**Colegiul Național Octav Onicescu**

**Olimpiada Județeană de Informatică**

**Elev: Dănoiu Antonie Profesor coordonator:**

**Clasa: a XI-a A Doina Dincă Mariana**

**An școlar: 2023-2024**

**Cuprins**

1. Scurtă prezentare

2. Noțiuni teoretice

a. Tehnici de programare

b. Algoritmi

3. Probleme rezolvare

4. Site realizat în HTML

5. Bibliografie

**1. Scurtă prezentare**

Olimpiada Județeană de Informatică, prescurtată OJI, se conturează ca un pilon esențial în peisajul educațional, oferind o platformă valoroasă pentru cultivarea interesului și talentului în domeniul informaticii. Această competiție nu doar încurajează pasiunea pentru tehnologie, ci și stimulează dezvoltarea abilităților practice și teoretice ale elevilor, pregătindu-i pentru provocările și oportunitățile oferite de societatea digitală contemporană.

Într-o lume în continuă evoluție tehnologică, participarea la OJI devine din ce în ce mai relevantă. Elevii nu numai că învață să scrie cod și să rezolve probleme complexe, ci și să aplice concepte și tehnici fundamentale de programare și algoritmi. De la algoritmi de sortare și căutare, precum QuickSort sau Binary Search, până la tehnici avansate de optimizare și eficientizare a codului, cum ar fi programarea dinamică, această competiție oferă oportunități inestimabile de învățare și dezvoltare.

**2. Noțiuni teoretice**

Pentru a putea participa la OJI și a obține un punctaj bun, trebuiesc invățate atât tehnici de programare, cât și anumiți algoritmi.

**a. Tehnici de programare**

**Divide et Impera**

- Este o metodă eficientă, având mereu o complexitate logaritmică.

- Se folosește la problemele pe vectori (sortare, căutarea unui număr, secvențe, etc.)

- De obicei, la OJI se dau multe probleme de acest tip, dar de multe ori pot fi folosite funcții din STL și astfel nu mai trebie implementat de mână algoritmul.

**Backtracking**

- Este o metodă ineficientă, întrucât va testa toate cazurile posibile ale unei probleme (care pot fi si câteva milioane)

- Se folosește la unele probleme de combinatorică și la anumite generări de numere, anagrame etc.

- De obicei, la OJI nu se dau probleme care sa necesite o astfel de rezolvare.

**Greedy**

- Este o metodă eficientă, care are o abordare foarte simplă, salvând tot felul de soluții și la final returnând-o pe cea mai bună.

- De obicei are complexitate pătratică sau în unele cazuri liniară.

- Există foarte mulți algoritmi care au o abordare de tip Greedy, deci prin urmare există și foarte multe probleme care se rezolvă cu această metodă.

- La OJI se dau foarte des probleme de acest tip, mai ales cu grafuri.

**Programare dinamică**

- Este cea mai eficientă metodă de programare, întrucât poate reduce complexitatea unor probleme de la pătratică la liniară.

- Se poate aplica pe foarte multe probleme de tip Greedy, dar și pe unele de numărare și combinatorică.

- La OJI, această tehnică este folosită pentru a eficientiza problemele.

**b. Algoritmi**

Exisă foarte mulți algoritmi care se pot aplica pe diverse structuri de date, dar cei mai complexi și greu de înțeles sunt cei aplicați grafuri. La OJI se dau foarte des probleme care se rezolvă folosind Dijkstra sau Kruskal.

**Dijkstra**

Este un algoritm folosit în grafurile cu costuri și determină distanța de la un nod la toate celelalte.

**Kruskal**

Este un algoritm folosit în grafurile cu costuri și determină arborele parțial de cost minim.

**3. Probleme rezolvate**

**Dragoni**

Supărați că lansarea părții a treia a filmului lor preferat s-a amânat până în iunie 2018, Henry și Hetty s-au gândit la propriul scenariu pentru finalul trilogiei: Într-o lume în care vikingii pot zbura cu dragonii există N insule. Hiccup, șeful tribului de vikingi aflat pe insula 1, știe M rute directe de zbor bidirecționale între insule. Pentru fiecare j intre 1 si M, ruta j unește insulele Aj și Bj și are lungime Dj. Pe fiecare insulă i, (1 ≤ i ≤ n) există dragoni din specia i care pot zbura fără a se opri pentru odihnă o distanță maximă Dmaxi. Cu alte cuvinte, dragonii de pe insula i vor putea parcurge orice rută j, (1 ≤ j ≤ m) pentru care Dj ≤ Dmaxi, indiferent de ce alte drumuri au făcut anterior. Hiccup dorește să ajungă de pe insula 1 pe insula N pentru a-l salva pe Toothless, dragonul lui. Pentru a ajunge acolo, el va lua inițial un dragon specia 1 (de pe insula 1). Apoi, dacă la un moment dat Hiccup se află pe o insula i, (1 ≤ i ≤ n) având cu el un dragon din specia t, el poate:  
1. Să zboare de pe insula i pe o altă insulă x cu dragonul pe care îl are, folosind o rută directă j între insulele i si x, bineînțeles doar dacă Dj ≤ Dmaxt.  
2. Să schimbe dragonul din specia t pe care îl are cu un dragon din specia i aflat pe insula respectivă.

### Cerințe

1. Să se determine distanța maxima Dmaxi caracteristică unui dragon la care Hiccup poate ajunge fără a schimba dragonul pe care l-a luat inițial de pe insula 1.  
2. Să se determine distanța minimă pe care Hiccup trebuie să o parcurgă pentru a ajunge de pe insula 1 pe insula N.

### Date de intrare

Fișierul de intrare dragoni.in conține pe prima linie numărul p. Pentru toate testele de intrare, numărul p poate avea doar valoarea 1 sau valoarea 2. Pe a doua linie se găsesc două numere naturale N și M reprezentând numărul de insule, respectiv numărul de rute directe între insule. Pe a treia linie se găsesc N numere naturale, al i-lea dintre acestea reprezentând distanta maximă Dmaxi pe care o poate zbura un dragon de pe insula i. Pe următoarele M linii sunt descrise cele M rute directe. Pe fiecare dintre aceste linii se găsesc câte trei numere naturale A, B și D cu semnificația că există rută bidirecțională de lungime D între insulele A și B.

### Date de ieșire

În fișierul de ieșire dragoni.out se va afișa un singur număr natural.  
Dacă valoarea lui p este 1, se rezolvă numai cerința 1. În acest caz numărul afișat va reprezenta distanța maximă Dmaxi a unui dragon i la care Hiccup poate ajunge fără a schimba dragonul pe care l-a luat inițial de pe insula 1.  
Daca valoarea lui p este 2, se va rezolva numai cerința 2. În acest caz numărul afișat va reprezenta distanța minima pe care Hiccup trebuie să o parcurgă pentru a ajunge de pe insula 1 pe insula N.

### Restricții și precizări

• 1 ≤ N ≤ 800  
• 1 ≤ M ≤ 6000  
• 1 ≤ Dmaxi ≤ 50 000, pentru orice 1 ≤ i ≤ N.  
• 1 ≤ Aj, Bj ≤ N, pentru orice 1 ≤ j ≤ M.  
• 1 ≤ Dj ≤ 50 000, pentru orice 1 ≤ j ≤ M.  
• Se garantează că Hiccup poate ajunge pe insula N.  
• Se garantează că răspunsul oricărei cerințe este un număr natural mai mic decât 109.  
• Pentru rezolvarea corectă a primei cerințe se acordă 20% din punctajul testului respectiv.  
• Pentru rezolvarea corectă a celei de-a doua cerințe se acordă 80% din punctajul testului respectiv.

Rezolvare

#include <fstream>

#include <vector>

#include <queue>

using namespace std;

ifstream fin ("dragoni.in");

ofstream fout("dragoni.out");

#define pair pair <int,int>

#define vecin first

#define cost second

#define pq priority\_queue <pair, vector <pair>, greater <pair>>

const int Inf = 1 << 30;

const int Max = 8e2 + 1;

int p, n, m;

vector <pair> graf[Max];

vector <int> D(Max);

void read()

{

    int x, y, z;

    fin >> p >> n >> m;

    for (int i = 1; i <= n; ++i)

        fin >> D[i];

    for (int i = 1; i <= m; ++i)

    {

        fin >> x >> y >> z;

        graf[x].push\_back({y, z});

        graf[y].push\_back({x, z});

    }

}

void dfs(int nod, int &r, bool v[])

{

    v[nod] = true;

    r = max(r, nod);

    for (auto i : graf[nod])

        if (!v[i.vecin] && D[1] >= i.cost)

            dfs(i.vecin, r, v);

}

void solve1()

{

    bool v[Max];

    int r = 1;

    dfs(1, r, v);

    fout << D[r];

}

void dijkstra(int nod, vector <int> &d)

{

    pq q;

    q.push({0, nod});

    int x = nod;

    d[nod] = 0;

    while (!q.empty())

    {

        nod = q.top().second;

        q.pop();

        for (auto i : graf[nod])

        {

            if (d[nod] + i.cost < d[i.vecin] && i.cost <= D[x])

            {

                d[i.vecin] = d[nod] + i.cost;

                q.push({d[i.vecin], i.vecin});

            }

        }

    }

}

void solve2()

{

    vector <vector<int>> d(n + 1);

    vector <int> dp(n + 1);

    for (int i = 1; i <= n; ++i)

    {

        d[i].assign(n + 1, Inf);

        dijkstra(i, d[i]);

        dp[i] = d[1][i];

    }

    for (int i = 1; i <= n; ++i)

        for (int j = 1; j <= n; ++j)

            if (dp[i] > dp[j] + d[j][i])

                dp[i] = dp[j] + d[j][i];

    fout << dp[n];

}

int main()

{

    read();

    if (p == 1)

        solve1();

    else

        solve2();

    fin.close();

    fout.close();

    return 0;

}

**Ninjago**

După ce eroii ninja l-au învins pe Nadakhan, de ziua celor dispăruți Zane trebuia să păzească cele n păpuși din muzeu. Între aceste păpuși există m coridoare pe care se poate circula în ambele sensuri. Se garantează faptul că pe cele m coridoare Zane poate ajunge la fiecare dintre cele n păpuși. Skulkiu, având la dispoziție 5 tipuri de obstacole A, B, C, D, E, încearcă să-l oprească pe Zane punând pe fiecare coridor câte 4 obstacole. Zane poate distruge obstacolele de tip A, B, C și D, dar nu poate să distrugă obstacolele de tipul E. Pentru a distruge un obstacol de tipul A arma lui Zane are nevoie de 1 unitate de energie, pentru a distruge un obstacol de tipul B de 2 unități de energie, pentru a distruge un obstacol de tipul C de 3 unități de energie, iar pentru a distruge un obstacol de tipul D de 4 unități de energie. Datorită dispozitivului cu care Skulkiu amplasează obstacolele pe coridor, cele patru obstacole de pe acelaşi coridor au o adâncime din ce în ce mai mare, ceea ce implică faptul că pentru a distruge al doilea obstacol amplasat pe coridor este nevoie de 5 ori mai multă energie decât cea obișnuită, pentru a distruge cel de-al treilea obstacol amplasat pe coridor este nevoie de 25 ori mai multă energie decât cea obișnuită, iar pentru a distruge al patrulea obstacol amplasat pe acelaşi coridor este nevoie de 125 de ori mai multă energie decât cea obișnuită. Indiferent de sensul de parcurgere al coridorului de către Zane pentru a înlătura obstacolele, energia consumată este aceeaşi, aceasta depinzând doar de ordinea în care au fost amplasate obstacolele de către Skulkiu. Zane nu va înlătura obstacolele de pe toate coridoarele ci doar strictul necesar pentru a avea acces la fiecare păpușă. Zane dorește să-i lase pe ceilalți ninja să se antreneze așa că face în așa fel încât ajutorul pentru distrugerea obstacolelor de tip E să fie minim și apoi ca el să utilizeze un număr minim de unități de energie. Pentru coridoarele pe care se află obstacole de tip E Zane consumă energie doar pentru obstacolele de tip A, B, C şi D. Inițial Zane se află lângă păpușa 1.

Cerințe

1. Precizați la câte dintre cele n păpuși poate ajunge Zane înainte de a cere ajutorul celorlalți ninja.

2. Precizați pentru eliberarea câtor coridoare trebuie să ceară ajutor extern pentru a reuși să ajungă la toate cele n păpuși și câte obstacole de tip E sunt în total pe aceste coridoare.

3. Precizați care este numărul minim de unități de energie utilizate.

Date de intrare

Fișierul de intrare ninjago.in conține pe prima linie v care poate avea doar valorile 1, 2 sau 3 reprezentând cerința care va fi rezolvată. Fișierul ninjago.in conține pe a doua linie numerele naturale n și m separate printr-un spațiu, iar pe următoarele m linii pentru fiecare coridor două numere naturale separate printr-un spațiu reprezentând cele două păpuși între care se circulă pe coridorul respectiv urmate de un spaţiu și patru litere corespunzătoare celor patru tipuri de obstacole în ordinea în care Skulkiu le-a amplasat pe coridorul respectiv. Între cele patru litere nu se află nici un spaţiu.

Date de ieșire

Dacă valoarea lui v este 1 atunci fișierul de ieșire ninjago.out va conține pe prima linie numai numărul păpușilor la care poate ajunge Zane înainte de a cere ajutorul celorlalți ninja.

Dacă valoarea lui v este 2 atunci fișierul de ieșire ninjago.out va conține pe prima linie numărul de coridoare pe care nu le poate elibera singur și pe a doua linie numărul total de obstacole de tip E de pe aceste coridoare.

Dacă valoarea lui v este 3 atunci fișierul de ieșire ninjago.out va conține pe prima linie numai numărul minim de unități de energie utilizate.

Restricții și precizări

• 1 ≤ N,M ≤ 31200;

• Pentru rezolvarea corectă a primei cerințe se acordă 20 de puncte;

• Pentru rezolvarea corectă a celei de-a doua cerințe se acordă 40 de puncte;

• Pentru rezolvarea corectă a celei de-a treia cerințe se acordă 30 de puncte.

• În concurs s-au acordat 10 puncte din oficiu. Pe site se acordă 10 puncte pentru exemple.

Rezolvare

#include <fstream>

#include <vector>

#include <algorithm>

using namespace std;

ifstream fin ("ninjago.in");

ofstream fout("ninjago.out");

const int Max = 31201;

int v, n, m;

vector <pair <int, pair <int, int>>> graf[Max];

vector <int> tata(Max, -1), dim(Max, 1);

struct muchii

{

    int x, y, c, e;

} a[Max];

int cost(string s, int &c)

{

    vector <int> aux = {1, 5, 25, 125};

    int sum = 0;

    for (int i = 0; i < 4; ++i)

    {

        if (s[i] != 'E')

            sum += (s[i] - 64) \* aux[i];

        else

            ++c;

    }

    return sum;

}

void read1()

{

    int x, y, z, c;

    string s;

    fin >> n >> m;

    for (int i = 1; i <= m; ++i)

    {

        fin >> x >> y >> s;

        c = 0;

        z = cost(s, c);

        graf[x].push\_back({y, {z, c}});

        graf[y].push\_back({x, {z, c}});

    }

}

void read2()

{

    string s;

    fin >> n >> m;

    for (int i = 1; i <= m; ++i)

    {

        fin >> a[i].x >> a[i].y >> s;

        a[i].e = 0;

        a[i].c = cost(s, a[i].e);

    }

}

void dfs(int nod, vector <bool> &viz, int &c)

{

    viz[nod] = true;

    ++c;

    for (auto i : graf[nod])

        if (!viz[i.first] && i.second.second == 0)

            dfs(i.first, viz, c);

}

int find\_set(int nod)

{

    if (tata[nod] == -1)

        return nod;

    return tata[nod] = find\_set(tata[nod]);

}

void union\_sets(int nod1, int nod2)

{

    nod1 = find\_set(nod1);

    nod2 = find\_set(nod2);

    if (nod1 == nod2)

        return;

    if (dim[nod1] < dim[nod2])

        swap(nod1, nod2);

    dim[nod1] += dim[nod2];

    tata[nod2] = nod1;

}

bool comp(muchii a, muchii b)

{

    if (a.e < b.e)

        return true;

    else if (a.e == b.e)

        return a.c < b.c;

    return false;

}

void kruskal()

{

    sort (a + 1, a + 1 + m, comp);

    int cost = 0, nr = 0, nr\_e = 0;

    for (int i = 1; i <= m; ++i)

    {

        if (find\_set(a[i].x) != find\_set(a[i].y))

        {

            union\_sets(a[i].x, a[i].y);

            cost += a[i].c;

            if (a[i].e)

            {

                ++nr;

                nr\_e += a[i].e;

            }

        }

    }

    if (v == 2)

        fout << nr << "\n" << nr\_e;

    else

        fout << cost;

}

void solve1()

{

    read1();

    vector <bool> viz(n + 1, false);

    int c = 0;

    dfs(1, viz, c);

    fout << c;

}

void solve2()

{

    read2();

    kruskal();

}

int main()

{

    fin >> v;

    if (v == 1)

        solve1();

    else

        solve2();

    fin.close();

    fout.close();

    return 0;

}

**Conexidad**

Fie un graf neorientat cu N noduri și M muchii, care NU este conex.

Cerința

Să i se adauge grafului un număr minim de muchii, astfel încât acesta să devină conex. Fie extrai numărul de muchii nou-adăugate care sunt incidente cu nodul i, iar max\_extra cea mai mare dintre valorile extra1, extra2,… , extraN. Mulțimea de muchii adăugate trebuie să respecte condiția ca valoarea max\_extra să fie minimă.

Date de intrare

Pe prima linie a fișierului de intrare conexidad.in se află două numere naturale N și M, iar pe fiecare dintre următoarele M linii se află câte o pereche de numere a, b, semnificând faptul că există muchia [a,b]. Numerele aflate pe aceeași linie a fișierului sunt separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire conexidad.out va conține pe prima linie valoarea max\_extra. Pe a doua linie va conține valoarea K reprezentând numărul de muchii nou-adăugate în graf. Fiecare dintre următoarele K linii va conține câte o pereche de numere c, d, separate prin câte un spațiu, semnificând faptul că se adaugă grafului muchia [c,d].

Restricții și precizări

• 1 ≤ N ≤ 100

• 0 ≤ M ≤ N\*(N-1)/2

• Nodurile grafului sunt numerotate de la 1 la N inclusiv.

• Muchiile prezente în fișierul de intrare sunt distincte.

• Pentru orice muchie [a,b] aflată în fișierul de intrare, avem a ≠ b.

• Graful din fișierul de intrare nu este conex.

• În cazul în care soluția afișată pentru un anumit test conectează graful cu număr minim de muchii, dar nu minimizează valoarea lui max\_extra, se vor acorda 50% din punctajul pentru testul respectiv.

• Dacă există mai multe soluții optime, se va admite oricare dintre acestea.

Rezolvare

#include <fstream>

#include <vector>

#include <queue>

#include <bitset>

using namespace std;

ifstream fin ("conexidad.in");

ofstream fout("conexidad.out");

const int Max = 1e2 + 1;

int n, m;

vector <int> graf[Max];

bitset <Max> v;

priority\_queue <pair <int, int>, vector <pair <int, int>>, greater <pair <int, int>>> q;

void read()

{

    int x, y;

    fin >> n >> m;

    for (int i = 1; i <= m; ++i)

    {

        fin >> x >> y;

        graf[x].push\_back(y);

        graf[y].push\_back(x);

    }

}

void dfs(int nod)

{

    v[nod] = 1;

    for (int i : graf[nod])

        if (!v[i])

        {

            q.push({0, i});

            dfs(i);

        }

}

void solve()

{

    int extra = 0;

    vector <pair <int, int>> ans;

    q.push({0, 1});

    dfs(1);

    for (int i = 2; i <= n; ++i)

        if (!v[i])

        {

            int nr = q.top().first + 1;

            int nod = q.top().second;

            extra = max(extra, nr);

            q.pop();

            q.push({nr, nod});

            q.push({1, i});

            ans.push\_back({nod, i});

            dfs(i);

        }

    fout << extra << "\n" << ans.size() << "\n";

    for (int i = 0; i < ans.size(); ++i)

        fout << ans[i].first << " " << ans[i].second << "\n";

}

int main()

{

    read();

    solve();

    fin.close();

    fout.close();

    return 0;

}

**Galeti**

Avem n găleți numerotate de la stânga la dreapta numere de la 1 la n. Fiecare găleată conține inițial 1 litru de apă. Capacitatea fiecărei găleți se consideră nelimitată. Vărsăm gălețile una în alta, respectând o anumită regulă, până când toată apa ajunge în prima găleată din stânga. Vărsarea unei găleți presupune un anumit efort.

Regula după care se răstoarnă gălețile este următoarea: se aleg două găleți astfel încât orice găleată situată între ele să fie goală. Se varsă apa din găleata din dreapta în găleata din stânga. Efortul depus este egal cu volumul de apă din găleata din dreapta (cea care se varsă). Formal, dacă notăm cu ai volumul de apă conținut în găleata cu numrăul i, regula de vărsare a acestei găleți în găleata cu numărul j poate fi descrisă astfel:

1. j < i;

2. ak=0,∀ k a.î. j < k < i (dacă j + 1 = k, atunci această regulă nu va putea exista);

3. efortul depus este ai;

4. după vărsare aj=aj+ai și ai=0.

Cerința

Cunoscând numărul de găleți n și un număr natural e, să se determine o succesiune de vărsări în urma căreia toată apa ajunge în găleata cea mai din stânga și efortul total depus este exact e.

Date de intrare

Fișierul de intrare galeti.in conține pe prima linie două numere naturale, n și e, în această ordine, separate printr-un spațiu. Primul număr, n, reprezintă numărul de găleți. Al doilea număr, e, reprezintă efortul care trebuie depus pentru a vărsa toată apa în găleata din stânga.

Date de ieșire

Fișierul de ieșire galeti.out trebuie să conțină n-1 linii care descriu vărsările, în ordinea în care acestea se efectuează, pentru a vărsa toată apa în găleata din stânga cu efortul total e. Fiecare dintre aceste linii trebuie să conțină două numere, i și j, separate printr-un spațiu, cu semnificația că apa din găleata cu numărul i se varsă în găleata cu numărul j.

Restricții și precizări

• 1 ≤ n ≤ 100.000

• 1 ≤ e ≤ 5.000.000.000

• Se asigură faptul că există cel puțin o soluție posibilă pentru fiecare test în parte;

• Dacă există mai multe soluții, oricare dintre ele se va considera validă;

• Pentru teste în valoare de 18 puncte, datele de intrare sunt cunoscute. Mai precis:

• Testul 0: n = 91, e = 90.

• Testul 1: n = 30, e = 435.

• Testul 2: n = 7, e = 16.

• Pentru alte teste în valoare de 15 puncte, n ≤ 9.

Rezolvare

#include <fstream>

#define ll long long

using namespace std;

ifstream fin ("galeti.in");

ofstream fout("galeti.out");

ll n, e;

void solve(int st, int dr, ll e)

{

    if (st == dr)

        return;

    else if (e >= 2 \* (dr - st) - 1)

    {

        solve(st + 1, dr, e - dr + st);

        fout << st + 1 << " " << st << "\n";

        return;

    }

    else

    {

        solve(st, dr - 1, e - 1);

        fout << dr << " " << st << "\n";

        return;

    }

}

int main()

{

    fin >> n >> e;

    solve(1, n, e);

    fin.close();

    fout.close();

    return 0;

}

**Braduți**

Robotul Vasile s-a angajat la fabrica decoraţiunilor de Crăciun unicat. El trebuie să monteze beculeţe colorate în brăduţi, astfel încât oricare doi brăduţi să fie diferiţi. Pe o bandă de asamblare robotul Vasile are la dispoziţie N beculeţe colorate b1, b2, …, bN, astfel încât oricare două beculeţe sunt colorate diferit. În vârful bradului va pune o steluţă, iar pentru montarea beculeţelor în brăduţ el construieşte lanţuri de becuri în modul următor:

– pentru primul lanţ selectează 3 beculeţe bi1, bi2, bi3, în ordinea în care acestea sunt plasate pe bandă (i1 < i2 < i3) şi montează în bi2 pe tulpina bradului, bi1 în stânga, iar b3i în dreapta;

– pentru al doilea lanţ selectează apoi 5 beculeţe bj1, bj2, bj3, bj4, bj5, în ordinea de pe bandă (j1 < j2 < j3 < j4 < j5) şi montează bj3 pe tulpina bradului imediat sub bi2, în stânga bj1 şi bj2 (în această ordine), iar în dreapta bj4 şi bj5 (în această ordine);

– continuă în acelaşi mod, construind la fiecare pas un lanţ în care se selectează două becuri în plus faţă de lanţul precedent, respectând ordinea în care se află becurile pe banda de asamblare; becul situat la mijlocul lanţului este montat pe tulpină, imediat sub becul precedent, celelalte becuri fiind plasate în ordine în partea stângă, respectiv în partea dreaptă;

– procedeul se repetă până când numărul de becuri rămase pe banda de asamblare este insuficient pentru construirea unui nou lanţ.

Înălţimea bradului este considerată egală cu numărul de lanţuri construite (egal cu numărul de beculeţe montate pe tulpină). Doi brazi sunt diferiţi dacă există cel puţin o poziţie în care în cei doi brazi se află beculeţe de culori diferite.

Cerința

Cunoscând numărul N de beculeţe aflate pe banda de asamblare, scrieţi un program care să rezolve următoarele două cerinţe:

1. determină înălţimea bradului (numărul de lanţuri ce pot fi construite cu cele N beculeţe);

2. determină numărul de brazi diferiţi ce pot fi construiţi cu cele N beculeţe.

Date de intrare

Fișierul de intrare braduti.in conţine pe prima linie numărul C care reprezintă cerinţa care trebuie să fie rezolvată (1 sau 2). Pe a doua linie se află numărul natural N care reprezintă numărul de beculeţe colorate de pe linia de asamblare.

Date de ieșire

Fișierul de ieșire braduti.out va conţine o singură linie pe care va fi scris un număr natural reprezentând rezultatul determinat pentru cerinţa C.

Restricții și precizări

• 3 ≤ N ≤ 5555

• Pentru teste valorând 10 puncte C = 1

• Pentru teste valorând 20 de puncte, C = 2 şi rezultatul va avea cel mult 18 cifre

• Pentru teste valorând 60 de puncte, C = 2 şi rezultatul va avea cel mult 10.000 cifre

Rezolvare

#include <fstream>

#include <vector>

using namespace std;

ifstream fin ("braduti.in");

ofstream fout("braduti.out");

const int Max = 50000;

int n, c, h, l;

void inaltime()

{

    while (l + 2 \* h + 3 <= n)

    {

        l += 2 \* h + 3;

        ++h;

    }

    l = n - l;

}

void inmultire(vector <int> &a, int x)

{

    int t = 0;

    for (int i = 1 ; i <= a[0] ; ++i)

    {

        a[i] = a[i] \* x + t;

        t = a[i] / 10;

        a[i] %= 10;

    }

    while (t)

    {

        a[++a[0]] = t % 10;

        t /= 10;

    }

}

void impartire(vector <int> &a, int x)

{

    unsigned int r = 0;

    for (int i = a[0] ; i>= 1 ; --i)

    {

        r  = r \* 10 + a[i];

        a[i] = r / x;

        r %= x;

    }

    while (!a[a[0]] && a[0] > 1)

        --a[0];

}

void solve()

{

    vector <int> a(Max);

    a[0] = a[1] = 1;

    for (int i = 2 \* h + 2 ; i <= n ; ++i)

        inmultire(a, i);

    for (int i = 2 ; i <= l ; ++i)

        impartire(a, i);

    for (int i = 1 ; i < h ; ++i)

        for (int j = 2 ; j <= 2 \* i + 1 ; ++j)

            impartire(a, j);

    for (int i = a[0] ; i >= 1 ; --i)

        fout << a[i];

    a.clear();

}

int main()

{

    fin >> c >> n;

    inaltime();

    if (c == 1)

        fout << h;

    else

        solve();

    fin.close();

    fout.close();

    return 0;

}

**Parcare2**

În cel mai recent eveniment al companiei Tesla, Paul Musk a anunțat un nou produs inovativ: parcarea autonomă. Fiind cunoscut pentru lansările produselor incomplete, nici parcarea nu este completă, fiind nevoie de o automatizare pentru a atribui câte un loc mașinilor care vor să folosească parcarea. Parcarea este formată din N locuri, numerotate de la 1 la N, și este deschisă timp de T secunde, începând cu secunda 1. Pe parcursul zilei, sosesc M mașini care vor să folosească parcarea, pentru fiecare dintre acestea știindu-se timpul de sosire s[i] și timpul de plecare p[i]. Mașinile vin în ordinea timpului de sosire s[i] și ocupă locul de parcare în intervalul de timp [ s[i], p[i] ]. Pentru fiecare dintre acestea, trebuie să afișați un loc liber de parcare (dacă sunt mai multe, se poate afișa oricare) în care aceasta se poate așeza sau -1 dacă parcarea este plină în momentul venirii mașinii. Dacă o mașină nu are loc în parcare la timpul de sosire, aceasta nu va mai intra în parcare la niciun timp viitor.

Cerința

La final, Paul este interesat de mașinile care mai sunt rămase în parcare la închiderea parcării, de aceea vă cere să afișați configurația parcării la timpul T.

Date de intrare

Pe prima linie a fișierului de intrare parcare.in se găsesc trei numere întregi N, M și T, reprezentând numărul de locuri din parcare, numărul de mașini care vin să folosească parcarea, respectiv numărul de secunde pentru care este deschisă parcarea. Următoarele M linii conțin fiecare câte două numere întregi s[i], p[i], reprezentând venirea unei mașini la secunda s\_i care va pleca la secunda p\_i. Mașinile apar în fișierul de intrare în ordine crescătoare după timpul de sosire s\_i.

Date de ieșire

În fișierul de ieșire parcare.out se vor afișa M + 1 linii în total, primele M linii conținând fiecare câte un număr întreg între 1 și N reprezentând locul de parcare pe care îl va ocupa mașina, sau -1 dacă nu există niciun loc de parcare disponibil.

Ultima linie va conține N numere întregi, reprezentând configurația parcării la închidere, unde cel de-al i-lea număr reprezintă timpul de sosire al mașinii de pe locul de parcare i, sau -1 dacă locul de parcare i este gol.

Restricții și precizări

• 1 ≤ N, M, T ≤ 200.000

• 1 ≤ s[i] ≤ T

• 1 ≤ s[i] < p[i] ≤ 200.000 • Considerând următoarele 2M valori: s[1], s[2], ..., s[M], p[1], p[2], ..., p[M], acestea sunt distincte două câte două.

• Dacă există mai multe soluții, se poate afișa oricare dintre acestea.

Rezolvare

#include <fstream>

#include <vector>

using namespace std;

ifstream fin ("parcare.in");

ofstream fout("parcare.out");

const int Max = 2e5 + 1;

int n, m, t;

vector <int> s(Max), p(Max);

void read()

{

    fin >> n >> m >> t;

    for (int i = 1; i <= m; ++i)

        fin >> s[i] >> p[i];

}

void park(int i, vector <int> &l)

{

    for (int j = 1; j <= n; ++j)

        if (p[l[j]] <= s[i])

        {

            l[j] = i;

            fout << j << "\n";

            return;

        }

    fout << -1 << "\n";

}

void solve()

{

    vector <int> l(n + 1), ans(n + 1);

    for (int i = 1; i <= n; ++i)

    {

        l[i] = i;

        fout << i << "\n";

    }

    for (int i = n + 1; i <= m; ++i)

    {

        park(i, l);

        if (s[i] == t)

            ans = l;

    }

    if (ans[1] == 0)

        ans = l;

    for (int i = 1; i <= n; ++i)

        if (p[ans[i]] > t)

            fout << s[ans[i]] << " ";

        else

            fout << -1 << " ";

}

int main()

{

    read();

    solve();

    fin.close();

    fout.close();

    return 0;

}

**Polihroniade**

O matrice pătratică de dimensiuni N \* N cu N par și elemente din mulțimea {0,1} se numește tablă de șah dacă oricare două celule vecine pe o linie sau pe o coloană au valori diferite (cu alte cuvinte, dacă nu există două valori egale alăturate).De ziua ei, Victor i-a cumpărat Elisabetei o astfel de matrice A, care nu este neapărat tablă de șah. Aflând despre pasiunea ei, acesta vrea acum să transforme matricea A într-o tablă de șah. Timpul fiind limitat, el poate efectua doar următoarele tipuri de operații asupra matricei:

1. Interschimbă liniile i și j din A (celelalte linii rămân neschimbate, iar valorile din interiorul liniilor i și j rămân neschimbate și își păstrează ordinea). Operația are sens pentru 1 ≤ i, j ≤ N.

2. Interschimbă coloanele i și j din A (celelalte coloane rămân neschimbate, iar valorile din interiorul coloanelor i și j rămân neschimbate și își păstrează ordinea). Operația are sens pentru 1 ≤ i, j ≤ N.

Cerințe

Dorind s-o impresioneze pe Elisabeta, Victor apelează la voi, programatori renumiți, să-l ajutați în a transforma matricea A într-o tablă de șah. Pentru aceasta, Victor are nevoie de următoarele informații:

1. Poate fi transformată matricea A în tablă de șah?

2. Care este numărul minim de operații necesare pentru a duce la îndeplinire acest scop?

3. Care ar fi o succesiune de operații care transformă matricea A într-o tablă de șah?

În cazul ultimei cerințe, pentru a intra în grațiile lui Victor va trebui ca numărul de operații efectuate să fie minim. Totuși, chiar și un număr neminim de operații va fi răsplătit, însă nu într-atât de mult.

Vi se dau două numere P, T și T matrice A, reprezentând mai multe instanțe ale problemei. Pentru fiecare matrice A dintre cele T va trebui să rezolvați cerința cu numărul P∊{1,2,3} dintre cele listate mai sus.

Date de intrare

Pe prima linie se găsesc două numere întregi pozitive P și T, reprezentând numarul cerinței de rezolvat și, respectiv, numărul de scenarii pentru care va trebui să rezolvați problema. Urmează cele T instanțe ale problemei, fiecare fiind compusă din N + 1 linii: pe prima linie se va afla numarul N, iar pe următoarele N linii câte N cifre binare neseparate prin spații, reprezentând câte o linie a matricei A.

Date de ieșire

Pentru fiecare dintre cele T instanțe se va afișa răspunsul, începând de la o linie nouă, după cum urmează:

1. Dacă P = 1, atunci se va afișa pe o singură linie 1 dacă matricea A poate fi transformată în tablă de șah, și 0 altfel.

2. Dacă P = 2, atunci se va afișa pe o singură linie un întreg reprezentând numărul minim de interschimbări de linii și/sau coloane necesare pentru a transforma matricea A în tablă de șah.

3. Dacă P = 3, atunci se va afișa pe o linie un număr X. Apoi, pe fiecare dintre următoarele X linii se va afișa câte o interschimbare de linii sau coloane, după următorul format: L i j are semnificația că liniile i și j se interschimbă, iar C i j are semnificația că se interschimbă coloanele i și j. Matricea obținută după aplicarea celor X operații asupra lui A în ordinea dată trebuie să fie o tablă de șah.

Restricții și precizări

• 1 ≤ T ≤ 350

• 1 ≤ N ≤ 1 000

• N este par.

• Pentru cerințele de tip P = 2 și P = 3 se garantează că matricea A poate fi transformată în tablă de șah folosind interschimbări de linii și/sau coloane.

• Suma valorilor N pentru cele T scenarii nu depășește 2.000.

• Pentru 40 de puncte, P = 1

• Pentru alte 34 de puncte, P = 2

• Pentru alte 26 de puncte, P = 3

• Dacă există mai multe soluții, oricare este considerată corectă.

• Dacă numărul X de operații folosite nu este minim, atunci se acordă 50% din punctajul pe test.

Rezolvare

#include <fstream>

#include <vector>

using namespace std;

ifstream cin ("f.in");

ofstream cout("f.out");

bool verify\_com (string x, string y, int n)

{

    for (int i = 0; i < n; ++i)

        if (x[i] == y[i])

            return false;

    return true;

}

bool solve1 (int n)

{

    vector <string> a(n + 1);

    int k0, k1;

    for (int i = 1; i <= n; ++i)

        cin >> a[i];

    for (int i = 1; i <= n; ++i)

    {

        k0 = k1 = 0;

        for (int j = 0; j < n; ++j)

        {

            if (a[i][j] == '1')

                ++k1;

            if (a[i][j] == '0')

                ++k0;

        }

        if (k0 != k1)

            return 0;

    }

    int com = 0;

    for (int i = 2; i <= n; ++i)

        if (verify\_com(a[1], a[i], n))

        {

            com = i;

            break;

        }

    if (!com)

        return 0;

    k0 = 0;

    k1 = 1;

    for (int i = 2; i <= n; ++i)

    {

        if (a[1] == a[i])

            ++k1;

        if (a[com] == a[i])

            ++k0;

    }

    if (k0 != k1)

        return 0;

    else if (k0 \* 2 == n)

        return 1;

    return 0;

}

void solve2 (int n)

{

    vector <string> a(n + 1);

    vector <pair <char, int>> x01, x10;

    vector <int> y01, y10;

    int aux = 1;

    for (int i = 1; i <= n; ++i)

        cin >> a[i];

    for (int i = 1; i <= n; i += 2)

        if (a[i][0] == '1')

            x01.push\_back({'L', i});

    for (int i = 2; i <= n; i += 2)

        if (a[i][0] == '0')

            y01.push\_back(i);

    if (!x01.empty() && x01[0].second == 1)

        aux = y01[0];

    for (int j = 1; j <= n; j += 2)

        if (a[aux][j - 1] == '1')

            x01.push\_back({'C', j});

    for (int j = 2; j <= n; j += 2)

        if (a[aux][j - 1] == '0')

            y01.push\_back(j);

    for (int i = 1; i <= n; i += 2)

        if (a[i][0] == '0')

            x10.push\_back({'L', i});

    for (int i = 2; i <= n; i += 2)

        if (a[i][0] == '1')

            y10.push\_back(i);

    aux = 1;

    if (!x10.empty() && x10[0].second == 1)

        aux = y10[0];

    for (int j = 1; j <= n; j += 2)

        if (a[aux][j - 1] == '0')

            x10.push\_back({'C', j});

    for (int j = 2; j <= n; j += 2)

        if (a[aux][j - 1] == '1')

            y10.push\_back(j);

    cout << min(x01.size(), x10.size()) << "\n";

}

void solve3 (int n)

{

    vector <string> a(n + 1);

    vector <pair <char, int>> x01, x10;

    vector <int> y01, y10;

    int aux = 1;

    for (int i = 1; i <= n; ++i)

        cin >> a[i];

    for (int i = 1; i <= n; i += 2)

        if (a[i][0] == '1')

            x01.push\_back({'L', i});

    for (int i = 2; i <= n; i += 2)

        if (a[i][0] == '0')

            y01.push\_back(i);

    if (!x01.empty() && x01[0].second == 1)

        aux = y01[0];

    for (int j = 1; j <= n; j += 2)

        if (a[aux][j - 1] == '1')

            x01.push\_back({'C', j});

    for (int j = 2; j <= n; j += 2)

        if (a[aux][j - 1] == '0')

            y01.push\_back(j);

    for (int i = 1; i <= n; i += 2)

        if (a[i][0] == '0')

            x10.push\_back({'L', i});

    for (int i = 2; i <= n; i += 2)

        if (a[i][0] == '1')

            y10.push\_back(i);

    aux = 1;

    if (!x10.empty() && x10[0].second == 1)

        aux = y10[0];

    for (int j = 1; j <= n; j += 2)

        if (a[aux][j - 1] == '0')

            x10.push\_back({'C', j});

    for (int j = 2; j <= n; j += 2)

        if (a[aux][j - 1] == '1')

            y10.push\_back(j);

    if (x01.size() < x10.size())

    {

        cout << x01.size() << "\n";

        for (auto i = 0; i < x01.size(); ++i)

            if (x01[i].second < y01[i])

                cout << x01[i].first << " " << x01[i].second << " " << y01[i] << "\n";

            else

                cout << x01[i].first << " " << y01[i] << " " << x01[i].second << "\n";

    }

    else

    {

        cout << x10.size() << "\n";

        for (auto i = 0; i < x10.size(); ++i)

            if (x10[i].second < y10[i])

                cout << x10[i].first << " " << x10[i].second << " " << y10[i] << "\n";

            else

                cout << x10[i].first << " " << y10[i] << " " << x10[i].second << "\n";

    }

}

int main()

{

    int P, T;

    cin >> P >> T;

    if (P == 1)

    {

        int n;

        for (int I = 1; I <= T; ++I)

        {

            cin >> n;

            cout << solve1(n) << "\n";

        }

    }

    else if (P == 2)

    {

        int n;

        for (int I = 1; I <= T; ++I)

        {

            cin >> n;

            solve2(n);

        }

    }

    else

    {

        int n;

        for (int I = 1; I <= T; ++I)

        {

            cin >> n;

            solve3(n);

        }

    }

    return 0;

}

**4. Site realizat în HTML**

Probleme anterioare pot fi găsite și pe acest site: <https://tonyy-y.github.io/Proiect/>

**5. Bibliografie**

[1] <https://www.pbinfo.ro/>

[2] <https://kilonova.ro/>

[3] <https://www.geeksforgeeks.org/>

[4] <https://en.cppreference.com/w/>

[5] <https://github.com/Giulian617/Hai-la-olimpiada-2023-2024>