**№1.**

***Теоретические сведения. Описание алгоритма***

**Определение 1. Паросочетание** M в двудольном графе — произвольное множество рёбер двудольного графа, такое что никакие два ребра не имеют общей вершины.

**Определение 2.** Вершины двудольного графа, инцидентные рёбрам паросочетания M, называются **покрытыми**, а неинцидентные — **свободными**.

**Определение 3.** Число рёбер в наибольшем паросочетании графа G называется **числом паросочетания**.

**Определение 4. Максимальное паросочетание** — это такое паросочетание M в графе G, которое не содержится ни в каком другом паросочетании этого графа, то есть к нему невозможно добавить ни одно ребро, которое бы являлось несмежным ко всем рёбрам паросочетания.

**Определение 5. Чередующаяся цепь** — путь в двудольном графе, для любых двух соседних рёбер которого верно, что одно из них принадлежит паросочетанию M, а другое нет.

**Определение 6. Дополняющая цепь (или увеличивающая цепь)** — чередующаяся цепь, у которой оба конца свободны.

**Определение 7. Уменьшающая цепь** — чередующаяся цепь, у которой оба конца покрыты.

***Алгоритм Куна поиска максимального паросочетания***

Задан граф G⟨V,E⟩, про который известно, что он двудольный, но разбиение не задано явно. Требуется найти наибольшее паросочетание в нём

Алгоритм можно описать так: сначала возьмём пустое паросочетание, а потом — пока в графе удаётся найти увеличивающую цепь, — будем выполнять чередование паросочетания вдоль этой цепи, и повторять процесс поиска увеличивающей цепи. Как только такую цепь найти не удалось — процесс останавливаем, — текущее паросочетание и есть максимальное.

В массиве matching хранятся паросочетания (v,matching[v]) (Если паросочетания с вершиной v не существует, то matching[v]=−1). А used — обычный массив "посещённостей" вершин в обходе в глубину (он нужен, чтобы обход в глубину не заходил в одну вершину дважды). Функция dfs возвращает true, если ей удалось найти увеличивающую цепь из вершины v, при этом считается, что эта функция уже произвела чередование паросочетания вдоль найденной цепи.

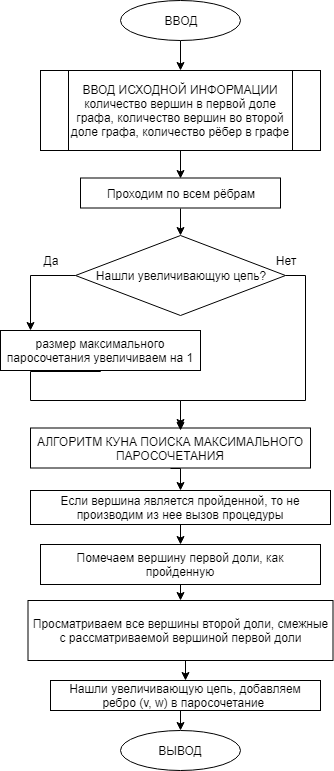
Внутри функции просматриваются все рёбра, исходящие из вершины v, и затем проверяется: если это ребро ведёт в ненасыщенную вершину to, либо если эта вершина to насыщена, но удаётся найти увеличивающую цепь рекурсивным запуском из matching[to], то мы говорим, что мы нашли увеличивающую цепь, и перед возвратом из функции с результатом true производим чередование в текущем ребре: перенаправляем ребро, смежное с to, в вершину v.

В основной программе сначала указывается, что текущее паросочетание — пустое (массив matching заполняется числами −1). Затем перебирается вершина v, и из неё запускается обход в глубину dfs, предварительно обнулив массив used.

Стоит заметить, что размер паросочетания легко получить как число вызовов dfs в основной программе, вернувших результат true. Само искомое максимальное паросочетание содержится в массиве matching. После того, как все вершины v∈V будут просмотрены, текущее паросочетание будет максимальным. Корректность алгоритма следует из теоремы о максимальном паросочетании и дополняющих цепях и теоремы, описанной выше.

***Логическая блок-схема алгоритма***

Основные этапы работы алгоритма представлены на рис. 1 логической блок схемы.



*Рис 1.*

***Оценка сложности алгоритма***

Итак, алгоритм Куна можно представить как серию из n запусков обхода в глубину на всём графе.

Следовательно, всего этот алгоритм исполняется за время O(nm), где m — количество рёбер, что в худшем случае есть O(n3)

Если явно задано разбиение графа на две доли размером n1 и n2, то можно запускать dfs только из вершин первой доли, поэтому весь алгоритм исполняется за время O(n1m). В худшем случае это составляет O(n21n2).

***Код программы.***

#include <cstdlib>

#include <iostream>

#include <vector>

#include <fstream>

using namespace std;

int n1; //Количество вершин в первой доле графа

int n2; //Количество вершин во второй доле графа

int m; //Количество рёбер в графе

vector<int> \*adj; //Список смежности

vector<int> used; //Массив для хранения информации о пройденных и непройденных вершинах

int mtSize = 0; //Размер максимального паросочетания

vector<int> mt; //яМассив для хранения рёбер, образующих максимальное паросочетание

//Алгоритм Куна поиска максимального паросочетания

bool kuhn(int v) {

//Если вершина является пройденной, то не производим из неё вызов процедуры

if (used[v]) {

return false;

}

used[v] = true; //Помечаем вершину первой доли, как пройденную

//Просматриваем все вершины второй доли, смежные с рассматриваемой вершиной первой доли

for (int i = 0; i < adj[v].size(); ++i){

int w = adj[v][i]; //Нашли увеличивающую цепь, добавляем ребро (v, w) в паросочетание

if (mt[w] == -1 || kuhn(mt[w])) {

mt[w] = v; return true;

}

}

return false;

} //Процедура считывания входных данных с консоли

void readData() { //Считываем количество вершин в первой и второй доли и количество рёбер графа

scanf("%d %d %d", &n1, &n2, &m); //Инициализируем список смежности размера n1

adj = new vector<int>[n1];

//Считываем граф, заданный списком рёбер

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

int v, w; scanf("%d %d", &v, &w); v--; w--; //Добавляем ребро (v, w) в граф

adj[v].push\_back(w);

}

used.assign(n1, false); mt.assign(n2, -1);

}

void solve() { //Находим максимальное паросочетание

for (int v = 0; v < n1; ++v) {

used.assign(n1, false); //Если нашли увеличивающую цепь,

// то размер максимального паросочетания увеличиваем на 1

if (kuhn(v)) {

mtSize++;

}

}

}

void printData() {

printf("%d\n", mtSize);

for (int i = 0; i < n2; ++i) {

if (mt[i] != -1) {

printf("%d %d\n", mt[i] + 1, i + 1);

}

}

}

int main() {

readData();

solve();

printData();

return 0;

}

***Тестовый пример***

Ввод:

5 4 8

1 1

2 1

3 1

3 2

4 2

4 3

5 3

5 4

Вывод:

4

1 1

3 2

4 3

5 4