Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №9 по курсу «Дискретный анализ»

Студент:	А. О. Дубинин
Преподаватель:	А. А. Кухтичев
Группа:	М8О-206Б
Дата:	
Оценка:	
Подпись:	

Москва, 2019

Лабораторная работа №8

Задача: Разработать программу на языке C или C++, реализующую указанный алгоритм.

Вариант 8 (Поиск максимального паросочетания алгоритмом Куна): Задан неориентированный двудольный граф, состоящий из n вершин и m ребер. Вершины пронумерованы целыми числами от 1 до n. Необходимо найти максимальное паросочетание в графе алгоритмом Куна. Для обеспечения однозначности ответа списки смежности графа следует предварительно отсортировать. Граф не содержит петель и кратных ребер.

Входные данные: В первой строке заданы 1 <= n <= 110000 и 1 <= m <= 40000. В следующих m строках записаны ребра. Каждая строка содержит пару чисел – номера вершин, соединенных ребром.

Выходные данные:

В первой строке следует вывести число ребер в найденном паросочетании. В следующих строках нужно вывести сами ребра, по одному в строке. Каждое ребро представляется парой чисел – номерами соответствующих вершин. Строки должны быть отсортированы по минимальному номеру вершины на ребре. Пары чисел в одной строке также должны быть отсортированы.

Пример:

Входной файл	Выходной файл
4 3	2
1 2	1 2
2 3	3 4
3 4	

1 Описание

Определение: Паросочетание (англ. matching) M в двудольном графе — произвольное множество рёбер двудольного графа, такое что никакие два ребра не имеют общей вершины.

Определение: Максимальное паросочетание (англ. maximal matching) — 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. Другими словами, паросочетание M графа G является максимальным, если любое ребро в G имеет непустое пересечение по крайней мере с одним ребром из M.

Определение: Чередующаяся цепь (англ. alternating path) — путь в двудольном графе, для любых двух соседних рёбер которого верно, что одно из них принадлежит паросочетанию M, а другое нет.

Определение: Дополняющая цепь (или увеличивающая цепь) (англ. augmenting path) — чередующаяся цепь, у которой оба конца свободны.

Теорема: Паросочетание M в двудольном графе G является максимальным тогда и только тогда, когда в G нет дополняющей цепи.

Согласно[2], 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.

Осталось детализировать способ нахождения увеличивающих цепей. Алгоритм Куна — просто ищет любую из таких цепей с помощью обхода в глубину. Алгоритм Куна просматривает все вершины графа по очереди, запуская из каждой обход, пытающийся найти увеличивающую цепь, начинающуюся в этой вершине.

Разобьем сначала наш граф явно на две доли, с помощью покраски графа в два цвета. И будем описывать далее наш алгоритм, считая, что он разбит 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. 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.

Итак, алгоритм Куна можно представить как серию из n запусков обхода в глубину на всём графе. Следовательно, всего этот алгоритм исполняется за время $O(nm)$, что в худшем случае есть $O(n^3)$.

2 Исходный код

main.cpp

```cpp
#include <iostream>
#include <vector>
#include <set>
#include <queue>
#include <algorithm>


bool Dfs(std::vector<char> &used, std::vector<int> &matching, const std::vector<std::set<int>> &adjList, int v) {
    if (used[v])
        return false;
    used[v] = true;

    for (auto to : adjList[v]) {
        if (matching[to] == -1 || Dfs(used, matching, adjList, matching[to])) {
            matching[to] = v;
            return true;
        }
    }

    return false;
}

void Kuhn(const int &n, const std::vector<std::set<int>> &adjList, const std::vector<char> &part) {

    std::vector<int> matching(n + 1, -1);
    std::vector<char> used;
    for (int v = 1; v <= n; ++v) {
        if (part[v] == 0) {
            used.assign(n + 1, false);
            Dfs(used, matching, adjList, v);
        }

    }

    int u, v;
    std::set<std::pair<int, int>> ans;
    for (int i = 1; i <= n; ++i) {
        if (matching[i] != -1) {
            u = std::min(matching[i], i);
            v = std::max(matching[i], i);
            ans.insert(std::make_pair(u, v));
        }
    }

```

```cpp
    std::cout << ans.size() << std::endl;

    for (auto i : ans) {
        std::cout << i.first << " " << i.second << "\n";
    }

}

void Bfs(const std::vector<std::set<int>> &adjList, std::vector<char> &color, const
    int &src) {

    std::queue<int> q;
    q.push(src);

    // Assign first color to source
    color[src] = 0;

    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (auto v : adjList[u]) {
            if (color[v] == -1) {
                color[v] = !color[u];
                q.push(v);
            }
        }
    }
}

std::vector<char> DivideToBipartite(const std::vector<std::set<int>> &adjList, const
    int &n) {
    // -1 = no color
    // 0 = first color
    // 1 = second color
    std::vector<char> color(n + 1, -1);

    for (int i = 1; i <= n; ++i) {
        if (color[i] == -1)
            Bfs(adjList, color, i);
    }

    return color;

}

int main() {

    int n, m, u, v;
```

```cpp
    std::cin >> n >> m;

    std::vector<std::set<int>> adjList(n + 1);

    for (int i = 0; i < m; ++i) {
        std::cin >> u >> v;
        adjList[u].insert(v);
        adjList[v].insert(u);
    }


    std::vector<char> color = DivideToBipartite(adjList, n);

    Kuhn(n, adjList, color);

    return 0;
}
```

3 Консоль

```
art@mars:~/study/DA/labs/lab_9$ ./main
4 3
1 2
2 3
3 4
2
1 2
3 4
```

4 Выводы

Графы обширно применяются в современном мире: прокладывание дорог и маршрутов, и вообще повсеместно в логистике. Также частный случай графа – дерево используется при хранении данных, в различных сервисах в интернете, в программировании. Графы очень удобны, поскольку реальность прекрасно отображается на них: есть конкретные пункты и есть пути между ними. Поэтому алгоритм нахождения максимального паросочитания является важным алгоритмом, например с помощью него можно решить задачу №1683. Распределение заказов на сайте informatics.mccme.ru.

В процессе обучения графы встречаются мне уже не в первый раз: сперва решение задач по дискретной математике, далее решение логических задач и разбор генеалогического дерева в логическом программировании. Полагаю, что и в дальнейшем мне предстоит с ними работать, поэтому полученный опыт найдёт своё применение.

Список литературы

[1] Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн *Алгоритмы: построение и анализ, 3-е изд.* — Издательский дом « Вильямс», 2013. Перевод с английского: ООО «И. Д. Вильямс». — 1328 с. (ISBN 978-5-8459-1794-2 (рус.))

[2] *Emaxx - Алгоритм Куна*
URL: `https://e-maxx.ru/algo/kuhn_matching`

[3] *Итмо – Алгоритм Куна для поиска максимального паросочетания*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Куна_для_поиска_максимального_паросочетания`