Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа № 9 по курсу дискретного анализа: Графы

Студент: А. О. Тояков
Преподаватель: А. Н. Ридли
Группа: М8О-307Б-18
Дата:
Оценка:
Подпись:

Москва
2021

Условие

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.

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.

Метод решения

Для решения этой задачи нужно будет реализовать поиск в глубину. Граф будем представлять в виде пар смежных вершин. Будем проходить по всем вершинам и добавлять их в ответ, если они ещё не были пройдены (так называемые белые вершины; после добавления сразу же перекрашиваем их в серый или чёрный). Если же поиск зайдёт в тупик, значит все компоненты связности для определённой части графа собраны и можно переходить к следующим вершинам.

Описание программы

Для хранения вершин я создал структуру TNode, в которой хранится цвет вершины (white, если вершина ещё не пройдена и grey или black иначе), её индекс, а также вектор, состоящий из индексов связанных с ней вершин. В итоге представление графа реализовано, как вектор G <TNode>, в котором мы храним все связи в нашем графе.

Поиск осуществляется с помощью функции DFS, в которой мы создаём вектор answer, куда мы будем заносить компоненты связности и поочереди их выводить. Мы проходим по всем индексам вектора G, и если наша текущая вершина белая, то заносим её в answer и запускаем функцию DfsVisit, которая пройдёт по всем вершинам связанным с текущей и добавит их в ответ. Когда функция зайдёт в тупик мы получим нашу полную текущую компоненту связности. Затем элементы в векторе answer отсортируются с помощью std::sort и распечатаются. Так мы будем проходить по всему вектору G, пока не выведем все компоненты связности.

Исходный код

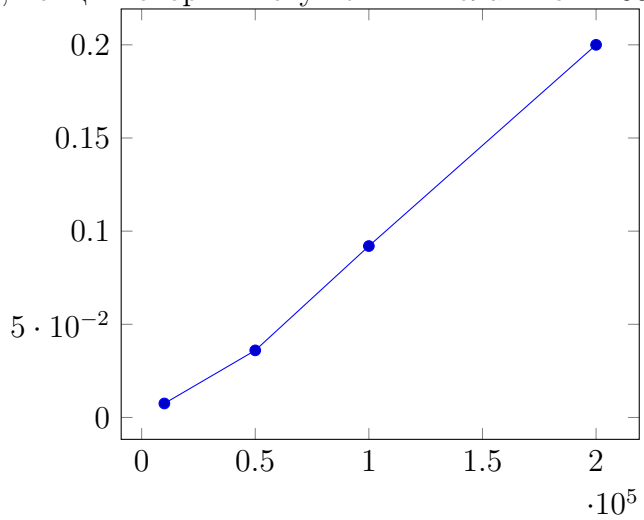
```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;
```

```
struct TNode {
int id;
char color = 'w';
vector<int> edges;
};

void DfsVisit(vector<TNode> &G, TNode &u, vector<int> &answ) {
u.color = 'g';
for (int i = 0; i < u.edges.size(); i++) {
if (G[u.edges[i]].color == 'w') {
answ.push_back(u.edges[i]);
DfsVisit(G, G[u.edges[i]], answ);
}
}
u.color = 'b';
}

void DFS(vector<TNode> &G) {
vector<int> answ;
for (int i = 1; i < G.size(); i++) {
if (G[i].color == 'w') {
answ.push_back(G[i].id);
DfsVisit(G, G[i], answ);
sort(answ.begin(), answ.end());
for (int j = 0; j < answ.size(); j++) {
cout << answ[j];
j == answ.size() - 1 ? cout << '\n' : cout << ' ';
}
answ.clear();
}
}

}

int main () {
int n, m, from, to;
cin >> n >> m;
vector<TNode> G(n + 1);
for (int i = 1; i <= n; i++) {
G[i].id = i;
}
```

```
for (int i = 0; i < m; i++) {
cin >> from >> to;
G[from].edges.push_back(to);
G[to].edges.push_back(from);
}
DFS(G);
return 0;
}
```

Тест производительности

Тесты представляют из себя файлы, в которых сгенерировано различное число отрезков, концы которых могут быть числами от -10000 до 10000.



Пояснения к графику: Ось y - время в секундах. Ось x - количество отрезков.

Выводы

Реализовать поиск в глубину было довольно просто. Этот алгоритм имеет много приложений помимо поиска компонент связности, например, он может использоваться в топологической сортировке. В целом теория графов помогает нам решать задачи, связанные с логистикой и различными передвижениями. На сегодняшний день существует огромное количество алгоритмов и приёмов, которые используются в теории графов.