Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

Лабораторная работа по предмету "Дискретный анализ" №9

Студент: Шипилов К. Ю.

Преподаватель: Макаров Н. К.

Группа: М8О-303Б-22

Дата: 12.12.2024

Оценка:

Подпись:

Москва 2024 г.

Оглавление

Постановка задачи

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 ввода

В первой строке заданы $1 \leqslant n \leqslant 10^5$ и $1 \leqslant \mathrm{m} \leqslant 10^5$. В следующих m строках записаны ребра. Каждая строка содержит пару чисел – номера вершин, соединенных ребром.

Формат вывода

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.

Алгоритм решения

Для поиска компонент связности необходимо выполнить поиск в глубину из каждой не посещенной вершины. При поиске в глубину номер вершины, добавляется в массив посещенных при поиске вершин. Этот массив и будет компонентой связности. После этого вершина помечается посещенной, и поиск в глубину выполняется рекурсивно для всех не посещенных вершин, в которые есть путь из текущей вершины. После окончания поиска массив, содержащий компоненту связности сортируется и выводится.

Далее алгоритм повторяется для остальных вершин, которые еще не были посещены.

Асимптотическая сложность алгоритма O(n + m), где n – количество вершин в графе, m – количество ребер, так как алгоритм DFS посещает каждую вершину 1 раз и не более двух раз переходит по каждому ребру графа.

Исходный код

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

void dfs(size_t v, const std::vector<std::vector<size_t>>&
graph,
         std::vector<bool>& check_visited, std::vector<size_t>&
path) {
  path.push_back(v);
  check_visited[v] = true;
  for (size_t u : graph[v]) {
    if (!check_visited[u]) {
      dfs(u, graph, check_visited, path);
    }
  }
}

void print_path(std::vector<size_t> path) {
  std::sort(path.begin(), path.end());
  for (size_t i : path) {
    std::cout << i + 1 << " ";
  }
  std::cout << std::endl;
}

int main() {
  size_t n, m;
  std::cin >> n >> m;
  std::vector<std::vector<size_t>> graph(n);

  for (size_t i = 0; i < m; ++i) {
    size_t v, u;
    std::cin >> v >> u;
    graph[v - 1].push_back(u - 1);
```

```
    graph[u - 1].push_back(v - 1);
  }

  std::vector<bool> check_visited(n, false);
  for (size_t v = 0; v < n; ++v) {
    if (check_visited[v]) {
      continue;
    }
    std::vector<size_t> path;
    dfs(v, graph, check_visited, path);
    print_path(path);
  }
}
```

Тесты

Test 1

Input	Output
5 4 1 2 2 3 1 3 4 5	1 2 3 4 5

Test 2

Input	Output
6 3 1 2 3 4 5 6	1 2 3 4 5 6

Test 3

Input	Output
4 6 1 2 1 3 1 4 2 3 2 4 3 4	1 2 3 4

Test 4

Input	Output
8 5 1 2 2 3 4 5 6 7 7 8	1 2 3 4 5 6 7 8

Вывод

Выполняя лабораторную работу, я повторил способы представления и алгоритмы обхода графов, а также их применение в алгоритме поиска компонент связности. Используемый подход на основе поиска в глубину (DFS) позволяет эффективно определять связные подграфы в неориентированных графах и обеспечивает линейную сложность, что делает его подходящим для работы с графами большого размера.