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:	М. А. Инютин
Преподаватель:	А. А. Кухтичев
Группа:	М8О-207Б-19
Дата:	
Оценка:	
Подпись:	

Москва, 2021

Лабораторная работа №9

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

Задан неориентированный граф, состоящий из n вершин и m ребер. Вершины пронумерованы целыми числами от 1 до n. Необходимо вывести все компоненты связности данного графа.

1 Описание

Требуется реализовать один из алгоритмов поиска компонент связности в неориентированном графе.

Как сказано в [3], это можно сделать как обходом в ширину [2], так и обходом в глубину [1]. В лабораторной работе я выбрал обход в глубину.

Суть обхода в глубину заключается в рекурсивном посещении всех соседей текущей вершины графа. Для каждой вершины необходимо хранить, посещена ли она или нет. В противном случае обход будет бесконечно возвращаться обратно или зависать на цикличном графе.

В ходе обхода будет посещена каждая вершина в компоненте и обход попробует перейти по каждому ребру. Так сложность алгоритма $O(|V| + |E|)$, где V — множество вершин в компоненте связности, а E — множество рёбер в компоненте.

Если граф несвязный (две и более компонент связности), то требуется выполнить обход в каждой компоненте. Пусть мы начали с какой-то компоненты. Тогда мы пометили все вершины в ней и обход в глубину не совершит рекурсивных вызовов для вершин в ней. Поэтому достаточно запустить обход в глубину из каждой вершины графа.

Независимо от того, связный граф или нет, итоговая сложность $O(n + m)$ времени и $O(n)$ дополнительной памяти. Учитывая время на сортировку компонент, получим временную сложность $O(n * \log(n) + m)$.

2 Исходный код

Для эффективного хранения графа используются списки смежности — список вершин, смежных с текущей. Так как граф неориентированный, то следует добавляет каждое ребро в два списка.

visited хранит для каждой вершины, была ли она посещена во время обхода графа.

Функция DFS возвращает истину, если она нашла новую компоненту и ложь в противном случае. Это нужно для расширения вектора под новую компоненту связности.

```
#include <algorithm>
#include <iostream>
#include <vector>

using TVecVecInt = std::vector< std::vector<int> >;

bool DFS(int u, const TVecVecInt & g, std::vector<bool> & visited, TVecVecInt & res) {
    if (visited[u]) {
        return false;
    }
    visited[u] = true;
    res.back().push_back(u);
    for (size_t i = 0; i < g[u].size(); ++i) {
        int v = g[u][i];
        DFS(v, g, visited, res);
    }
    return true;
}

int main() {
    int n, m;
    std::cin >> n >> m;
    TVecVecInt g(n);
    for (int i = 0; i < m; ++i) {
        int u, v;
        std::cin >> u >> v;
        --u;
        --v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    std::vector<bool> visited(n);
    TVecVecInt res;
    bool flagNewConnectivity = true;
    for (int i = 0; i < n; ++i) {
        if (flagNewConnectivity) {
            res.push_back(std::vector<int>());
        }
        flagNewConnectivity = DFS(i, g, visited, res);
```

```
    }
    if (!flagNewConnectivity) {
        res.pop_back();
    }
    for (size_t i = 0; i < res.size(); ++i) {
        std::sort(res[i].begin(), res[i].end());
    }
    for (size_t i = 0; i < res.size(); ++i) {
        for (size_t j = 0; j < res[i].size(); ++j) {
            std::cout << res[i][j] + 1 << ' ';
        }
        std::cout << '\n';
    }
}
```

3 Консоль

```
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ make
g++ -g -O2 -pedantic -std=c++17 -Wall -Wextra -Werror main.cpp -o solution
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ cat tests/1.in
5 4
1 2
2 3
1 3
4 5
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ ./solution <tests/1.in
1 2 3
4 5
```

4 Тест производительности

В тесте производительности сравниваются два алгоритма обхода графа: в глубину и в ширину.

Тесты состоят из графов на 10^3, 10^4 и 10^5 вершин для несвязных (число рёбер на порядок меньше числа вершин) и плотных графов (число рёбер на порядок больше числа вершин).

Обход в глубину:

```
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ make dfs
g++ -g -O2 -pedantic -std=c++17 -Wall -Wextra -Werror dfs.cpp -o dfs
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ ./dfs <tests/nodes3edges2.in
DFS time is 0.55 ms
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ ./dfs <tests/nodes3edges4.in
DFS time is 0.104 ms
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ ./dfs <tests/nodes4edges3.in
DFS time is 0.508 ms
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ ./dfs <tests/nodes4edges5.in
DFS time is 1.533 ms
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ ./dfs <tests/nodes5edges4.in
DFS time is 5.101 ms
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ ./dfs <tests/nodes5edges6.in
DFS time is 22.915 ms
```

Обход в ширину:

```
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ make bfs
g++ -g -O2 -pedantic -std=c++17 -Wall -Wextra -Werror bfs.cpp -o bfs
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ ./bfs <tests/nodes3edges2.in
BFS time is 0.64 ms
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ ./bfs <tests/nodes3edges4.in
BFS time is 0.921 ms
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ ./bfs <tests/nodes4edges3.in
BFS time is 0.657 ms
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ ./bfs <tests/nodes4edges5.in
BFS time is 12.335 ms
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ ./bfs <tests/nodes5edges4.in
BFS time is 7.488 ms
engineerxl@engineerxl-GF63-Thin-9RCX:~/Study/DA/lab9$ ./bfs <tests/nodes5edges6.in
BFS time is 178.615 ms
```

Видно, что обход в ширину на порядок проигрывает обходу в глубину на плотных графах, так как требует большого числа операций с очередью.

5 Выводы

Выполнив девятую лабораторную работу по курсу «Дискретный анализ», я изучил способы представления графа в компьютере и базовые алгоритмы на графах: обход графа, поиск кратчайших путей, максимального паросочетания и потока в графе.

Мы сталкиваемся с графами каждый день, начиная дорогой от дома до института (кратчайший путь) и заканчивая вопросом, что надеть на встречу (паросочетание). Поэтому графы и связанные с ними задачи максимально близки к жизни и требуют эффективных алгоритмов решения этих задач.

Одна из задач тысячелетия — равенство классов P и NP тесно связано с теорией графов. Для многих задач ещё не найдены эффективные алгортимы, которые могли бы сильно упростить нашу жизнь.

Список литературы

[1] *MAXimal :: algo :: Поиск в глубину — e-maxx.ru*
URL: `https://e-maxx.ru/algo/dfs` (дата обращения: 30.04.2021).

[2] *MAXimal :: algo :: Поиск в ширину — e-maxx.ru*
URL: `https://e-maxx.ru/algo/bfs` (дата обращения: 30.04.2021).

[3] *MAXimal :: algo :: Алгоритм поиска компонент связности в графе — e-maxx.ru*
URL: `https://e-maxx.ru/algo/connected_components` (дата обращения: 30.04.2021).