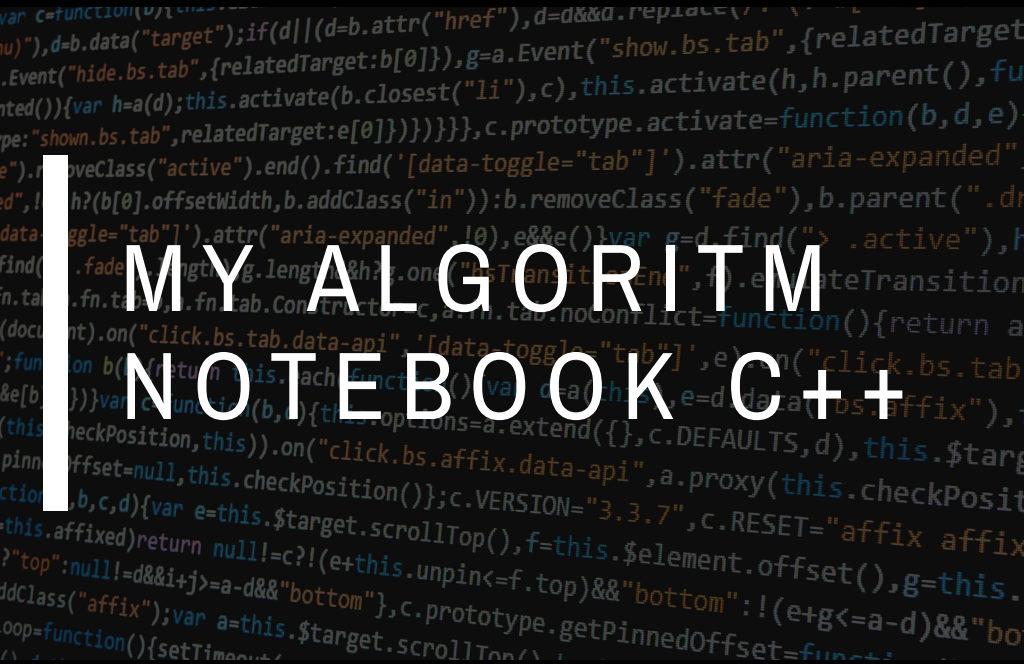
GRAPH THEORY 1 ST YEAR SEMESTER 2



MY ALGORITM NOTEBOOK C++

EAT.SLEEP.CODE.LOVE.REPEAT

AUTHOR: ANUJIN BAATARTSOGT

Table of Contents

Задача A. От списка ребер к матрице смежности, ориентированный граф (1 балл) (!)

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Простой ориентированный граф задан списком ребер, выведите его представление в виде матрицы смежности.

Формат входного файла

Входной файл содержит числа n ($1 \le n \le 100$) — число вершин в графе и m ($1 \le m \le n(n-1)$) — число ребер. Затем следует m пар чисел — ребра графа.

Формат выходного файла

Выведите в выходной файл матрицу смежности заданного графа.

Пример

input.txt	output.txt
3 4 1 2 2 3 3 1 3 2	0 1 0 0 0 1 1 1 0

```cpp
#include<bits/stdc++.h>
using namespace std;

int main()
{

 ifstream fin("input.txt");
 ofstream fout("output.txt");

 int n, m;
 fin >> n >> m;
 int arr[n][n];

 for(int i = 0; i < n; i++)
 {
  for(int j = 0; j < n; j++)
  {
   arr[i][j] = 0;
  }
 }

 for(int i = 0; i < m; i++)
 {
  int v1, v2;
  fin >> v1 >> v2;
  arr[v1-1][v2-1] = 1;
 }

 for(int i = 0; i < n; i++)
 {
  for(int j = 0; j < n; j++)
  {
   fout << arr[i][j] << " ";
  }
  fout << "\n";
 }

 fin.close();
 fout.close();
 }
```

Задача B. Проверка на неориенитрованность (1 балл)

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

По заданной квадратной матрице $n \times n$ из нулей и единиц определите, может ли данная матрица быть матрицей смежности простого неориентированного графа.

Формат входного файла

Входной файл содержит число n ($1 \le n \le 100$) — размер матрицы, и затем n строк по n чисел, каждое из которых равно 0 или 1 — саму матрицу.

Формат выходного файла

Выведите в выходной файл «`YES`» если приведенная матрица может быть матрицей смежности простого неориентированного графа и «`NO`» в противном случае.

Пример

input.txt	output.txt
3 0 1 1 1 0 1 1 1 0	YES
3 0 1 0 1 0 1 1 1 0	NO
3 0 1 0 1 1 1 0 1 0	NO

```cpp
#include<bits/stdc++.h>
using namespace std;

int main()
{
 ifstream fin("input.txt");
 ofstream fout("output.txt");

 int n;
 fin >> n;
 vector<vector<int>> arr;
 arr.resize(n);

 for(int i = 0; i < n; i++)
 {
  arr[i].resize(n);
 }


 for(int i = 0; i < n; i++)
 {
  for(int j = 0; j < n; j++)
  {
   fin >> arr[i][j];
  }
 }

 for(int i = 0; i < n; i++)
 {
  for(int j = 0; j < n; j++)
  {
   if((arr[i][j] != arr[j][i]) || (arr[j][j] == 1))
   {
    fout << "NO" << "\n";
    fin.close();
    fout.close();
   }

  }
 }

 fout << "YES" << "\n";
 fin.close();
 fout.close();
}
```

Задача C. Проверка на наличие параллельных ребер, неориентированный граф (1 балл)

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Неориентированный граф задан списком ребер. Проверьте, содержит ли он параллельные ребра.

Формат входного файла

Входной файл содержит числа n ($1 \le n \le 100$) — число вершин в графе и m ($1 \le m \le 10\,000$) — число ребер. Затем следует m пар чисел — ребра графа.

Формат выходного файла

Выведите в выходной файл «`YES`» если граф содержит параллельные ребра и «`NO`» в противном случае.

Пример

input.txt	output.txt
3 3 1 2 2 3 1 3	NO
3 3 1 2 2 3 2 1	YES

```cpp
#include<bits/stdc++.h>
using namespace std;

int main()
{
 ifstream fin("input.txt");
 ofstream fout("output.txt");
 int n, m, x, y;
 int arr[n][n];
 fin >> n >> m;

 for(int i = 0; i < m; i++)
 {
  fin >> x >> y;
  arr[x--][y--]++;
 }

 for(int i = 0; i < n; i++)
 {
  for(int j = 0; j < n; j++)
  {
   if(arr[i][j] + arr[j][i] > 1)
   {
    fout << "YES" << "\n";
    fin.close();
    fout.close();
   }
  }
 }

 fout << "NO" << "\n";
 return 0;
}
```

Задача D. Компоненты связности (1 балл)

Имя входного файла:	`components.in`
Имя выходного файла:	`components.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Дан неориентированный граф. Требуется выделить компоненты связности в нем. Подсказка: для решения задачи можно воспользоваться поиском в ширину или поиском в глубину.

Формат входного файла

Первая строка входного файла содержит два натуральных числа n и m — количество вершин и ребер графа соответственно ($1 \le n \le 100\,000$, $0 \le m \le 200\,000$).

Следующие m строк содержат описание ребер по одному на строке. Ребро номер i описывается двумя натуральными числами b_i, e_i — номерами концов ребра ($1 \le b_i, e_i \le n$). Допускаются петли и параллельные ребра.

Формат выходного файла

В первой строке выходного файла выведите целое число k — количество компонент связности графа. Во второй строке выведите n натуральных чисел $a_1, a_1, \ldots, a_n$, не превосходящих k, где a_i — номер компоненты связности, которой принадлежит i-я вершина.

Пример

components.in	components.out
3 1 1 2	2 1 1 2
4 2 1 3 2 4	2 2 1 2 1

```cpp
#include<bits/stdc++.h>
using namespace std;

const int SIZE = 1e5 + 1;
vector<int> adj[SIZE];
int cmp[SIZE];
int vis[SIZE];


void dfs(int v)
{
 vis[v] = true;
 for(int i = 0; i < adj[v].size(); i++)
 {
  int next_v = adj[v][i];
  // go to all adjacent vertices and give a component id
  // if all already visit, go to next vertice
  if(!vis[next_v])
  {
   cmp[next_v] = cmp[v];
   dfs(next_v);
  }
 }
}


int main()
{
 ifstream fin("components.in");
 ofstream fout("components.out");

 int n, m;
 int a, b;


 fin >> n >> m;

 for(int i = 0; i < m; i++)
 {
  fin >> a >> b;
  adj[a-1].push_back(b-1);
  adj[b-1].push_back(a-1);
 }

 //go to vertices that are not visited
 // give them cmp_id;
 //go to adjacent vertice
 //adj vertive not visited --> cmp_id
 //adj vertice visited --> next vertice --> next cmp_id;

 int cmp_num = 0;
 for(int i = 0; i < n; i++)
 {
  if(!vis[i])
  {
   cmp_num++;
   cmp[i] = cmp_num;
   dfs(i);
  }
 }


 fout << cmp_num << endl;

 for(int i = 0; i < n; i++)
 {
  fout << cmp[i] << ' ';
```

```
 }
 return 0;
}
```

Задача E. Кратчайший путь в невзвешенном графе (1 балл)

Имя входного файла:	`pathbge1.in`
Имя выходного файла:	`pathbge1.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Дан неориентированный невзвешенный граф. Найдите кратчайшее расстояние от первой вершины до всех вершин.

Формат входного файла

В первой строке входного файла два числа: n и m ($2 \le n \le 30000, 1 \le m \le 400000$), где n — количество вершин графа, а m — количество ребер.

Следующие m строк содержат описание ребер. Каждое ребро задается стартовой вершиной и конечной вершиной. Вершины нумеруются с единицы.

Формат выходного файла

Выведите n чисел — для каждой вершины кратчайшее расстояние до нее.

Пример

pathbge1.in	pathbge1.out
2 1 2 1	0 1

```
#include<bits/stdc++.h>

int main()
{

 std::ifstream fin("pathbge1.in");
 std::ofstream fout("pathbge1.out");

 int n, m;
 fin >> n >> m;

 std::vector<int> adj[n];
 for(int i = 0; i < m; i++)
 {
  int a, b;
  fin >> a >> b;
  adj[a-1].push_back(b-1);
  adj[b-1].push_back(a-1);
 }

 std::queue<int> q;
 q.push(0);
 std::vector<int> d(n, INT32_MAX);
 d[0] = 0;

 while(!q.empty())
 {
  int v = q.front();
  q.pop();
  for(auto u: adj[v])
  {
   if(d[u] == INT32_MAX)
   {
    d[u] = d[v] + 1;
    q.push(u);
   }

  }
 }

 for(auto u: d)
 {
  fout << u << ' ';
 }

}
```

Задача F. Лабиринт (2 балла)

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Лабиринт представляет собой поле $n \times m$. По некоторым его клеткам ходить можно, а по некоторым — нет. Узник находится в одной из клеток лабиринта и может перемещаться за ход на одну из четырех соседних клеток. Помогите ему дойти до выхода за минимальное число шагов или сообщите, что выйти невозможно.

Формат входного файла

Во входном файле записаны два числа n и m ($0 < n, m < 100$). Далее n строк по m символов описывают лабиринт. Клетка, по которой можно ходить, обозначена символом "`.`", клетка, по которой нельзя ходить, обозначена символом "`#`". Клетки, обозначенные символами `S` и `T`, задают начальную и конечную клетки соответственно.

Формат выходного файла

Если узник может дойти до выхода, выведите в выходной файл минимальное количество действий и далее последовательность команд — символов `U, D, R` и `L`, показывающих, в какую сторону нужно идти. Если выйти невозможно, выведите `-1`.

Пример

input.txt	output.txt
5 4 .S.. ###. T... .##.	7 RRDDLLL

```
#include <bits/stdc++.h>

int grid[100][100];

class Point
{
 public:
  int y;
  int x;
  int d;
    Point(int x, int y, int w) : y(x), x(y), d(w)
    {
 }
};

int main() {
    std::ifstream fin("input.txt");
    std::ofstream fout("output.txt");

    int m,n;
    fin >> n >> m;
    char c;
    Point start(0,0,0), end(0,0,0);

    fin.get(c);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            fin.get(c);
            switch (c) {
                case '.':
                    grid[i][j] = INT32_MAX;
                    break;
                case '#':
                    grid[i][j] = -1;
                    break;
                case 'S':
                    grid[i][j] = 0;
                    start.x = j;
                    start.y = i;
                    start.d = 0;
                    break;
                case 'T':
                    grid[i][j] = INT32_MAX;
                    end.x = j;
                    end.y = i;
                    break;

                default: {

                }
            }
        }
        fin.get(c);
    }

    std::queue<Point> q;
    q.push(start);

    while (!q.empty()) {
        Point point = q.front();
        q.pop();

        if (point.x == end.x && point.y == end.y) {
            break;
        }
```

```
        if (point.x + 1 < m && grid[point.y][point.x + 1] == INT32_MAX) {
            grid[point.y][point.x + 1] = point.d + 1;
            Point p(point.y, point.x + 1, point.d + 1);
            q.push(p);
        }
        if (point.x - 1 >= 0 && grid[point.y][point.x - 1] == INT32_MAX) {
            grid[point.y][point.x - 1] = point.d + 1;
            Point p(point.y, point.x - 1, point.d + 1);
            q.push(p);
        }
        if (point.y + 1 < n && grid[point.y + 1][point.x] == INT32_MAX) {
            grid[point.y + 1][point.x] = point.d + 1;
            Point p(point.y + 1, point.x, point.d + 1);
            q.push(p);
        }
        if (point.y - 1 >= 0 && grid[point.y - 1][point.x] == INT32_MAX) {
            grid[point.y - 1][point.x] = point.d + 1;
            Point p(point.y - 1, point.x, point.d + 1);
            q.push(p);
        }
    }

    if (grid[end.y][end.x] == INT32_MAX) {
        fout << -1 << std::endl;
        return 0;
    }

    fout << grid[end.y][end.x] << std::endl;

    std::string s;
    int x = end.x, y = end.y;
    while (x != start.x || y != start.y)
{
        if (x - 1 >= 0 && grid[y][x-1] + 1 == grid[y][x]) {
            s = 'R' + s;
            x--;
            continue;
        }
        if (y - 1 >= 0 && grid[y-1][x] + 1 == grid[y][x]) {
            s = 'D' + s;
            y--;
            continue;
        }
        if (y + 1 < n && grid[y+1][x] + 1 == grid[y][x]) {
            s = 'U' + s;
            y++;
            continue;
        }
        if (x + 1 < n && grid[y][x+1] + 1 == grid[y][x]) {
            s = 'L' + s;
            x++;
            continue;
        }
    }

    fout << s;

    return 0;
}
```

Задача A. Топологическая сортировка (1 балл) (!)

Имя входного файла:	`topsort.in`
Имя выходного файла:	`topsort.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Дан ориентированный невзвешенный граф. Необходимо его топологически отсортировать.

Формат входного файла

В первой строке входного файла даны два натуральных числа N и M ($1 \le N \le 100\,000$, $0 \le M \le 100\,000$) — количество вершин и рёбер в графе соответственно. Далее в M строках перечислены рёбра графа. Каждое ребро задаётся парой чисел — номерами начальной и конечной вершин соответственно.

Формат выходного файла

Вывести любую топологическую сортировку графа в виде последовательности номеров вершин. Если граф невозможно топологически отсортировать, вывести -1.

Пример

topsort.in	topsort.out
6 6 1 2 3 2 4 2 2 5 6 5 4 6	4 6 3 1 2 5
3 3 1 2 2 3 3 1	-1

```cpp
#include <bits/stdc++.h>
using namespace std;

vector<vector<int>> adj;
vector<int> visited;
stack<int> s;

bool dfs(int n)
{
 visited[n] = 1;
 for(auto i: adj[n])
 {
  if(visited[i] == 0)
   if(!dfs(i))
   return false;
  if(visited[i] == 1)
   return false;
 }
 visited[n] = 2;
 s.push(n);
 return true;
}

int main()
{
 ifstream fin("topsort.in");
 ofstream fout("topsort.out");
 int v, e;
 fin >> v >> e;

 adj.resize(v);
 visited.resize(v);

 for(int i = 0; i < e; i++)
 {
  int a, b;
  fin >> a >> b;
  adj[--a].push_back(--b);
 }


   for (int i = 0; i < v; i++)
   {
    visited[i] = 0;
 }

   for (int i = 0; i < v; i++)
{
      if (visited[i] == 0)
         if (!dfs(i))
  {
             fout << "-1";
             return 0;
          }
   }

   while(!s.empty())
{
      fout << s.top() + 1 << " ";
      s.pop();
   }

    return 0;
}
```

Задача B. Поиск цикла (2 балла)

Имя входного файла:	`cycle.in`
Имя выходного файла:	`cycle.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Дан ориентированный невзвешенный граф. Необходимо определить есть ли в нём циклы, и если есть, то вывести любой из них.

Формат входного файла

В первой строке входного файла находятся два натуральных числа N и M ($1 \le N \le 100\,000, M \le 100\,000$) — количество вершин и рёбер в графе соответственно. Далее в M строках перечислены рёбра графа. Каждое ребро задаётся парой чисел — номерами начальной и конечной вершин соответственно.

Формат выходного файла

Если в графе нет цикла, то вывести «NO», иначе — «YES» и затем перечислить все вершины в порядке обхода цикла.

Пример

cycle.in	cycle.out
2 2 1 2 2 1	YES 2 1
2 2 1 2 1 2	NO

```
#include<bits/stdc++.h>
using namespace std;

int cycle_start = -1;
int cycle_end;

bool dfs(int v, int visit[], vector<vector<int>> &graph, vector<int> &parent, vector<int> &result)
{
 visit[v] = 1;
 for(int i = 0; i < graph[v].size(); i++)
 {
  int adj = graph[v][i];
  if(visit[adj] == 0)
  {
   parent[adj]  = v;
   dfs(adj, visit, graph, parent, result);
  }
  else if(visit[adj] == 1)
  {
   cycle_end = v;
   cycle_start = adj;
  }
 }

 visit[v] = 2;
 result.push_back(v);
}

int main()
{
 ifstream fin("cycle.in");
 ofstream fout("cycle.out");
 int n, m;
 fin >> n >> m;
 vector<vector<int>> graph;
 graph.resize(n+1);
 int a, b;
 for(int i = 0; i < m; i++)
 {
  fin >> a >> b;
  graph[a].push_back(b);
 }

 vector<int> result;
 int* visit = new int[n+1];

 for(int i= 0; i <= n; i++)
  visit[i] = 0;

 vector<int> parent;
 parent.resize(n+1);
 int j = 1;
 while(j!= graph.size())
 {
  if(visit[j] == 0)
  {
   dfs(j, visit, graph, parent, result);
   j++;
  }
  else
   j++;
 }

 if(cycle_start == -1)
 {
  fout << "NO";
```

```
}
else
{
 fout << "YES" << '\n';
 vector<int> cycle;
 cycle.push_back(cycle_start);
 for(int v = cycle_end; v!= cycle_start; v = parent[v])
  cycle.push_back(v);

 for(int i = cycle.size() - 1; i >= 0; i--)
 {
  fout << cycle[i] << " ";
 }
}


}
```

Задача C. Двудольный граф (1 балл)

Имя входного файла:	`bipartite.in`
Имя выходного файла:	`bipartite.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Двудольным называется неориентированный граф $\langle V, E\rangle$, вершины которого можно разбить на два множества L и R, так что $L \cap R = \emptyset$, $L \cup R = V$ и для любого ребра $(u, v) \in E$ либо $u \in L, v \in R$, либо $v \in L, u \in R$.

Дан неориентированный граф. Требуется проверить, является ли он двудольным.

Формат входного файла

Первая строка входного файла содержит два натуральных числа n и m — количество вершин и ребер графа соответственно ($1 \le n \le 100\,000$, $0 \le m \le 200\,000$).

Следующие m строк содержат описание ребер по одному на строке. Ребро номер i описывается двумя натуральными числами b_i, e_i — номерами концов ребра ($1 \le b_i, e_i \le n$). Допускаются петли и параллельные ребра.

Формат выходного файла

В единственной строке выходного файла выведите «YES», если граф является двудольным и «NO» в противном случае.

Пример

bipartite.in	bipartite.out
4 4 1 2 1 3 2 4 4 2	YES
3 3 1 2 2 3 3 1	NO

```
#include<bits/stdc++.h>
using namespace std;
vector<int> color;
vector<vector <int> > graph;
bool bipartite = true;

void isBipartite(int src)
{
 for(int u = 0; u < graph[src].size(); u++)
 {
  if(color[graph[src][u]] == -1)
  {
   color[graph[src][u]] = 1 - color[src];
   isBipartite(graph[src][u]);
  }
  else if(color[graph[src][u]] == color[src])
  bipartite =  false;
 }
}

int main()
{
 ifstream fin("bipartite.in");
 ofstream fout("bipartite.out");
 int n, m;
 fin >> n >> m;
 graph.resize(n);
 color.assign(n, -1);
 int a, b;

 for(int i = 0; i < m; i++)
 {
  fin >> a >> b;
  graph[a-1].push_back(b-1);
  graph[b-1].push_back(a-1);
 }

 for(int i = 0; i < n; i++)
 {
  if(color[i] == -1)
  {
   color[i] = 1;
   isBipartite(i);
  }
 }

 fout << (bipartite? "YES" : "NO");
 return 0;
}
```

Задача D. Конденсация графа (2 балла)

Имя входного файла:	`cond.in`
Имя выходного файла:	`cond.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Дан ориентированный невзвешенный граф. Необходимо выделить в нем компоненты сильной связности и топологически их отсортировать.

Формат входного файла

В первой строке входного файла находятся два натуральных числа N и M ($1 \le N \le 20\,000, 1 \le M \le 200\,000$) — количество вершин и рёбер в графе соответственно. Далее в M строках перечислены рёбра графа. Каждое ребро задаётся парой чисел — номерами начальной и конечной вершин соответственно.

Формат выходного файла

Первая строка выходного файла должна содержать целое число k — количество компонент сильной связности в графе. Вторая строка выходного файла должна содержать n чисел — для каждой вершины выведите номер компоненты сильной связности, которой она принадлежит. Компоненты должны быть занумерованы таким образом, чтобы для каждого ребра (u, v) номер компоненты, которой принадлежит u не превосходил номер компоненты, которой принадлежит v.

Пример

cond.in	cond.out
6 7 1 2 2 3 3 1 4 5 5 6 6 4 2 4	2 1 1 1 2 2 2

```
#include<bits/stdc++.h>
using namespace std;
vector<vector <int> > graph;
vector<vector <int> > trans_graph;
stack<int> s;
vector<int> component;
vector<bool> visited;
int comp = 1;

void dfs_G(int u)
{
 visited[u] = true;
  for(int i = 0; i < graph[u].size(); i++)
  {
    int v = graph[u][i];
    if(!visited[v])
     dfs_G(v);
  }
 s.push(u);
}
void dfs_T(int u)
{
 visited[u] = true;
 component[u] = comp;
 for(int i = 0; i < trans_graph[u].size(); i++)
 {
  int v = trans_graph[u][i];
  if(!visited[v])
   dfs_T(v);
 }
}

int main()
{
 ifstream fin("cond.in");
 ofstream fout("cond.out");
 int n, m;
 fin >> n >> m;
 graph.resize(n);
 trans_graph.resize(n);
 visited.resize(n, false);
 component.resize(n);
 for(int i = 0; i < m; i++)
 {
  int u, v;
  fin >> u >> v;
  graph[u-1].push_back(v-1);
  trans_graph[v-1].push_back(u-1);
 }

 for(int i = 0; i < n; i++)
 {
  if(!visited[i])
   dfs_G(i);
 }

 visited.assign(n, false);
 while(!s.empty())
 {
       int v = s.top();
       s.pop();
       if (!visited[v])
       {
           dfs_T(v);
        comp++;
       }
```

```
    }

  fout << comp - 1 << '\n';

  for(int i = 0; i < n; i++)
   fout << component[i] << " ";
  fin.close();
  fout.close();
  return 0;

}
```

Задача E. Гамильтонов путь (2 балла)

Имя входного файла:	`hamiltonian.in`
Имя выходного файла:	`hamiltonian.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Дан ориентированный граф без циклов. Требуется проверить, существует ли в нем путь, проходящий по всем вершинам.

Формат входного файла

Первая строка входного файла содержит два целых числа n и m — количество вершин и дуг графа соответственно. Следующие m строк содержат описания дуг по одной на строке. Ребро номер i описывается двумя натуральными числами b_i и e_i — началом и концом дуги соответственно ($1 \le b_i, e_i \le n$).

Входной граф не содержит циклов и петель.

$1 \le n \le 100\,000$, $0 \le m \le 200\,000$.

Формат выходного файла

Если граф удовлетворяет требуемому условию, то выведите `YES`, иначе `NO`.

Пример

hamiltonian.in	hamiltonian.out
3 3 1 2 1 3 2 3	YES
3 2 1 2 1 3	NO

```
#include<bits/stdc++.h>
using namespace std;
#define SIZE 100000

vector<vector<int> > graph;
vector<bool> visited;
stack<int> s;
bool hamiltonian = false;
int prev_v = 0;
int cur_v = 0;

void dfs(int u)
{
 visited[u] = true;
 for(int i = 0; i < graph[u].size(); i++)
 {
  int v = graph[u][i];
  if(!visited[v])
   dfs(v);
 }
 s.push(u);
}

int main()
{
 ifstream fin("hamiltonian.in");
 ofstream fout("hamiltonian.out");
 int n, m;
 fin >> n >> m;
 graph.resize(SIZE);
 visited.assign(n, false);

 int u, v;
 for(int i = 0; i < m; i++)
 {
  fin >> u >> v;
  graph[u-1].push_back(v-1);
 }

 for(int i = 0; i < n; i++)
 {
  if(!visited[i])
  {
   dfs(i);
  }
 }

 if(!s.empty())
 {
  prev_v = s.top();
  s.pop();
  hamiltonian = true;
 }

 while(!s.empty())
 {
  cur_v = s.top();
  s.pop();
  int path = 0;

  for(int prev: graph[prev_v])
  {
   if(prev == cur_v)
    path++;
  }
```

```
  if(path == 0)
  {
   hamiltonian = false;
   break;
  }

  prev_v = cur_v;
 }

 if (hamiltonian)
  fout << "YES";
 else fout << "NO";
}
```

Задача F. Игра (2 балла)

Имя входного файла:	`game.in`
Имя выходного файла:	`game.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Дан ориентированный невзвешенный ациклический граф. На одной из вершин графа стоит «фишка». Двое играют в игру. Пусть «фишка» находится в вершине u, и в графе есть ребро (u, v). Тогда за ход разрешается перевести «фишку» из вершины u в вершину v. Проигрывает тот, кто не может сделать ход.

Формат входного файла

В первой строке входного файла находятся три натуральных числа N, M и S ($1 \le N, S, M \le 100\,000$) — количество вершин рёбер и вершина, в которой находится «фишка» в начале игры соответственно. Далее в M строках перечислены рёбра графа. Каждое ребро задаётся парой чисел — номерами начальной и конечной вершин.

Формат выходного файла

Если выигрывает игрок, который ходит первым, выведите «First player wins», иначе — «Second player wins».

Пример

game.in	game.out
3 3 1 1 2 2 3 1 3	First player wins
3 2 1 1 2 2 3	Second player wins

```cpp
#include <bits/stdc++.h>
using namespace std;
#define SIZE 100000

vector<vector<int>> graph;
vector<bool> visited;
vector<int> state;

void dfs(int u)
{
    visited[u] = true;
    for (int i = 0; i < graph[u].size(); i++)
 {
  int v = graph[u][i];
        if (!visited[v])
  {
            dfs(v);
            if (state[v] == 0)
                state[u] = 1;
        }
        else if (visited[v])
  {
            if (state[v] == 0)
                state[u] = 1;
        }
    }
}

int main()
{
    ifstream fin("game.in");
    ofstream fout("game.out");
    graph.resize(SIZE);

    int n, m, start_v;
    fin >> n >> m >> start_v;
    start_v--;
    visited.assign(n, false);
    state.assign(n, 0);

    int u, v;
    for (int i = 0; i < m; i++)
 {
        fin >> u >> v;
        graph[u-1].push_back(v-1);
    }

    for (int i = 0; i < n; i++)
 {
        if (!visited[i])
  {
            dfs(i);
        }
    }
    if (state[start_v])
        fout << "First player wins" << '\n';
    else fout << "Second player wins" << '\n';

    return 0;
}
```

Задача A. Степени вершин (1 балл) (!)

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Неориентированный граф задан списком ребер. Найдите степени всех вершин графа.

Формат входного файла

Входной файл содержит числа n ($1 \le n \le 100$) — число вершин в графе и m ($1 \le m \le n(n-1)/2$) — число ребер. Затем следует m пар чисел — ребра графа. Гарантируется, что граф не содержит петель и кратных ребер.

Формат выходного файла

Выведите в выходной файл n чисел — степени вершин графа.

Пример

input.txt	output.txt
4 4 1 2 1 3 2 3 3 4	2 2 3 1

```cpp
#include<bits/stdc++.h>
using namespace std;

int main()
{
 ifstream fin("input.txt");
 ofstream fout("output.txt");

 int n, m;
 fin >> n >> m;

 vector<vector<int>> g;
 g.resize(n);

 int u, v;
 for(int i = 0; i < m; i++)
 {
  fin >> u >> v;
  g[u-1].push_back(v-1);
  g[v-1].push_back(u-1);
 }

 for(int i = 0; i < n; i++)
 {
  fout << g[i].size() << " ";
 }

 return 0;
}
```

Задача B. Остовное дерево (2 балла)

Имя входного файла:	`spantree.in`
Имя выходного файла:	`spantree.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Даны точки на плоскости, являющиеся вершинами полного графа. Вес ребра равен расстоянию между точками, соответствующими концам этого ребра. Требуется в этом графе найти остовное дерево минимального веса.

Формат входного файла

Первая строка входного файла содержит натуральное число n — количество вершин графа ($1 \le n \le 5000$). Каждая из следующих n строк содержит два целых числа x_i, y_i — координаты i-й вершины ($-10\,000 \le x_i, y_i \le 10\,000$). Никакие две точки не совпадают.

Формат выходного файла

Первая строка выходного файла должна содержать одно вещественное число — вес минимального остовного дерева.

Примеры

spantree.in	spantree.out
3 0 0 1 0 0 1	2

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <iomanip>

using namespace std;

struct Point
{
   int x;
   int y;
};

Point *arr;
double *key;

float minKey(int &n)
{
   int min = INT_MAX;
   int min_index;
   for(int i = 1; i < n; i++)
      if(key[i] < min && key[i])
      {
         min = key[i];
         min_index = i;
      }
   return min_index;
}

bool mstSet(int &n)
{
   for(int i = 1; i < n; i++)
 {
  if(key[i])
  return false;
 }
   return true;
}

double distance(Point a, Point b)
{
 return pow(a.x - b.x, 2) + pow(a.y - b.y, 2);
}
int main()
{
   ifstream input("spantree.in");
   ofstream output("spantree.out");
   int n;
   input >> n;
   n++;
   arr = new Point[n];
   key = new double[n];

   for(int i = 1; i < n; i++)
   {
    int x, y;
      input >> x >> y;
      arr[i].x = x;
      arr[i].y = y;
      key[i] = INT32_MAX;
   }

   double result = 0;
   int v;
   while(!mstSet(n))
   {
```

```
        if (key[1])
        {
            key[1] = 0;
            v = 1;
        }
        else
        {
            v = minKey(n);
            result += sqrt(key[v]);
            key[v] = 0;
        }
        for(int u = 1; u < n; u++)
        {
            if(key[u] && u != v && key[u] > distance(arr[v], arr[u]))
            {
                key[u] = distance(arr[v], arr[u]);
            }
        }
    }
    output << setprecision(9) << result;
    return 0;
}
```

Задача C. Остовное дерево 3 (3 балла)

Имя входного файла:	`spantree3.in`
Имя выходного файла:	`spantree3.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайта

Требуется найти в связном графе остовное дерево минимального веса.

Формат входного файла

Первая строка входного файла содержит два натуральных числа n и m — количество вершин и ребер графа соответственно. Следующие m строк содержат описание ребер по одному на строке. Ребро номер i описывается тремя натуральными числами b_i, e_i и w_i — номера концов ребра и его вес соответственно ($1 \le b_i, e_i \le n$, $0 \le w_i \le 100\,000$). $n \le 50\,000, m \le 200\,000$.

Граф является связным.

Формат выходного файла

Первая строка выходного файла должна содержать одно натуральное число — вес минимального остовного дерева.

Примеры

spantree3.in	spantree3.out
4 4 1 2 1 2 3 2 3 4 5 4 1 4	7

```
#include <fstream>
#include <vector>
#include <algorithm>

using namespace std;

class Edge {
public:
    int x, y, len;
    bool operator < (Edge &e)
 {
        return this->len < e.len;
    }
};

int n;
vector<int> p (n);

int dsu_get (int v)
{
 if(v != p[v])
  p[v] = dsu_get(p[v]);
 return p[v];
}

void dsu_unite (int a, int b)
{
 a = dsu_get (a);
 b = dsu_get (b);
 if (rand() & 1)
  swap (a, b);
 if (a != b)
  p[a] = b;
}

int main() {
    ifstream fin("spantree3.in");
    ofstream fout("spantree3.out");
    int x, y, len, k = 0, m;
    long long cost = 0;
    fin >> n >> m;
    vector<Edge> graph(m);
    for (int i = 0; i < m; i++)
 {
        fin >> x >> y >> len;
        x--, y--;
        graph[i] = {x, y, len};
    }

    sort(graph.begin(), graph.end());
    p.resize (n);
    for (int i=0; i<n; ++i)
        p[i] = i;
    for (int i=0; i<m; ++i)
 {
        int a = graph[i].x,  b = graph[i].y,  l = graph[i].len;
        if (dsu_get(a) != dsu_get(b)) {
            cost += l;
            dsu_unite(a, b);
        }
    }
    fout << cost;
    return 0;
}
```

Задача A. Кратчайший путь (1 балл)

Имя входного файла:	`pathmgep.in`
Имя выходного файла:	`pathmgep.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Дан ориентированный взвешенный граф. Найдите кратчайшее расстояние от одной заданной вершины до другой.

Формат входного файла

В первой строке входного файла три числа: N, S и F ($1 \leqslant N \leqslant 2000, 1 \leqslant S, F \leqslant N$), где N — количество вершин графа, S — начальная вершина, а F — конечная. В следующих N строках по N чисел a_{ij} ($0 \leqslant a_{ij} \leqslant 10^9$) — матрица смежности графа, где -1 означает отсутствие ребра между вершинами, а любое неотрицательное число — присутствие ребра данного веса. На главной диагонали матрицы всегда нули.

Формат выходного файла

Вывести искомое расстояние или -1, если пути между указанными вершинами не существует.

Пример

pathmgep.in	pathmgep.out
3 1 2 0 -1 2 3 0 -1 -1 4 0	6

```
#include <bits/stdc++.h>
using namespace std;


void Dijkstra (vector<vector<pair <int, int>>> &graph, vector <bool> &used, vec

    dist[vertex] = 0;

    for (int i = 0; i < n; i++) {

        vertex = -1;
        for (int j = 0; j < n; j++) {
            if (!used[j] && (vertex == -1 || dist[j] < dist[vertex]))
                vertex = j;
        }

        used[vertex] = true;

        for (int j = 0; j < graph[vertex].size(); j++) {

            int to = graph[vertex][j].second;
            int weight = graph[vertex][j].first;

            dist[to] = min (dist[to], dist[vertex] + weight);
        }
    }
}


int main () {
    ifstream fin("pathmgep.in");
    ofstream fout("pathmgep.out");

    int n;
    int src;
    int last;

    fin >> n >> src >> last;

    vector <vector <pair <int, int>>> graph(n);

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {

            int weight;
            fin >> weight;
            if (weight != -1)
                graph[i].push_back({weight, j});
        }
    }

    vector <bool> used;
    used.assign(n, false);
    vector <long long> dist;
    dist.assign(n, INT64_MAX);

    Dijkstra(graph, used, dist, n, src - 1);

    if (dist[last - 1] >= INT64_MAX)
        fout << -1;
    else
```

```
        fout << dist[last - 1];

    return 0;
}
```

Задача B. Кратчайший путь от каждой вершины до каждой (1 балл) (!)

Имя входного файла:	`pathsg.in`
Имя выходного файла:	`pathsg.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Задан ориентированный взвешенный связный граф. Найдите матрицу расстояний между его вершинами.

Формат входного файла

Первая строка входного файла содержит числа n и m — количество вершин и ребер в графе соответственно ($1 \le n \le 200$, $0 \le m \le 10000$). Следующие m строк содержат по три числа — вершины, которые соединяет соответствующее ребро графа и его вес. Веса ребер неотрицательны и не превышают 10^4.

Формат выходного файла

Выведите в выходной файл n строк по n чисел — для каждой пары вершин выведите расстояние между ними.

Примеры

pathsg.in	pathsg.out
3 3 1 2 5 2 3 2 3 1 8	0 5 7 10 0 2 8 13 0

```
#include<bits/stdc++.h>
#define INF 10000

using namespace std;

class Edge
{
    public:
    int src;
    int dest;
    int weight;
    Edge(int src_, int dest_, int weight_) : src(src_), dest(dest_), weight(weig
};

void bellmanFord (vector <int> &dist, vector <Edge> &edges, int n, int m, int
{

    dist[vertex] = 0;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < m; j++) {

            int src = edges[j].src;
            int dest = edges[j].dest;
            int weight = edges[j].weight;

            dist[dest] = min(dist[dest], dist[src] + weight);
        }
    }
}

int main() {
    ifstream fin("pathsg.in");
    ofstream fout("pathsg.out");

    int n, m;
    fin >> n >> m;

    vector<Edge> edges;

    int src, dest, weight;
    for (int i = 0; i < m; i++)
    {
        fin >> src >> dest >> weight;
        Edge e(src - 1, dest - 1, weight);
        edges.push_back(e);
    }

    vector <int> dist;
    dist.assign(n, INF);
    vector <vector <int>> result;
    result.assign(n, vector <int> (n));

    for (int i = 0; i < n; i++)
    {

        bellmanFord(dist, edges, n, m, i);

        for (int j = 0; j < n; j++)
        {
            result[i][j] = dist[j];
        }
```

```
        dist.assign(n, INF);
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            fout << result[i][j] << " ";
        }
        fout << '\n';
    }

    return 0;
}
```

Задача C. Кратчайший путь (2 балла)

Имя входного файла:	`pathbgep.in`
Имя выходного файла:	`pathbgep.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Дан неориентированный взвешенный граф. Найдите кратчайшее расстояние от первой вершины до всех вершин.

Формат входного файла

В первой строке входного файла два числа: n и m ($2 \le n \le 30000, 1 \le m \le 400000$), где n — количество вершин графа, а m — количество ребер.

Следующие m строк содержат описание ребер. Каждое ребро задается стартовой вершиной, конечной вершиной и весом ребра. Вес каждого ребра — неотрицательное целое число, не превосходящее 10^4.

Формат выходного файла

Выведите n чисел — для каждой вершины кратчашее расстояние до нее.

Пример

pathbgep.in	pathbgep.out
4 5 1 2 1 1 3 5 2 4 8 3 4 1 2 3 3	0 1 4 5

```
#include<bits/stdc++.h>
using namespace std;

void Dijkstra (vector <vector<pair <int, int>>> &graph,
               vector <long long> &dist,
               set <pair <long long, int>> &MinPQ,
               int src)
{
   dist[src] = 0;
   MinPQ.insert(make_pair(0, src));

   while (!MinPQ.empty())
 {

  pair<int, int> newMinPQ = *(MinPQ.begin());
       MinPQ.erase(MinPQ.begin());
       src = newMinPQ.second;

       for (int j = 0; j < graph[src].size(); j++)
  {

           int neighbor = graph[src][j].second;
           int weight = graph[src][j].first;

           if (dist[neighbor] > dist[src] + weight) {
              MinPQ.erase({dist[neighbor], neighbor});
              dist[neighbor] = dist[src] + weight;
              MinPQ.insert({dist[neighbor], neighbor});
           }
       }
   }
}


int main ()
{
   ifstream fin("pathbgep.in");
   ofstream fout("pathbgep.out");

   int n, m;
   fin >> n >> m;

   vector <vector<pair <int, int>>> graph(n);

   int u, v, weight;
   for (int i = 0; i < m; i++)
 {
       fin >> u >> v >> weight;

       graph[u - 1].push_back({weight, v - 1});
       graph[v - 1].push_back({weight, u - 1});
   }

   vector <bool> visited;
   visited.assign(n, false);
   vector <long long> dist;
   dist.assign(n, INT64_MAX);
   set <pair<long long, int>> MinPQ;

   Dijkstra(graph, dist, MinPQ, 0);

   for (int i = 0; i < n; i++)
 {
       fout << dist[i] << " ";
   }
   fout.close();
   return 0;
```

Задача D. Кратчайшие пути и прочее (2 балла)

Имя входного файла:	`path.in`
Имя выходного файла:	`path.out`
Ограничение по времени:	2 seconds
Ограничение по памяти:	64 megabytes

Дан взвешенный ориентированный граф и вершина s в нем. Требуется для каждой вершины u найти длину кратчайшего пути из s в u.

Формат входного файла

Первая строка входного файла содержит n, m и s — количество вершин, ребер и номер выделенной вершины соответственно ($2 \le n \le 2\,000$, $1 \le m \le 5\,000$).

Следующие m строк содержат описание ребер. Каждое ребро задается стартовой вершиной, конечной вершиной и весом ребра. Вес каждого ребра — целое число, не превосходящее 10^{15} по модулю. В графе могут быть кратные ребра и петли.

Формат выходного файла

Выведите n строк — для каждой вершины u выведите длину кратчайшего пути из s в u, '*' если не существует путь из s в u и '-' если не существует кратчайший путь из s в u.

Пример

path.in	path.out
6 7 1 1 2 10 2 3 5 1 3 100 3 5 7 5 4 10 4 3 -18 6 1 -1	0 10 - - - *

```cpp
#include <bits/stdc++.h>
using namespace std;

const long long INF = 8e18;

struct Edge {
    int src;
    int dest;
    long long weight = INF;
};

vector <Edge> edges;
vector <vector<int>> graph;
vector <long long> dist;
vector <int> parent;
int relaxDest = -1;
vector <bool> visited;


void DFS (int src)
{
    visited[src] = true;

    for (int dest : graph[src])
    {
        if (!visited[dest])
        {
            DFS(dest);
        }
    }
}

ofstream fout ("path.out");

void Bellman(int n, int m, int s)
{

    dist[s] = 0;
    for (int i = 0; i < n; i++)
    {
        relaxDest = -1;
        for (int j = 0; j < m; j++)
        {
            int src = edges[j].src;
            int dest = edges[j].dest;
            long long weight = edges[j].weight;

            if (dist[src] < INF)
            {
                if (dist[dest] > dist[src] + weight)
                {
                    dist[dest] = std::max (-INF, dist[src] + weight);
                    parent[dest] = src;
                    relaxDest = dest;
                }
            }
        }
    }

  if (relaxDest != -1)
    {
```

```
        for (int i = 0; i < n; i++)
            relaxDest = parent[relaxDest];

        DFS(relaxDest);

        for (int i = 0; i < n; i++)
        {
            if (visited[i])
                dist[i] = -INF;
        }
    }

    for (auto d: dist)
    {
        if (d == -INF)
            fout << "-\n";
        else if ( d == INF)
            fout << "*\n";
        else
            fout << d << '\n';
    }
}


int main() {
    ifstream fin ("path.in");
    int n, m, s;
    fin >> n >> m >> s;

    dist.assign(n, INF);
    parent.assign(n, -1);
    graph.resize(n);
    visited.resize(n);

    int src, dest;
    long long weight;

    for (int i = 0; i < m; i++)
    {
        fin >> src >> dest >> weight;
        edges.push_back({src - 1, dest - 1, weight});
        graph[src - 1].push_back(dest - 1);
    }

    Bellman(n, m, s - 1);
    return 0;
}
```

Задача E. Цикл отрицательного веса (1 балл)

Имя входного файла:	`negcycle.in`
Имя выходного файла:	`negcycle.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Дан ориентированный взвешенный граф. Определить, есть ли в нем цикл отрицательного веса, и если да, то вывести его.

Формат входного файла

Во входном файле в первой строке число n ($1 \le n \le 250$) — количество вершин графа. В следующих n строках находится по n чисел — матрица смежности графа. Все веса ребер не превышают по модулю 10000. Если ребра нет, то соответствующее число равно 10^9.

Формат выходного файла

В первой строке выходного файла выведите `YES`, если цикл существует или `NO` в противном случае. При его наличии выведите во второй строке количество вершин в искомом цикле (считая одинаковые первую и последнюю) и в третьей строке — вершины, входящие в этот цикл в порядке обхода.

Примеры

negcycle.in	negcycle.out
2 0 -1 -1 0	YES 3 1 2 1

```
#include <bits/stdc++.h>
#define INF 1000000000
using namespace std;

struct Edge
{
   int src, dest, weight;
};

int n;
int vertex = -1;
vector<Edge> graph;
vector<int> dist;
vector<int> parent;
vector<int> way;

bool ford() {
   dist[0] = 0;
   for (int i = 0; i < n; i++)
 {
      vertex = -1;
      for (Edge edge : graph)
  {
         if (dist[edge.dest] > dist[edge.src] + edge.weight)
  {
            dist[edge.dest] = dist[edge.src] + edge.weight;
            parent[edge.dest] = edge.src;
            vertex = edge.dest;
         }
      }
   }

   if (vertex != -1)
 {
      int negCycleEnd = vertex;
      for (int i = 0; i < n; i++)
         negCycleEnd = parent[negCycleEnd];
      for (int negCycleNow = negCycleEnd; negCycleNow != negCycleEnd || way.empty(); negCycleNow =
parent[negCycleNow])
         way.push_back(negCycleNow);
      way.push_back(negCycleEnd);
      reverse(way.begin(), way.end());
      return true;
   }
   else if(vertex == -1)
   {
    return false;
 }
}

int main() {
   ifstream fin("negcycle.in");
   ofstream fout("negcycle.out");
   fin >> n;
   dist.resize(n, INF);
   graph.resize(n);
   parent.resize(n);
   for (int src = 0; src < n; src++) {
      for (int dest = 0; dest < n; dest++)
 {
         int weight;
       fin >> weight;
         graph.push_back({ src, dest, weight });
      }
   }
   if(ford())
```

```
    {
        fout << "YES\n";
        fout << way.size() << endl;
        for (int vertex : way)
            fout << vertex + 1 << ' ';
    }
    else
    fout << "NO";
    return 0;
}
```

Задача A. Максимальный поток (2 балла)

Имя входного файла:	`maxflow.in`
Имя выходного файла:	`maxflow.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Задан ориентированный граф, каждое ребро которого обладает целочисленной пропускной способностью. Найдите максимальный поток из вершины с номером 1 в вершину с номером n.

Формат входного файла

Первая строка входного файла содержит n и m — количество вершин и количество ребер графа ($2 \le n \le 100$, $1 \le m \le 1000$). Следующие m строк содержат по три числа: номера вершин, которые соединяет соответствующее ребро графа и его пропускную способность. Пропускные способности не превосходят 10^5.

Формат выходного файла

В выходной файл выведите одно число — величину максимального потока из вершины с номером 1 в вершину с номером n.

Примеры

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

```cpp
#include <fstream>
#include <vector>
#include <queue>
using namespace std;

vector <vector <int>> graph;
int u, v;
queue<int> q;
vector<bool> visited;
vector <int> parent;

bool BFS(int s, int t)
{
   visited.assign(graph.size(), false);
     q.push(s);
     visited[s] = true;

     while (!q.empty())
     {
         u = q.front();
         q.pop();

         for (v = 0; v < graph.size(); v++)
         {
             if (visited[v] == false && graph[u][v] > 0)
             {
                 q.push(v);
                 visited[v] = true;
                 parent[v] = u;
             }
         }
     }

     return visited[t];
}

int fordFulkerson(int s, int t)
{
     int max_flow = 0;
     int path_flow = 0;


     while (BFS(s, t))
     {
         path_flow = INT32_MAX;

         for (v = t; v != s; v = parent[v])
         {
             u = parent[v];
             path_flow = std::min (path_flow, graph[u][v]);
         }

         for (v = t; v != s; v = parent[v])
         {
             u = parent[v];
             graph[u][v] -= path_flow;
             graph[v][u] += path_flow;
         }

         max_flow += path_flow;
     }
```

```
    return max_flow;
}

int main()
{
    ifstream fin("maxflow.in");
    fstream fout("maxflow.out");

    int n, m;
    fin >> n >> m;
    graph.resize(n, vector <int> (n));
    parent.resize(n);

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            graph[i][j] = 0;

    int u, v, capacity;
    for (int i = 0; i < m; i++)
    {
        fin >> u >> v >> capacity;
        graph[u - 1][v - 1] = capacity;
    }
    fout << fordFulkerson(0, n - 1);

    return 0;
}
```

Задача B. Паросочетание (2 балла)

Имя входного файла:	`matching.in`
Имя выходного файла:	`matching.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Дан двудольный невзвешенный граф. Необходимо найти максимальное паросочетание.

Формат входного файла

В первой строке входного файла три целых числа n, m и k ($1 \le n, m \le 200$, $1 \le k \le n \times m$) — количество чисел в первой и второй долях, а также число ребер соответственно. Далее следуют k строк, в каждой из которых два числа a_i и b_i, что означает ребро между вершиной с номером a_i первой доли и вершиной с номером b_i второй доли. Вершины в обеих долях нумеруются с единицы.

Формат выходного файла

В выходной файл выведите одно число — максимальное число ребер в паросочетании.

Примеры

matching.in	matching.out
3 3 5 1 1 1 3 2 1 2 2 3 2	3

```cpp
#include <bits/stdc++.h>
using namespace std;

vector<vector<char>> graph;
vector<char> seen;
vector<int> matchNM;

bool dfs(int u, int M)
{
    for (int v = 0; v < M; ++v)
    {
        if (graph[u][v] && seen[v] == false)
        {
            seen[v] = 1;

            if (matchNM[v] == -1 ||
                dfs(matchNM[v], M))
            {
                matchNM[v] = u;
                return true;
            }
        }
    }
    return false;
}

int max_matching(int N, int M)
{
    for (int u = 0; u < N; ++u)
    {
        seen.assign(M, 0);
        dfs(u, M);
    }

    return M - std::count(matchNM.begin(), matchNM.end(), -1);
}


int main()
{
    ifstream fin("matching.in");
    ofstream fout("matching.out");

    int N = 0, M = 0, k = 0;
    fin >> N >> M >> k;
    matchNM.assign(M, -1);
    seen.assign(M, 0);

    graph.resize(N);
  for (int i = 0; i < N; ++i)
     graph[i].resize(M);

    for(int i = 0; i < N; i++)
      for(int j = 0; j < M; j++)
        graph[i][j] = 0;

    for (int i = 0; i < k; ++i)
    {
        int u = 0, v = 0;
        fin >> u >> v;
        graph[u - 1][v - 1] = 1;
```

```
    }
    fout << max_matching(N, M);
    return 0;
}
```

Задача C. Декомпозиция потока (3 балла)

Имя входного файла:	`decomposition.in`
Имя выходного файла:	`decomposition.out`
Ограничение по времени:	5 секунд
Ограничение по памяти:	256 мегабайт

Задан ориентированный граф, каждое ребро которого обладает целочисленной пропускной способностью. Найдите максимальный поток из вершины с номером 1 в вершину с номером n и постройте декомпозицию этого потока.

Формат входного файла

Первая строка входного файла содержит n и m — количество вершин и количество ребер графа ($2 \le n \le 500, 1 \le m \le 10000$). Следующие m строк содержат по три числа: номера вершин, которые соединяет соответствующее ребро графа и его пропускную способность. Пропускные способности не превосходят 10^9.

Формат выходного файла

В первую строку выходного файла выведите одно число — количество путей в декомпозции максимального потока из вершины с номером 1 в вершину с номером n. Следующий строки должны содержать описания элементарых потоков, на который был разбит максимальный. Описание следует выводить в следующем формате: величина потока, количество ребер в пути, вдоль которого течет данный поток и номера ребер в этом пути. Ребра нумеруются с единицы в порядке появления во входном файле.

Примеры

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

```cpp
#include <fstream>
#include <queue>
#include <algorithm>
using namespace std;

struct Edge
{
    int u, v, capacity, flow, id;
};
vector<vector<int>> graph;
vector<Edge> edges;
vector<int> ptr;
vector<int> level;
vector<vector<int>> path;

int n, m, s = 0, t = 0;
//Finds if more flow can be sent from s to t.
// Also assigns levels to nodes.
bool bfs() {
    queue<int> q;
    level.assign(n, 0);
    q.push(s);
    level[s] = 1;
    while (!q.empty() && !level[t]) {
        int u = q.front();
        q.pop();
        for (auto& e : graph[u]) {
            if (!level[edges[e].v] && edges[e].capacity > edges[e].flow) {
                q.push(edges[e].v);
                level[edges[e].v] = level[u] + 1;
            }
        }
    }
    return level[t];
}

int dfs(int vertex, int flow) {
    if (!flow)
        return 0;
   //if we reach sink, we return flow
    if (vertex == t)
        return flow;
    // Traverse all adjacent edges one -by - one.
    while (ptr[vertex] < graph[vertex].size()) {
      //pick next edge from adj list vertex
        int e = graph[vertex][ptr[vertex]];
        if (level[vertex] + 1 == level[edges[e].v]){
        // find minimum flow from vertex to t
            int temp_flow = dfs(edges[e].v, min(flow, edges[e].capacity - edges[e]
           // flow is greater than zero
       if (temp_flow) {
             // add flow  to current edge
                edges[e].flow += temp_flow;
                // subtract flow from reverse edge
                // of current edge
                edges[e ^ 1].flow -= temp_flow;
                return temp_flow;
            }
        }
        ptr[vertex]++;  // count of edges explored from i. it is to keep track of next ed
    }
```

```
    return 0;
}
//Output order: flow, number of edges, edge ID s
int decompose(int vertex, int cur_flow)
{
    if (vertex == t)
   {
        path.emplace_back();
        return cur_flow;
    }
    //find path from s -> t
    for (auto& e : graph[vertex])
   {
    //If the flow is still positive, we find a path at every step:
        if (edges[e].flow > 0)
    {
            int result = decompose(edges[e].v, min(cur_flow, edges[e].flow));
            if (result)
      {
                path.back().push_back(edges[e].id); //edge id
                if (vertex == s)
        {
                    path.back().push_back(path[path.size() - 1].size()); //number of e
                    path.back().push_back(result);//flow
                    reverse(path.back().begin(), path.back().end());
                }
                edges[e].flow -= result;
                return result;
            }
        }
    }
    return 0;
}

int main() {
    ifstream fin("decomposition.in");
    ofstream fout("decomposition.out");
    fin >> n >> m;
    graph.resize(n);
    ptr.resize(n, 0);
    for (int i = 0; i < m; i++) {
        int u, v, capacity;
        fin >> u >> v >> capacity;
        graph[u - 1].push_back(edges.size());
        edges.emplace_back(Edge{u - 1, v - 1, capacity, 0, i + 1});
        graph[v - 1].push_back(edges.size());
        edges.emplace_back(Edge{v - 1, u - 1, 0, 0, i + 1});
    }
    t = n - 1;
    while (bfs()) {
        while (dfs(s, INT_MAX));
        ptr.assign(n, 0);
    }
    while (decompose(s, INT_MAX));
    fout << path.size() << "\n";
    for (auto& i : path) {
        for (auto& j : i)
            fout << j << ' ';
        fout << '\n';
    }
    return 0;
```

Задача A. Наивный поиск подстроки в строке (!) (1 балл)

Имя входного файла:	`search1.in`
Имя выходного файла:	`search1.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Даны строки p и t. Требуется найти все вхождения строки p в строку t в качестве подстроки.

Формат входного файла

Первая строка входного файла содержит p, вторая — t ($1 \le |p|, |t| \le 10^4$). Строки состоят из букв латинского алфавита.

Формат выходного файла

В первой строке выведите количество вхождений строки p в строку t. Во второй строке выведите в возрастающем порядке номера символов строки t, с которых начинаются вхождения p. Символы нумеруются с единицы.

Примеры

search1.in	search1.out
aba abaCaba	2 1 5

```cpp
#include<bits/stdc++.h>
using namespace std;

vector<int> pattern;

vector<int> search(string pat, string txt)
{
   int M = pat.size();
   int N = txt.size();

   for(int i = 0; i <= N - M; i++)
   {
     int j;
     for(j = 0; j < M; j++)
     {
       if(txt[i+j] != pat[j])
         break;
     }

     if(j == M)
       pattern.push_back(i+1);
   }
   return pattern;
}

int main()
{
   ifstream fin("search1.in");
   ofstream fout("search1.out");

   string p, t;
   fin >> p >> t;

   search(p, t);
   fout << pattern.size() << '\n';

   for(auto i: pattern)
   {
     fout << i << " ";
   }
}
```

Задача B. Быстрый поиск подстроки в строке (2 балла)

Имя входного файла:	`search2.in`
Имя выходного файла:	`search2.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Даны строки p и t. Требуется найти все вхождения строки p в строку t в качестве подстроки.

Формат входного файла

Первая строка входного файла содержит p, вторая — t ($1 \le |p|, |t| \le 10^6$). Строки состоят из букв латинского алфавита.

Формат выходного файла

В первой строке выведите количество вхождений строки p в строку t. Во второй строке выведите в возрастающем порядке номера символов строки t, с которых начинаются вхождения p. Символы нумеруются с единицы.

Примеры

search2.in	search2.out
aba abaCaba	2 1 5

```cpp
#include <bits/stdc++.h>
using namespace std;

void computeLPSArray(string pat, int M, int* lps);
vector<int> ans;

vector<int> KMPSearch(string pat, string txt)
{
    int M = pat.size();
    int N = txt.size();
    int lps[M];

    computeLPSArray(pat, M, lps);

    int i = 0;
    int j = 0;
    while (i < N) {
        if (pat[j] == txt[i]) {
            j++;
            i++;
        }

        if (j == M) {
          ans.push_back(i - j + 1);
            j = lps[j - 1];
        }

        else if (i < N && pat[j] != txt[i]) {
            if (j != 0)
                j = lps[j - 1];
            else
                i = i + 1;
        }
    }
    return ans;
}

void computeLPSArray(string pat, int M, int* lps)
{
    int len = 0;
    lps[0] = 0;

    int i = 1;
    while (i < M) {
        if (pat[i] == pat[len]) {
            len++;
            lps[i] = len;
            i++;
        }
        else
        {
            if (len != 0) {
                len = lps[len - 1];
            }
            else
            {
                lps[i] = 0;
                i++;
            }
        }
    }
```

```
}

int  main()
{
   ifstream  fin("search2.in");
   ofstream  fout("search2.out");
     string  t,  p;
     fin  >>  p  >>  t;
     KMPSearch(p,  t);
     fout  <<  ans.size()  <<  '\n';
   for(int  i  =  0;  i  <  ans.size();  i++)
   {
     fout  <<  ans[i]  <<  " ";
   }


     return  0;
}
```

Задача C. Префикс-функция (2 балла)

Имя входного файла:	`prefix.in`
Имя выходного файла:	`prefix.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Постройте префикс-функцию для заданной строки s.

Формат входного файла

Первая строка входного файла содержит s ($1 \le |s| \le 10^6$). Строка состоит из букв латинского алфавита.

Формат выходного файла

Выведите значения префикс-функции строки s для всех индексов $1, 2, \ldots, |s|$.

Примеры

prefix.in	prefix.out
aaaAAA	0 1 2 0 0 0
abacaba	0 0 1 0 1 2 3

```cpp
#include <bits/stdc++.h>
using namespace std;

vector<int> prefix_function(string s)
{
     int n = (int)s.length();
     vector<int> pi(n);
     for (int i = 1; i < n; i++) {
         int j = pi[i-1];
         while (j > 0 && s[i] != s[j])
             j = pi[j-1];
         if (s[i] == s[j])
             j++;
         pi[i] = j;
     }
     return pi;
}

int main()
{
   ifstream fin("prefix.in");
     ofstream fout("prefix.out");

     string s;
     fin >> s;
     vector<int> prefix = prefix_function(s);
     for (auto& i : prefix)
   {
         fout << i << " ";
     }
     return 0;
}
```

Задача D. Автомат Кнута-Морриса-Пратта (3 балла)

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Постройте автомат Кнута-Морриса-Пратта для заданной мощности алфавита n и строки s.

Формат входного файла

В первой строке находится число n ($1 \le n \le 26$) — мощность алфавита. Во второй строке находится строка, состоящая из строчных латинских букв ($s \le 10^5$). Гарантируется, что в данной строке не встречается символ, номер в алфавите которого больше n.

Формат выходного файла

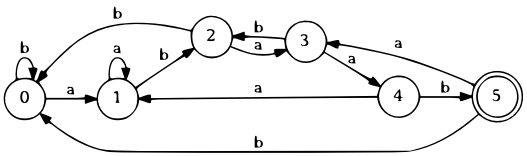
Выведите $|s| + 1$ строк, где j-я строка состоит из n чисел, разделенных пробелом — i-е число в строке обозначает, в какое состояние автомата мы перейдем по i-му символу из j-го состояния автомата. Состояния автомата нумеруются с 0. Если мы передадим в качестве слова автомату префикс данного слова длины k, то мы должны оказаться в состоянии с номером k.

Примеры

стандартный ввод	стандартный вывод
3 abacaba	1 0 0 1 2 0 3 0 0 1 2 4 5 0 0 1 6 0 7 0 0 1 2 4
2 abaab	1 0 1 2 3 0 4 2 1 5 3 0

Примечание

Пример построенного автомата для последнего примера.



```cpp
#include <bits/stdc++.h>
using namespace std;

vector<int> prefix_function(string s)
{
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}

void compute_automaton(string s, int limit, vector<vector<int> > &aut)
{
    s += "#";
    int n = s.size();
    vector<int> pi = prefix_function(s);
    aut.assign(n, vector<int>(limit));
    for (int i = 0; i < n; i++)
  {
        for (int c = 0; c < limit; c++) {
            if (i > 0 && 'a' + c != s[i])
                aut[i][c] = aut[pi[i-1]][c];
            else
                aut[i][c] = i + ('a' + c == s[i]);
        }
    }
}

int main()
{
    int limit;
    string s;
     cin >> limit;
    cin >> s;
  vector<int> prefix = prefix_function(s);
    vector<vector<int> > KMP(s.length(), vector<int>(limit));
    compute_automaton(s, limit, KMP);
  for (auto& row : KMP)
  {
        for (auto& i : row) {
            cout << i << " ";
        }
        cout << '\n';
    }
    return 0;
}
```

Anujin Baatartsogt 2020.06.12