ФГАОУ ВПО «УрФУ имени первого президента России Б.Н.Ельцина»
Институт математики и компьютерных наук

Кафедра математической экономики

Эйлеровы графы.

Курсовая работа
студента 2 курса группы ФИИТ-201
Садовничей Евгении Александровны

Научный руководитель
Асанов Магаз Оразкимович

Екатеринбург
2014

Эйлеровы графы.

Лемма о рукопожатиях.
Любой конечный неориентированный граф имеет чётное число вершин нечётных степеней.

Эйлеров цикл - это цикл, который содержит каждое ребро графа ровно один раз.

Эйлерова цепь - это маршрут в графе, проходящий по каждому ребру ровно один раз.

Граф эйлеров, если в нем существует эйлеров цикл.

Теорема Эйлера.
Пусть G - связный граф, тогда следующие условия эквивалентны.

1) G - эйлеров граф
2) Степень каждой вершины четна
3) Множество ребер графа можно разбить на циклы

Теорема.
Пусть G = (V, E) - связный граф, содержащий 2l, где l>=1, вершин нечетной степени, тогда каждое минимальное разбиение графа G состоит из l цепей, каждая из которых соединяет две вершины нечетной степени.

Доказательство.
Добавим к графу G вершину z, которую соединим ребрами со всеми вершинами нечетной степени. По теореме Эйлера полученный граф является эйлеровым. В следствии чего в графе имеется эйлеров цикл. Начнем обход этого цикла с верины z. Следующая за z вершина в этом обходе будет являться началом первой цепи. Далее в течении обхода мы вернемся в вершину z. Последняя перед z вершина будет окончанием первой цепи. Таким образом будем выделять цепи, пока не совершим полный обход по циклу. Т.к при выделении каждой новой цепи мы проходим по двум ребрам связанным с z, всего таких пар ребер l, значит цепей тоже l. По построению вершины z ясно, что все цепи начинаются и заканчиваются вершинами нечетной степени.

Докажем, что меньше, чем на l цепей разбить нельзя. Допустим мы разбили граф на какое-то количество цепей. Тогда, если мы возьмем какую-нибудь цепь и удалим все ребра содержащиеся в ней, то изменится четность только первой и последней вершин в цепи. Заметим также, что если мы так удалим все цепи из разбиения, то степень каждой верины в графе станет равной 0. Т.о. нам нужно изменить четность всех нечетных вершин, а для этого каждая нечетная вершина должна являться либо концом, либо началом какой-то цепи. Значит меньше чем 2l/2, т.е. l, цепей быть не может.

Теперь рассмотрим случай, когда l = 0. Тогда граф G - эйлеров, а значит его можно разбить на одну цепь, которая будет являться эйлеровым циклом.

Алгоритм.

Приведем алгоритм нахождения эйлерова цикла в эйлеровом графе (он пригодится и для поиска разбиения графа на цепи).

S - стек
P - ответ

```
S.add(v)
while not S.isEmpty():
  w := S.top()
  if E contains(w, u):
    S.add(u)
    remove(w, u)
  else:
    S.pop()
    P.add(w)
```

Доказательство.

Заметим, что первой в S помещается вершина v, и она будет последней перемещена из S в P. Следовательно, она будет последней вершиной в P. Далее, первый раз, когда обнаружится, что все инцидентные активной вершине ребра пройдены, активной будет стартовая вершина v (Так как степени всех вершин четны). Значит, эта вершина будет первой перемещена из S в P. Итак, по окончании работы алгоритма в начале и в конце последовательности вершин, содержащейся в P, находится вершина v. Иначе говоря, если эта последовательность представляет маршрут, то этот маршрут замкнут.

Покажем, что P это маршрут содержащий все ребра.

Допустим, что в момент окончания работы алгоритма имеются еще не пройденные ребра. Поскольку граф связен, должно существовать хотя бы одно непройденное ребро, инцидентное посещенной вершине. Но тогда эта вершина не могла быть удалена из S, и S не мог стать пустым.

Будем говорить, что ребро (w,u) представлено в S или P, если в какой-то момент работы алгоритма вершины w и u находятся рядом. Каждое ребро графа представлено в S. Допустим в какой-то момент из S в P перемещена вершина w, а следующей в S лежит u. Возможны 2 варианта:

1) На следующем шаге u перемещена в S. Тогда (w,u) представлено в P.

2) Сначала будет пройдена некоторая последовательность ребер, начинающаяся в вершине u. Ввиду четности степеней эта последовательность может закончиться только в вершине u, а значит она следующей попадет в P и (w,u) будет представлено в P.

Отсюда понятно, что последовательность вершин в P является маршрутом и что каждое ребро графа в конечном итоге будет содержаться в этом маршруте, причем один раз.

Этот алгоритм можно применять и в случае, когда l>=1, если принять за первую вершину z. И потом в получившемся массиве P считать z разделителем двух цепей.

Задача.

Пусть помещение банка - это вершина графа, а туннели - это ребра. (v, w) - ребро, если между банками v и w есть туннель.
Получился связный граф, который нужно разбить на минимальное количество цепей.

Код.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <stack>
using namespace std;

vector<vector<int>> arr;
vector<int> ans;
stack<int> s;
int n, m;
int k; // количество вершин с нечетной степенью

void printAns() {
	if (k == 0) {
		cout << 1 << endl;
	}
	else {
		cout << k / 2 << endl;
	}
	for (int i = 0; i < ans.size(); i++) {
		if (ans[i] != n) {
			cout << ans[i] + 1 << " ";
		}
		else {
			cout << endl;
		}
	}
}
void removeEdge(int x, int y) {
	for (int j = 0; j < arr[y].size(); j++) {
		if (arr[y][j] == x) {
			arr[y][j] = -1;
			break;
		}
	}
}
void buildCycle(int v) {
	s.push(v);
	while (!s.empty()) {
		int x = s.top();
```

```
                bool flag = false;
                for (int i = 0; i < arr[x].size(); i++) {
                        if (arr[x][i] == -1) continue;
                        flag = true;
                        s.push(arr[x][i]);
                        removeEdge(x, arr[x][i]);
                        arr[x][i] = -1;
                        break;
                }
                if (flag == false) {
                        ans.push_back(s.top());
                        s.pop();
                }
        }
}
int main()
{
        cin >> n >> m;
        arr.resize(n + 1, vector<int>(0));
        for (int i = 0; i < m; i++) {
                int x, y;
                cin >> x >> y;
                arr[x - 1].push_back(y - 1);
                arr[y - 1].push_back(x - 1);
        }
        for (int i = 0; i < n; i++) {
                if (arr[i].size() % 2 == 1) {
                        k++;
                        arr[n].push_back(i);
                        arr[i].push_back(n);
                }
        }
        if (arr[n].size() == 0) {
                buildCycle(0);
        }
        else {
                buildCycle(n);
        }
        printAns();
}
```