Национальный исследовательский университет
"Московский авиационный институт"
Факультет No8 "Информационные технологии и прикладная математика"
Кафедра 806 "Вычислительная математика и программирование"

# ЛАБОРАТОРНАЯ РАБОТА №9
# ПО КУРСУ “ДИСКРЕТНЫЙ АНАЛИЗ”
# 4 СЕМЕСТР

Выполнил студент: Поляков А.И.
Группа: М80-208Б-19
Оценка:
Подпись:

Москва 2022

# Лабораторная работа №9

**Вариант № 3 - Поиск компонент связности**

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.

**Формат входных данных**

В первой строке заданы $1 \leq n \leq 105$ и $1 \leq m \leq 105$. В следующих m строках записаны ребра. Каждая строка содержит пару чисел – номера вершин, соединенных ребром.

**Формат результата**

Каждую компоненту связности нужно выводить в отдельной строке, в виде списка номеров вершин через пробел. Строки при выводе должны быть отсортированы по минимальному номеру вершины в компоненте, числа в одной строке также должны быть отсортированы.

# Описание

Для решения можно воспользоваться как обходом в глубину, так и обходом в ширину.

Фактически, мы будем производить серию обходов: сначала запустим обход из первой вершины, и все вершины, которые он при этом обошёл — образуют первую компоненту связности. Затем найдём первую из оставшихся вершин, которые ещё не были посещены, и запустим обход из неё, найдя тем самым вторую компоненту связности. И так далее, пока все вершины не станут помеченными.

Итоговая асимптотика составит $O(n + m)$: в самом деле, такой алгоритм не будет запускаться от одной и той же вершины дважды, а, значит, каждое ребро будет просмотрено ровно два раза (с одного конца и с другого конца).

# Исходный код

Код программы состоит из трех файлов. *main.cpp* - основной файл программы, *graph.hpp* - заголовочный файл, содержащий описание структуры класса, и *graph.cpp* - содержащий реализацию функций.

Структура графа выглядит следующим образом:

```
class TGraph {
private:
    std::set<int> *Edges;
    bool *Checked;
    std::vector<int> ConnectivityComponent;
public:
    TGraph(int vertices);
    void AddConnection(int v1, int v2);
    void DFS(int vertex);
    void PrintComponent();
    void RemoveCurrentComponent();
    bool IsTraversed(int v);
    ~TGraph();
};
```

Метод DFS() совершает обход в глубину, начиная с указанной вершины, добавляя найденные вершины в компонент связности (поле объекта класса).

```
void TGraph::DFS(int vertex) {
    Checked[vertex] = true;
    ConnectivityComponent.push_back(vertex);
    for (std::set<int>::iterator iter = Edges[vertex].begin(); iter != Edges
    [vertex].end(); iter++) {
        if (Checked[(*iter)] == false) {
            DFS(*iter);
        }
    }
}
```

Так же добавим реализацию добавления связей в графе, вывода нынешнего компонента связности, а так же геттер для проверки, была ли посещена данная вершина.

```
void TGraph::AddConnection(int v1, int v2) {
    Edges[v1].insert(v2);
    Edges[v2].insert(v1);
}


void TGraph::PrintComponent() {
    std::sort(ConnectivityComponent.begin(), ConnectivityComponent.end());
    for (size_t i = 0; i < ConnectivityComponent.size(); i++) {
        std::cout << ConnectivityComponent[i] << " ";
    }
    std::cout << "\n";
}

void TGraph::RemoveCurrentComponent() {
    ConnectivityComponent.clear();
}

bool TGraph::IsTraversed(int v) {
```

```
    return Checked[v];
}
```

Точка входа программы выглядит так:

```
int main() {
    int vertices;
    int edges;
    std::cin >> vertices >> edges;
    TGraph graph(vertices);
    int vertex_1, vertex_2;
    for (int i = 0; i < edges; i++) {
        std::cin >> vertex_1 >> vertex_2;
        graph.AddConnection(vertex_1, vertex_2);
    }
    for (int i = 1; i < vertices + 1; i++) {
        if (graph.IsTraversed(i) == false) {
            graph.DFS(i);
            graph.PrintComponent();
            graph.RemoveCurrentComponent();
        }
    }
    return 0;
}
```

# Тест производительности

**Консольный вывод**

```
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab9/solution$ make
make: Nothing to be done for 'all'.
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab9/solution$
./solution < ../test1.txt
1 2 3
4 5
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab9/solution$
./solution < ../test2.txt
1
2
3
4
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab9/solution$
./solution < ../test3.txt
1 2 3
4
5
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab9/solution$
./solution < ../test4.txt
1 3 4
2
5
6
```

**Тесты** Проведем тест производительности. Сгенерируем тесты размеров 10, 20, 100, 1000 входных данных.

```
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab9/bench$ python3 test
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab9/bench$ ./bench < in
Elapsed in: 0.000208 sec.
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab9/bench$ ./bench < in
Elapsed in: 0.000398 sec.
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab9/bench$ ./bench < in
Elapsed in: 0.002285 sec.
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab9/bench$ ./bench < in
Elapsed in: 0.019934 sec.
```

Можно заметить, что время выполнения увеличивается пропорционально увеличению количества входных данных. Это можно объяснить тем, что сложность алгоритма $O(n + m)$.

# Вывод

После выполнения данной лабораторной работы по дисциплине "Дискретный анализ"я узнал, как можно эффективно находить компоненты связности в неориентированных графах.

Данный алгоритм может пригодиться как в решении олимпиадных задач, так и в для реальных условиях, например в социальных сетях, при поиске общих возможных знакомых, ведь можно составить граф так называемых "рукопожатий и находить компании, предлагать возможных друзей и многое другое.

# Литература

[1] Классические алгоритмы поиска образца в строке, Андрей Калинин. 28 октября 2011 г.

[2] Дэн Гасфилд. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология. — Издательский дом «Невский Диалект», 200ц. Перевод с английского: И. В. Романовского. — 654с. (ISBN 5-7940-0103-8 (рус.))