9. Алгоритм Куна поиска наибольшего паросочетания в двудольном графе

**Двудо́льный граф** — это граф, множество вершин которого можно разбить на две части таким образом, что каждое ребро графа соединяет какую-то вершину из одной части с какой-то вершиной другой части, то есть не существует ребра, соединяющего две вершины из одной и той же части.

**Паросочетанием** M называется набор попарно несмежных рёбер графа (иными словами, любой вершине графа должно быть инцидентно не более одного ребра из множества M). Мощностью паросочетания назовём количество рёбер в нём. Наибольшим (или максимальным) паросочетанием назовём паросочетание, мощность которого максимальна среди всех возможных паросочетаний в данном графе. Все те вершины, у которых есть смежное ребро из паросочетания (т.е. которые имеют степень ровно один в подграфе, образованном M), назовём насыщенными этим паросочетанием.

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

**Чередующейся цепью** (в двудольном графе, относительно некоторого паросочетания) назовём цепь, в которой рёбра поочередно принадлежат/не принадлежат паросочетанию.

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

**Теорема Бержа**

**Формулировка**. Паросочетание является максимальным тогда и только тогда, когда не существует увеличивающих относительно него цепей.

### Алгоритм Куна

Алгоритм Куна — непосредственное применение теоремы Бержа. Его можно кратко 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.

Осталось детализировать способ нахождения увеличивающих цепей. **Алгоритм Куна** — просто ищет любую из таких цепей с помощью [**обхода в глубину**](http://e-maxx.ru/algo/dfs)или [**в ширину**](http://e-maxx.ru/algo/bfs). Алгоритм Куна просматривает все вершины графа по очереди, запуская из каждой обход, пытающийся найти увеличивающую цепь, начинающуюся в этой вершине.

Удобнее описывать этот алгоритм, считая, что граф уже разбит на две доли (хотя на самом деле алгоритм можно реализовать и так, чтобы ему не давался на вход граф, явно разбитый на две доли).

Алгоритм просматривает все вершины v первой доли графа: v = 1 \ldots n_1. Если текущая вершина v уже насыщена текущим паросочетанием (т.е. уже выбрано какое-то смежное ей ребро), то эту вершину пропускаем. Иначе — алгоритм пытается насытить эту вершину, для чего запускается поиск увеличивающей цепи, начинающейся с этой вершины.

Поиск увеличивающей цепи осуществляется с помощью специального обхода в глубину или ширину (обычно в целях простоты реализации используют именно обход в глубину). Изначально обход в глубину стоит в текущей ненасыщенной вершине v первой доли. Просматриваем все рёбра из этой вершины, пусть текущее ребро — это ребро (v,to). Если вершина to ещё не насыщена паросочетанием, то, значит, мы смогли найти увеличивающую цепь: она состоит из единственного ребра (v,to); в таком случае просто включаем это ребро в паросочетание и прекращаем поиск увеличивающей цепи из вершины v. Иначе, — если to уже насыщена каким-то ребром (p,to), то попытаемся пройти вдоль этого ребра: тем самым мы попробуем найти увеличивающую цепь, проходящую через рёбра (v,to), (to,p). Для этого просто перейдём в нашем обходе в вершину p — теперь мы уже пробуем найти увеличивающую цепь из этой вершины.

Можно понять, что в результате этот обход, запущенный из вершины v, либо найдёт увеличивающую цепь, и тем самым насытит вершину v, либо же такой увеличивающей цепи не найдёт (и, следовательно, эта вершина v уже не сможет стать насыщенной).

После того, как все вершины v = 1 \ldots n_1 будут просмотрены, текущее паросочетание будет максимальным.

## Реализация

Приведём здесь реализацию вышеописанного алгоритма, основанную на обходе в глубину, и принимающей двудольный граф в виде явно разбитого на две доли графа. Эта реализация весьма лаконична, и, возможно, её стоит запомнить именно в таком виде.

Здесь n — число вершин в первой доле, k — во второй доле, g[v] — список рёбер из вершины v первой доли (т.е. список номеров вершин, в которые ведут эти рёбра из v). Вершины в обеих долях занумерованы независимо, т.е. первая доля — с номерами 1 \ldots n, вторая — с номерами 1 \ldots k.

Дальше идут два вспомогательных массива: \rm mt и \rm used. Первый — \rm mt — содержит в себе информацию о текущем паросочетании. Для удобства программирования, информация эта содержится только для вершин второй доли: mt[i] — это номер вершины первой доли, связанной ребром с вершиной iвторой доли (или -1, если никакого ребра паросочетания из i не выходит). Второй массив — \rm used — обычный массив "посещённостей" вершин в обходе в глубину (он нужен, просто чтобы обход в глубину не заходил в одну вершину дважды).

Функция \rm try\_kuhn — и есть обход в глубину. Она возвращает \rm 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 первой доли, 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 \rm mt[to], то мы говорим, что мы нашли увеличивающую цепь, и перед возвратом из функции с результатом \rm true производим чередование в текущем ребре: перенаправляем ребро, смежное с to, в вершину v.

В основной программе сначала указывается, что текущее паросочетание — пустое (список \rm mt заполняется числами -1). Затем перебирается вершина vпервой доли, и из неё запускается обход в глубину \rm try\_kuhn, предварительно обнулив массив \rm used.

Стоит заметить, что размер паросочетания легко получить как число вызовов \rm try\_kuhn в основной программе, вернувших результат \rm true. Само искомое максимальное паросочетание содержится в массиве \rm mt.

int n, k;

vector < vector<int> > g;

vector<int> mt;

vector<char> used;

bool try\_kuhn (int v) {

if (used[v]) return **false**;

used[v] = **true**;

for (size\_t i=0; i<g[v].size(); ++i) {

int to = g[v][i];

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

mt[to] = v;

return **true**;

}

}

return **false**;

}

int main() {

... чтение графа ...

mt.assign (k, -1);

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

used.assign (n, **false**);

try\_kuhn (v);

}

for (int i=0; i<k; ++i)

if (mt[i] != -1)

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

}

Ещё раз повторим, что алгоритм Куна легко реализовать и так, чтобы он работал на графах, про которые известно, что они двудольные, но явное их разбиение на две доли не найдено. В этом случае придётся отказаться от удобного разбиения на две доли, и всю информацию хранить для всех вершин графа. Для этого массив списков g теперь задаётся не только для вершин первой доли, а для всех вершин графа (понятно, теперь вершины обеих долей занумерованы в общей нумерации — от 1 до n). Массивы \rm mt и \rm used теперь также определены для вершин обеих долей, и, соответственно, их нужно поддерживать в этом состоянии.