.

Sljedeći red sadrži jedan cio broj Q ($1 \leq Q \leq 1000$) – broj parova za koje tražimo najkraće rastojanje. Sljedećih Q redova sadrže po dvije lokacije za koje određujemo najkraću šetačku putanju.

Izlazni podaci

Štampati Q redova. U svakom redu štampati najkraću šetačku putanju za date lokacije, pri čemu brojeve razdvajati jednim blankom.

Test primjeri

Ulaz	Izlaz
6 7 3 2 3 3 5 3 2 2 3 2 6 4 1 1 3 1 4 2 0 1 walking 0 2 lift 1 2 stairs 2 3 walking 3 4 escalator 5 3 escalator 4 5 walking 5 0 1 1 2 3 5 5 3 5 1	0 1 1 0 2 3 4 5 5 3 5 3 2 0 1

Rješenje:

Kreira se težinski graf u skladu sa postavkom zadatka. Primjenom Flojd-Varšalovog algoritma ili Dijkstrin-og algoritma odrede se najkraće putanje između datih parova čvorova u grafu.

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <iomanip>
#include <iostream>
#include <map>
#include <string>
#include <vector>
using namespace std;

const int MAXPLACES = 1000;
const double INF = 848238233;

struct Place {
    int x, y, floor;
```

```
    Place(int x, int y, int floor) : x(x), y(y), floor(floor)
{};
    Place() : x(0), y(0), floor(0) {};
};
vector<Place> places(MAXPLACES);

// Structures for Floyd-Warshall
vector<vector<double> > dist(MAXPLACES,
vector<double>(MAXPLACES, INF));
vector<vector<int> > parent(MAXPLACES, vector<int>(MAXPLACES, -
1));

// Euclidean distance between 2 places
double euc(const struct Place& a, const struct Place& b) {
    int difx = a.x - b.x;
    int dify = a.y - b.y;
    int diff = 5*(a.floor - b.floor);
    return sqrt(difx*difx + dify*dify + diff*diff);
}

int main(int argc, char *argv[]) {
    int N, M, a, b;
    string name, type;

    cin >> N >> M;
    // Read positions
    for (int i = 0; i < N; ++i) {
        cin >> places[i].floor >> places[i].x >> places[i].y;
        dist[i][i] = 0;
    }

    // Read connections, keeps the minimum for each pair of
nodes
    for (int i = 0; i < M; ++i) {
        cin >> a >> b >> type;
        double d = euc(places[a], places[b]);
        if (type[1] == 's') { // escalator
            dist[a][b] = min(dist[a][b], 1.);
            dist[b][a] = min(dist[b][a], d*3);
        } else if (type[0] == 'l') { // lift
            dist[a][b] = min(dist[a][b], 1.);
            dist[b][a] = min(dist[b][a], 1.);
        } else { // walking, stairs
            dist[a][b] = min(dist[a][b], d);
            dist[b][a] = min(dist[b][a], d);
        }
    }

    // Floyd-Warshall
    // Prepare the path recovery
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            if (i == j || dist[i][j] == INF)
                parent[i][j] = -1;
            else
```

```
                parent[i][j] = i;

    // Compute shortest path
    for (int k = 0; k < N; ++k)
        for (int i = 0; i < N; ++i)
            for (int j = 0; j < N; ++j) {
                double newD = dist[i][k] + dist[k][j];
                if (newD < dist[i][j]) {
                    dist[i][j] = newD;
                    parent[i][j] = parent[k][j];
                }
            }

    // Debug
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j)
            if (dist[i][j] != -1)
                cerr << " " << parent[i][j];
            else
                cerr << " X";
        cerr << endl;
    }

    // Queries
    vector<int> result(MAXPLACES);
    int Q;
    cin >> Q;
    for (int q = 0; q < Q; ++q) {
        cin >> a >> b;
        cerr << a << " " << b << " dist: "
             << setprecision(8) << dist[a][b] << endl;

        // Recover path from a to b
        int i = 0;
        result[i++] = b;
        while ((b = parent[a][b]) != -1)
            result[i++] = b;
        for (int j = i-1; j >= 0; --j) {
            if (j < i-1)
                cout << " ";
            cout << result[j];
        }
        cout << endl;
    }
    return EXIT_SUCCESS;
}
```