федеральное государственное автономное образовательное учреждение высшего профессионального образования «Казанский (Приволжский) федеральный университет»

Институт информационных технологий и интеллектуальных систем

Семестровая работа

дисциплина “Алгоритмы и структуры данных”

Алгоритм Куна

для поиска максимального паросочетания

Выполнила студентка

1 курса группы 11-002

Мартазова О.А.

Руководитель

Зиятдинов М.Т.

# Краткая историческая справка

Алгоритм Куна был разработан и опубликован Гарольдом Куном в 1955 году. Алгоритм используется для поиска максимального паросочетания в двудольном графе.

Терминология

*Паросочетанием* M называется набор попарно несмежных рёбер графа (иными словами, любой вершине графа должно быть инцидентно не более одного ребра из M).

Все вершины, у которых есть смежное ребро из паросочетания (т.е. которые имеют степень ровно один в подграфе, образованном M), назовём *насыщенными* этим паросочетанием.

Мощностью паросочетания назовём количество рёбер в нём.

*Mаксимальным паросочетанием* назовём паросочетание, мощность которого максимальна среди всех возможных паросочетаний в данном графе.

*Цепью* длины k назовём некоторый простой путь (т.е. не содержащий повторяющихся вершин или рёбер), содержащий ровно k рёбер.

*Чередующейся цепью* относительно некоторого паросочетания назовем простой путь длины k, в которой рёбра поочередно принадлежат/не принадлежат паросочетанию.

*Увеличивающей цепью* относительно некоторого паросочетания назовём чередующуюся цепь, у которой начальная и конечная вершины не принадлежат паросочетанию.

# Основной принцип устройства

Задача. Пусть есть L мальчиков и R девочек. Про каждого мальчика и про каждую девочку известно, с кем они не против танцевать. Нужно составить как можно больше пар, в которых партнёры хотят танцевать друг с другом. Формализуем эту задачу, представив мальчиков и девочек как вершины в двудольном графе, ребрами которого будет отношение «могут танцевать вместе».

Иными словами, задан двудольный граф, требуется найти максимальное паросочетание в нем.

Алгоритм можно описать так: сначала возьмём пустое паросочетание, а потом — пока в графе удаётся найти увеличивающую цепь, — будем выполнять чередование (убираем из паросочетания все ребра, принадлежащие цепи, и добавляем все остальные) 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 (вектор) matching хранятся паросочетания (v,matching[v]). v∈R, matching[v]∈L. Если паросочетания с вершиной v не существует, то matching[v] = −1.

Поиск увеличивающей цепи осуществляется за счет dfs (depth-first search) обхода в глубину. В коде эта функция называется try\_kuhn().

Used — массив "посещённостей" вершин в обходе в глубину (он нужен, чтобы обход в глубину не заходил в одну вершину дважды).

Если ребро из вершины v ведёт в ненасыщенную вершину to, либо если эта вершина to насыщена, но удаётся найти увеличивающую цепь рекурсивным запуском из matching[to], то try\_kuhn() возвращает true, ей удалось найти увеличивающую цепь из вершины v, при этом считается, что эта функция уже произвела чередование паросочетания вдоль найденной цепи в текущем ребре: ребро, смежное с to, перенаправляется в вершину v.

В массиве used\_first на i-той позиции хранится true, если из i-той вершины существует паросочетание. В противном случае хранится false.

В основной программе перебирается вершина v∈L, и, если эта вершина не покрыта паросочетанием при заполнении used\_first, из неё запускается try\_kuhn(), предварительно обнулив массив used.

Стоит заметить, что размер паросочетания легко получить как число вызовов try\_kuhn() в основной программе, вернувших результат true. Размер паросочетания считается в переменной mtSize.После того, как все вершины l∈L будут просмотрены, текущее паросочетание будет максимальным.

Корректность алгоритма Куна основывается на теореме Бержа.

≪Паросочетание является максимальным тогда и только тогда, когда не существует увеличивающих относительно него цепей.≫

# Оценка временной сложности

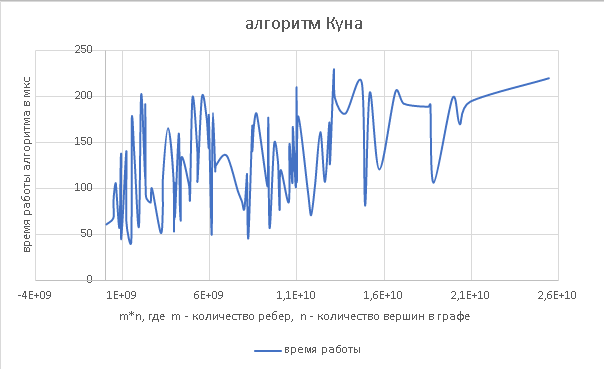
Алгоритм Куна ровно n раз ищет увеличивающий путь, каждый раз просматривая не более m рёбер, а значит работает за O(nm), где n = L+R ー количество вершин в двудольном графе.

В худшем случае сложность O(), потому что максимальное количество рёбер в графе m = n\*(n-1)/2.

Если явно задано разбиение графа на две доли размером L и R, то можно запускать обход в глубину только из вершин первой доли, поэтому весь алгоритм выполняется за время O(L\*m). В худшем случае это составляет O().

# Графики

График зависимости времени выполнения алгоритма (в мкс) от входных данных:



Данные, изображенные на графике, отличаются от теоретических (прослеживаются перепады) из-за случайно заданных значений двудольного графа, а также из-за сторонних процессов компьютера.

# Вывод

Построение максимального паросочетания — это задача оптимизации.

Преимущества алгоритма:

Нахождение максимального паросочетания в двудольном графе с асимптотической сложностью O(m\*n).

Недостатки алгоритма:

Существуют алгоритмы построения максимального паросочетания в произвольном графе с большей асимптотической сложностью и в двудольном графе с меньшей.

Моя реализация алгоритма Куна предназначена только для явно заданного двудольного графа.

Применение:

Алгоритм подходит для решения некоторых задач оптимизации.

# Список используемой литературы (источников)

[паросочетания - Алгоритмистика](https://algorithmica.org/ru/matching)

[алгоритм Куна](https://e-maxx.ru/algo/kuhn_matching)

[вики-конспекты ИТМО Алгоритм Куна](https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D1%83%D0%BD%D0%B0_%D0%B4%D0%BB%D1%8F_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0_%D0%BC%D0%B0%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BF%D0%B0%D1%80%D0%BE%D1%81%D0%BE%D1%87%D0%B5%D1%82%D0%B0%D0%BD%D0%B8%D1%8F#.D0.90.D0.BB.D0.B3.D0.BE.D1.80.D0.B8.D1.82.D0.BC)

[вики-конспекты ИТМО Терминология](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9F%D0%B0%D1%80%D0%BE%D1%81%D0%BE%D1%87%D0%B5%D1%82%D0%B0%D0%BD%D0%B8%D1%8F:_%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5_%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F,_%D1%82%D0%B5%D0%BE%D1%80%D0%B5%D0%BC%D0%B0_%D0%BE_%D0%BC%D0%B0%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC_%D0%BF%D0%B0%D1%80%D0%BE%D1%81%D0%BE%D1%87%D0%B5%D1%82%D0%B0%D0%BD%D0%B8%D0%B8_%D0%B8_%D0%B4%D0%BE%D0%BF%D0%BE%D0%BB%D0%BD%D1%8F%D1%8E%D1%89%D0%B8%D1%85_%D1%86%D0%B5%D0%BF%D1%8F%D1%85)

[паросочетание Википедия](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%BE%D1%81%D0%BE%D1%87%D0%B5%D1%82%D0%B0%D0%BD%D0%B8%D0%B5)

[паросочетание](https://pro-prof.com/forums/topic/graph_max_matching)

# Приложение

# Код программы

Задача:

ввести число L девочек,

ввести число R мальчиков.

Для каждой девочки ввести количество мальчиков, с которыми она не против танцевать. (tmp)

Ввести номера вершин "мальчиков", которых выбрала девочка.

#include <iostream>

#include <vector>

using namespace std;

int l, r, m;

const int N = 1000;

vector<int>g[N]; *//bipartite graph with edges only from the first part to the second*

vector<int> matching; *//which vertex is matched with the vertex of the second part*

vector<bool> used;

int mtSize = 0;

bool try\_kuhn(int from) {

if (used[from])

return false;

used[from] = true;

for (int to : g[from]) {

if (matching[to] == -1 || try\_kuhn(matching[to])) {

*// if vertex is single, matched*

*// if vertex has matched, then you can match it*

*// only when you can find some other vertex from its current vertex mt[to]*

matching[to] = from;

return true;

}

}

return false;

}

void fill() {

cout << "Enter the number of vertices in the first part" << endl;

cin >> l;

cout << "Enter the number of vertices in the second part" << endl;

cin >> r;

cout << "Enter adjacency lists of bipartite graph" << endl << endl;

cout << "for each vertex in the first part enter" << endl;

cout << "the number of adjacent vertices" << endl;

cout << "then enter adjacent vertices" << endl;

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

int tmp;

cin >> tmp;

m += tmp;

for (int j = 0; j < tmp; j++) {

int v;

cin >> v;

v--;

g[i].push\_back(v);

}

}

*//no match yet = -1*

matching.assign(r+l, -1);

}

void solve() {

vector<bool> used\_first(l, false);

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

for (int to : g[v]) {

if (matching[to] == -1) {

*//add matching*

matching[to] = v;

mtSize++;

used\_first[v] = true;

break;

}

}

}

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

if (used\_first[v]) continue;

*//start depth-first search*

used.assign(l, false);

if (try\_kuhn(v)) mtSize ++;

}

}

void print() {

cout << "maximal matching:" << endl;

cout << mtSize << endl;

cout << "matched edges:" << endl;

for (int i = l; i < r+l; ++i)

if (matching[i] != -1)

cout << matching[i] + 1 << " - " << i + 1 << endl;

}

int main() {

fill();

solve();

print();

}

Ссылка на репозиторий [github](https://github.com/OlgaMartazova/algorithms_2semester/tree/main/semestrovka1)

# Инструкция по запуску тестов

[README.md](https://github.com/OlgaMartazova/algorithms_2semester/blob/main/semestrovka2/README.md)

# Таблицы полученных значений времени работы в зависимости от размера данных

[Kuhn\_algorithm.xlsx](https://github.com/OlgaMartazova/algorithms_2semester/blob/main/semestrovka2/graphs/Kuhn_algorithm.xlsx)