Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

УТВЕРЖДАЮ
Зав.кафедрой,
к.ф.-м.н.
_____ С. В. Миронов

ОТЧЕТ О ПРАКТИКЕ

студента 2 курса 251 группы факультета КНиИТ
Соколкова Павла Вячеславовича

вид практики: педагогическая
кафедра: математической кибернетики и компьютерных наук
курс: 2
семестр: 4
продолжительность: 2 нед., с 12.07.2016 г. по 21.07.2016 г.

Руководитель практики от университета,

доцент, к. ф.-м. н. _____ Ю. Н. Кондратова

Руководитель практики от организации (учреждения, предприятия),

руководитель ЦОПП _____ М. Р. Мирзаянов

Тема практики:«Мосты и точки сочленения в графах. Компоненты сильной связности. Конденсация графа»

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Граф — формальная модель различных дискретных систем внешнего мира.

Мост — такое ребро графа, удаление которого приводит к увеличению числа компонент связности.

Древесное ребро — ребро, образующее дерево поиска в глубину.

Обратное ребро — ребро, ведущее вверх по дереву.

Точка сочленения — это такая вершина, удаление которой и всех инцидентных ей рёбер увеличивает количество компонент связности.

Обход — алгоритм, который в некотором порядке посещает вершины графа.

DFS (*Depth First Search*) — поиск в глубину.

BFS (*Breadth First search*) — поиск в ширину.

Орграф — ориентированный граф.

ВВЕДЕНИЕ

Целью практики является изучение алгоритмов, методов решения олимпиадных задач, освоение навыков преподавания нового материала, проведение разборов задач для других участников.

В результате прохождения практики должны быть отработаны навыки:

- изучения нового материала;
- решения задач тематического характера;
- решения олимпиадных задач;
- проведения самостоятельного разбора задач для других участников;
- самостоятельная подготовка к соревнованиям;
- алгоритм поиска мостов в графе;
- алгоритм поиска точек сочленения в графе;
- выделение компонент сильной связности в ориентированном графе;
- конденсация графа;

1 Теоретические сведения

1.1 Мосты в графах

Мост — это ребро, которое не принадлежит какому-либо простому циклу. Мосты выбираются только из древесных рёбер.

Рассмотрим дерево DFS. Пусть (u, v) – древесное ребро. Вершина u – верхняя вершина, а вершина v – нижняя вершина, тогда (u, v) не является мостом тогда и только тогда, когда найдётся обратное ребро, которое начинается в поддереве вершины v, а заканчивается в вершине u или выше. Таким образом, для каждого древесного ребра необходимо проверить, есть ли подходящее обратное ребро.

В процессе поиска в глубину будем строить для вершин массив целочисленных пометок fup (forward up), где $fup[x] =$ наименьшая глубина такой вершины y, что вершина y достижима из x путём перехода 0 или более раз вниз по дереву (forward) и не более 1 раза по обратному ребру вверх (up).

Для лучшего понимания рассмотрим на рисунке 1 некоторое дерево обхода в глубину и для каждой вершины укажем соответствующее ей значение пометки fup. Около каждой вершины указан её номер и через запятую значение fup для этой вершины.

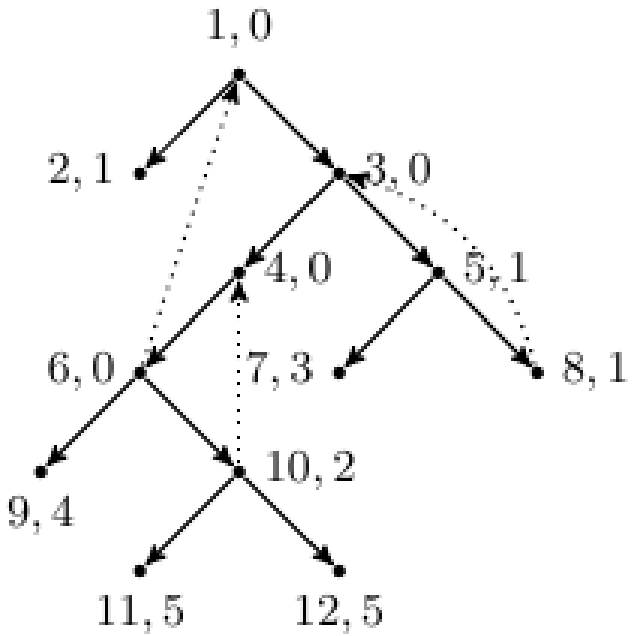

Рисунок 1 – Значения fup вершин в дереве обхода в глубину

Ребро (u, v) не является мостом, если найдётся обратное ребро из поддерева вершины v, которое ведёт строго выше вершины v. Иными словами, (u, v)

не мост тогда и только тогда, когда $fup[v] < v$ в дереве обхода (аналогично $fup[v] \leq u$). На рисунке 2 представлена данная ситуация. Волнистой линией показан некоторый путь от корня дерева до вершины u. Вершина y – некоторая вершина, которая лежит выше u (или совпадает с u), x – некоторая вершина в поддереве v (может совпадать с v). Поэтому в ходе работы алгоритма будем сравниваться значение пометки с глубиной v.

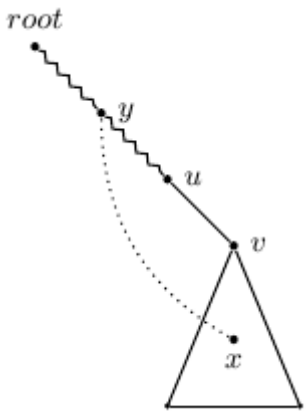

Рисунок 2 – Мост в дереве обхода

Ниже представлен псевдокод алгоритма для выделения мостов за $O(n + m)$, где n – количество вершин, m – количество рёбер. Массив dep содержит информацию о реальной глубине вершины в дереве. Если вершина v еще не встречалась в обходе, то $dep[v] = -1$.

```
dfs(u, d, p)
        fup[u] = dep[u] = d;
        for (v : g[u])
                if (v == p)
                        continue;
                if (dep[v] >= 0)
                        fup[u] = min(fup[u], dep[v]);
                else
                        dfs(v, d + 1, u);
                        fup[u] = min(fup[u], fup[v]);
                        if (fup[v] > d)
                                (u, v) - мост;
```

1.2 Точки сочленения в графе

Точки сочленения также называю *шарнирными вершинами*. Корень дерева dfs является точкой сочленения тогда и только тогда, когда в дереве dfs

у него больше 1 сына. Для всех остальных вершин u достаточно наличие хотя бы одной вершины v, которая является потомком в дереве обхода, для которой $fup[v] \geq d$, где d – глубина вершины u.

Для лучшего понимания рассмотрим рисунок 3, на котором точки сочленения выделены красным цветом.

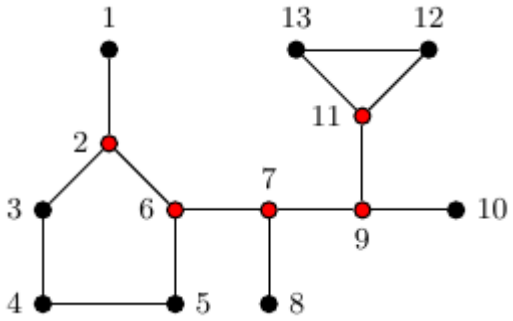

Рисунок 3 – Точки сочленения в неориентированном графе

Ниже представлен алгоритм для выделения точек сочленения в графе за $O(n+m)$, где n – количество вершин, m – количество рёбер. Как и для выделения мостов $fup[v]$ – минимальная глубина, достижимая из v, $dep[u]$ – глубина вершины u (массив изначально инициализируется -1 для всех вершин).

```
dfs(u, d, p)
        fup[u] = dep[u] = d;
        sons = 0;
        for (v : g[u])
                if (v == p)
                        continue;
                if (dep[v] >= 0)
                        fup[u] = min(fup[u], dep[v]);
                else
                        sons++;
                        dfs(v, d + 1, u);
                        fup[u] = min(fup[u], fup[v]);
                        if (u != p && fup[v] >= d)
                                u - точка сочленения;
        if (u == p && sons > 1)
                u - точка сочленения;
```

1.3 Компоненты сильной связности. Конденсация графа.

Вершины u и v находятся в отношении сильной связности тогда и только тогда, когда из u достижима v и из v достижима u. Иными словами, u и v

принадлежат одному (не обязательно простому) циклу.

На рисунке 4 вершины, принадлежащие одной компоненте сильной связности, окрашены в один цвет.

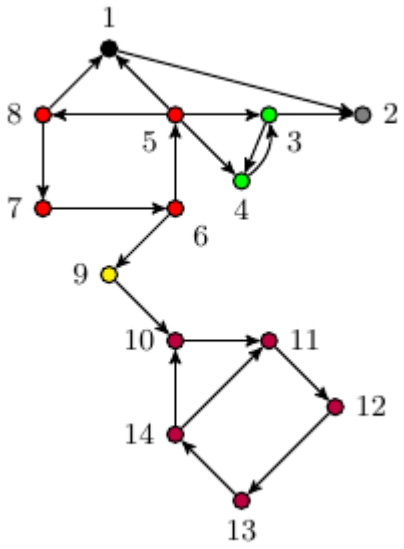

Рисунок 4 – Компоненты сильной связности в орграфе

Отношение взаимной достижимости — это отношение эквивалентности:

— Каждая вершина достижима из самой себя — рефлексивность;
— Если u взаимно достижима с v, то v взаимно достижима с u — симметричность;
— Если u взаимно достижима с v, а v взаимно достижима с w, то u взаимно достижима с w — транзитивность;

Компонента сильной связности — это блок отношения взаимной достижимости. Иными словами, это максимальное по включению подмножество вершин, в котором любая пара вершин взаимно достижима. Иными словами, это максимальное по включению подмножество, в котором любые 2 вершины принадлежат 1 (не обязательно простому) циклу.

Рассмотрим орграф, который получается из заданного стягиванием каждого цикла в 1 вершину (петли удаляются). Получившийся граф не содержит циклов и называется конденсацией.

На рисунке 5 представлена конденсация графа из рисунка 4.

Задача:

Выделить в орграфе компоненты сильной связности.

Если граф ациклический или все его циклы – петли, то все компоненты сильной связности имеют размер 1 (сами вершины).

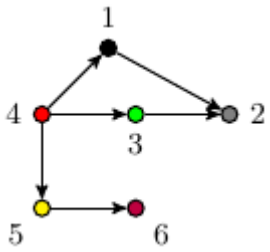

Рисунок 5 – Компоненты сильной связности в орграфе

Алгоритм выделения компонент связности и конденсации графа:

1. Сделаем серию поисков в глубину и выпишем все вершины в порядке увеличения времени окончания их обработки;
2. Перевернём все дуги графа (рассмотрим транспонированный граф);
3. Отдельной серией поисков в глубину пройдём по всем вершинам транспонированного графа в порядке противоположном пункту 1;
4. Каждое отдельное дерево поиска в глубину из предыдущего пункта — компонента сильной связности.

Например, для графа, представленного на рисунке 6, массив вершин для пункта 1 алгоритма будет следующим: $\{4, 5, 6, 1, 2, 3, 7\}$. При этом вершины $1, 5, 4$ принадлежат одной компоненте связности.

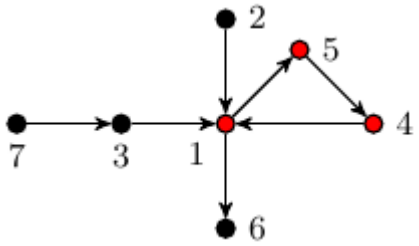

Рисунок 6 – Пример орграфа для пункта 1

А для пункта 2 граф будет выглядеть следующим образом – рисунок 7.

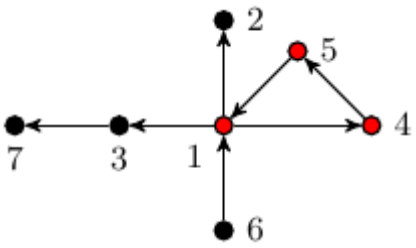

Рисунок 7 – Транспонированный граф

Этот алгоритм автоматически нумерует компоненты сильной связности в порядке топологической сортировки конденсации.

Замечание: в реализации проще на этапе построения (чтения графа) хранить как прямой (обычный), так и обратный (транспонированный) орграф.

2 Примеры задач и их решения

2.1 Мосты

Условие.

ограничение по времени на тест: 1 секунда

ограничение по памяти на тест: 256 мегабайт

ввод: стандартный ввод

вывод: стандартный вывод

Дан неориентированный граф. Требуется найти все мосты в нем.

Входные данные. Первая строка входных данных содержит два натуральных числа n и m — количества вершин и ребер графа соответственно ($1 \leq n \leq 20000$, $1 \leq m \leq 200000$).

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

Граф не обязательно связный. Не содержит петель и кратных ребер.

Выходные данные. Первая строка выходных данных должна содержать одно натуральное число b – количество мостов в заданном графе. На следующей строке выведите b целых чисел – номера ребер, которые являются мостами, в возрастающем порядке. Ребра нумеруются с единицы в том порядке, в котором они заданы во входном файле.

Решение. Создадим массивы g, fup, dep. В массиве g будем хранить граф списком смежности, в fup будем хранить минимальную высоту в дереве обхода в глубину, которую можно достичь из i-й вершины нулём или более спусками вниз и 1 подъёмом по обратному ребру. Массив dep будет содержать настоящую глубину вершин, изначально все элементы равны -1. Сделаем серию поисков в глубину. Пусть мы находимся в вершине x и хотим перейти в вершину y. Если y не является непосредственным предком x, из которого мы пришли в x, то, если ребро (x, y) обратное ($dep[y] \neq -1$), обновим значение $fup[x]$. В ином случае запустим обход в глубину из y. Обновим значение $fup[x]$. Если $fup[y] > d$, то ребро (x, y) – мост, внесём его номер в ответ.

Полный код программы приведен в приложении A.

2.2 Точки сочленения

Условие.

ограничение по времени на тест: 1 секунда

ограничение по памяти на тест: 256 мегабайт

ввод: стандартный ввод

вывод: стандартный вывод

Дан неориентированный граф. Требуется найти все точки сочленения в нем.

Входные данные. Первая строка входного файла содержит два натуральных числа n и m — количества вершин и ребер графа соответственно ($1 \leq n \leq 20000$, $1 \leq m \leq 200000$).

Следующие m строк содержат описание ребер по одному на строке. Ребро номер i описывается двумя натуральными числами b_i, e_i – номерами концов ребра ($1 \leq b_i, e_i \leq n$).

Выходные данные. Первая строка выходного файла должна содержать одно натуральное число b – количество точек сочленения в заданном графе. На следующей строке выведите b целых чисел – номера вершин, которые являются точками сочленения, в возрастающем порядке.

Решение. Создадим массивы g, fup, dep. В массиве g будем хранить граф списком смежности, в fup будем хранить минимальную высоту в дереве обхода в глубину, которую можно достичь из i-й вершины нулём или более спусками вниз и 1 подъёмом по обратному ребру. Массив dep будет содержать настоящую глубину вершин, изначально все элементы равны -1. Сделаем серию поисков в глубину. Для каждой вершины заведём переменную $sons$, которая изначально равно 0. Пусть мы находимся в вершине x и хотим перейти в вершину y. Если y не является непосредственным предком x, из которого мы пришли в x, то, если ребро (x, y) обратное ($dep[y] \neq -1$), обновим значение $fup[x]$. В ином случае увеличим значение переменной $sons$, запустим обход в глубину из y. Обновим значение $fup[x]$. Если $fup[y] \geq d$ и x не является корнем ($x \neq p$, где p – предок), то вершина x – точка сочленения, внесём её в ответ. После просмотра всех соседей точки x проверим, является ли она корнем, и если является и имеет более 1 ребёнка (переменная $sons$), то внесём её в ответ.

Полный код программы приведен в приложении Б.

2.3 Конденсация графа (усложненная версия)

Условие.

ограничение по времени на тест: 1 секунда

ограничение по памяти на тест: 256 мегабайт

ввод: стандартный ввод

вывод: стандартный вывод

Вам задан связный ориентированный граф с n вершинами и m рёбрами. Петли и кратные ребра допускаются.

Такие пары вершин u, v, что из u достижима v и из v достижима u принадлежат одной компоненте сильной связности орграфа.

Очевидно, что если построить новый орграф у которого вершиной является компонента сильной связности старого, а дуга между парой вершин существует тогда и только тогда, когда она есть между какой-то парой вершин из соответствующих компонент сильной связности, то этот граф будет ациклическим. Такой граф называется конденсацией. Так как ациклический, его возможно топологически отсортировать. Сделайте это.

Входные данные. Граф задан во входном файле следующим образом: первая строка содержит числа n и m ($1 \leq n \leq 20000$, $1 \leq m \leq 200000$).

Каждая из следующих m строк содержат описание ребра – два целых числа из диапазона от 1 до n – номера начала и конца ребра.

Выходные данные. В первую строку выведите число k — количество компонент сильной связности в заданном графе.

В следующую строку выведите n чисел — для каждой вершины выведите номер компоненты сильной связности, которой принадлежит эта вершина. Компоненты сильной связности должны быть занумерованы таким образом, чтобы для любого ребра номер компоненты сильной связности его начала не превышал номера компоненты сильной связности его конца.

Решение. Создадим массивы g, gr, $used$, fo. В массиве g будем хранить граф списком смежности, в gr транспонированный граф, массив $used$ будет содержать информацию о рассмотренных вершинах, в fo будет содержаться список вершин в порядке увеличения времени их обработки. Сделаем серию поисков в глубину. Перевернём массив fo, очистим массив $used$. Создадим переменную clr. Затем серией поисков в глубину в порядке нового массива fo будем запускать обход из очередной вершины и красить её в цвет clr. После каждого обхода, увеличивая clr. Таким образом, каждая компонента связности будет окрашена в свой цвет, эта информация будет храниться в массиве $used$.

Так как алгоритм автоматически нумерует компоненты сильной связности в порядке топологической сортировки, осталось только вывести цвет каждой вершины из массива $used$.

Полный код программы приведен в приложении В.

ЗАКЛЮЧЕНИЕ

В ходе прохождения практики были изучены и отработаны навыки по следующим темам:

- графы;
- реализация программ;
- структуры данных;
- поиск в глубину, обход в ширину;
- поиск мостов в графе;
- поиск точек сочленения в графе;
- поиск компонент сильной связности в орграфе;
- конденсация графа;
- применение лекционного материала на практике;
- проведение разборов задач;

ПРИЛОЖЕНИЕ А
Программный код задачи «Мосты»

```
#define _CRT_SECURE_NO_WARNINGS
#include <vector>
#include <iostream>
#include <string>
#include <set>
#include <map>
#include <fstream>
#include <iomanip>
#include <queue>
#include <cmath>
#include <algorithm>

using namespace std;

#define mp make_pair
#define pb push_back
#define X first
#define Y second
#define all(a) a.begin(), a.end()

long long  INF = 1e9;
int n, m;

vector<vector<pair<int,int>>> g(n, vector<pair<int,int>>());
vector<int> fup(n);
vector<int> dep(n, -1);
set<int> answ;

void dfs(int u, int d, int p){
  fup[u] = dep[u] = d;

  for (int i = 0; i < g[u].size(); i++){
    int v = g[u][i].first;
    if (v == p){
      continue;
    }
    if (dep[v] >= 0){
      fup[u] = min(fup[u], dep[v]);
    }
    else{
      dfs(v, d + 1, u);
      fup[u] = min(fup[u], fup[v]);
      if (fup[v] > d){
        answ.insert( g[u][i].second);
      }
```

```
46      }
47    }
48  }
49
50  int main() {
51  #ifdef _DEBUG
52    freopen("input.txt", "r", stdin);
53    freopen("output.txt", "w", stdout);
54  #endif
55
56    cin >> n >> m;
57
58    g = vector<vector<pair<int,int>>>(n, vector<pair<int,int>>());
59    fup = vector<int> (n);
60    dep = vector<int> (n, -1);
61
62    for (int i = 0; i < m; i++){
63      int a, b;
64      cin >> a >> b;
65      a--, b--;
66      g[a].pb(mp(b, i));
67      g[b].pb(mp(a, i));
68    }
69
70    for (int i = 0; i < n; i++){
71      if (dep[i] == -1){
72        dfs(i, 0, i);
73      }
74    }
75    cout << answ.size() << endl;
76
77    for (auto it = answ.begin(); it != answ.end(); it++){
78      cout << *it + 1 << " ";
79    }
80
81  }
```

ПРИЛОЖЕНИЕ Б

Программный код задачи «Точки сочленения»

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <vector>
3  #include <iostream>
4  #include <string>
5  #include <set>
6  #include <map>
7  #include <fstream>
8  #include <iomanip>
9  #include <queue>
```

```cpp
#include <cmath>
#include <algorithm>

using namespace std;

#define mp make_pair
#define pb push_back
#define X first
#define Y second
#define all(a) a.begin(), a.end()

double eps = 1e-7;
long long  INF = 1e9;

int n, m;

vector<vector<int>> g(n, vector<int>());
vector<int> fup(n);
vector<int> dep(n, -1);
set<int> answ;

void dfs(int u, int d, int p){
  fup[u] = dep[u] = d;

  int sons = 0;
  for (int i = 0; i < g[u].size(); i++){
    int v = g[u][i];
    if (v == p){
      continue;
    }
    if (dep[v] >= 0){
      fup[u] = min(fup[u], dep[v]);
    }
    else{
      sons++;
      dfs(v, d + 1, u);
      fup[u] = min(fup[u], fup[v]);
      if (u != p && fup[v] >= d){
        answ.insert(u);
      }
    }
  }

  if (u == p && sons > 1){
    answ.insert(u);
  }
}

```

```
58 int main() {
59 #ifdef _DEBUG
60   freopen("input.txt", "r", stdin);
61   freopen("output.txt", "w", stdout);
62 #endif
63
64   cin >> n >> m;
65
66   g = vector<vector<int>>(n, vector<int>());
67   fup = vector<int> (n);
68   dep = vector<int> (n, -1);
69
70   for (int i = 0; i < m; i++){
71     int a, b;
72     cin >> a >> b;
73     a--, b--;
74     g[a].pb(b);
75     g[b].pb(a);
76   }
77
78   for (int i = 0; i < n; i++){
79     if (dep[i] == -1){
80       dfs(i, 0, i);
81     }
82   }
83   cout << answ.size() << endl;
84
85   for (auto it = answ.begin(); it != answ.end(); it++){
86     cout << *it + 1 << " ";
87   }
88 }
```

ПРИЛОЖЕНИЕ В

Программный код задачи «Конденсация графа (усложнённая версия)»

```
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <vector>
3  #include <iostream>
4  #include <string>
5  #include <set>
6  #include <map>
7  #include <fstream>
8  #include <iomanip>
9  #include <queue>
10 #include <cmath>
11 #include <algorithm>
12
13 using namespace std;
14
```

```
15 #define mp make_pair
16 #define pb push_back
17 #define X first
18 #define Y second
19 #define all(a) a.begin(), a.end()
20
21 double eps = 1e-7;
22 long long  INF = 1e9;
23
24 int n, m;
25
26 vector<vector<int>> g(n, vector<int>());
27 vector<vector<int>> gr(n, vector<int>());
28 vector<int> used(n, -1);
29 vector<int> fo;
30
31 bool f = true;
32 void dfs(int u, int p, int clr){
33   used[u] = clr;
34
35   for (int i = 0; i < g[u].size(); i++){
36     if (!used[g[u][i]]){
37       dfs(g[u][i], u, clr);
38     }
39   }
40
41   fo.pb(u);
42 }
43
44 void dfs2(int u, int i, int clr){
45   used[u] = clr;
46   for (int i = 0; i < gr[u].size(); i++){
47     if (!used[gr[u][i]]){
48       dfs2(gr[u][i], u, clr);
49     }
50   }
51 }
52
53 int main() {
54 #ifdef _DEBUG
55   freopen("input.txt", "r", stdin);
56   freopen("output.txt", "w", stdout);
57 #endif
58
59   cin >> n >> m;
60
61   set<pair<int,int>> was;
62   set<pair<int,int>> wasr;
```

```
63
64      g = vector<vector<int>>(n, vector<int>());
65      gr =  vector<vector<int>>(n, vector<int>());
66      used = vector<int>(n, 0);
67      fo = vector<int>();
68
69      for (int i = 0; i < m; i++){
70        int a, b;
71        cin >> a >> b;
72
73        a--, b--;
74
75        if (a != b && !was.count(mp(a,b))){
76          g[a].pb(b);
77        }
78
79        if (a != b &&  !wasr.count(mp(b,a))){
80          gr[b].pb(a);
81        }
82      }
83
84      for (int i = 0; i < n; i++){
85        if (!used[i]){
86          dfs(i, i, 1);
87        }
88      }
89      used = vector<int>(n, 0);
90
91      reverse(all(fo));
92
93      int clr = 1;
94
95      for (int i = 0; i < n; i++){
96        if (!used[fo[i]]){
97          dfs2(fo[i], i + 1, clr++);
98        }
99      }
100
101     cout << clr - 1 << endl;
102
103     for (int i = 0; i < n; i++){
104       cout << used[i] << " ";
105     }
106 }
```