Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №9 по курсу «Дискретный анализ»

Студент: И. С. Глушатов
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.

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.

1 Описание

Неориентированный граф - это совокупность двух множеств: множеств элементов-вершин графа и их парные связи - рёбра. Компонентой связности графа называется максимальный связный подграф данного графа, т.е. такой подграф, в котором из любой вершины существует путь к любой другой её вершине. Для нахождения компонент связности можно использовать простой поиск в глубину или ширину. Для каждой вершины, в которой мы еще не побывали ни разу, запускаем обход в ширину, записываем, что вершины обхода принадлежат одной компоненте, заносим эти вершины в контейнер посещенных и продолжаем так, пока все вершины не будут обойдены. Таким образом мы поучим список компонент связности, представленных тоже списком, уже вершин.

2 Исходный код

Класс Графа содержит хэш-таблицу, где ключом является номер вершины, а значением множество вершин, в которые есть путь из данной. Метод AddEdge добавляет ребро неориентированного графа. Метод Components возвращает список компонент связности.

```
class Graph {

  using vertex_type = size_t;

  public:
    unordered_map<vertex_type, set<vertex_type>> edges;

  Graph() = default;
  Graph(const size_t n) {
    for (size_t i = 1; i <= n; i++)
      edges[i] = set<vertex_type>{i};
  }

  void AddEdge(const vertex_type& from, const vertex_type& to) {
    edges[from].insert(to);
    edges[to].insert(from);
  }

  vector<vector<vertex_type>> Components() {
    vector<vector<vertex_type>> result;
    map<vertex_type, vertex_type> visited;
    vertex_type current;

    for (const auto& vt : edges) {
      if (visited.find(vt.first) == visited.end()) {

        current = vt.first;
        vector<vertex_type> p;
        queue<vertex_type> queue;

        queue.push(current);

        while (!queue.empty()) {
          current = queue.front();
          queue.pop();

          if (visited.find(current) == visited.end()) {
            visited.insert({current, current});
            p.push_back(current);

            for (const vertex_type& ver : edges[current]) {
              queue.push(ver);
            }
          }
        }

        sort(p.begin(), p.end());
        result.push_back(p);

      }
    }

    sort(result.begin(), result.end(),
    [](const vector<vertex_type>& v1, const vector<vertex_type> v2) {
```

```
      return v1 < v2;
    });

    return result;
  }
};
```

В main считывается количество вершин и ребер, затем создается граф из n-1 вершины и добавляются ребра. Затем ищем компоненты связности и печатаем на экран.

```
int main() {

  size_t n = 0, m = 0;
  cin >> n >> m;

  Graph gr(n);

  for (size_t i = 0; i < m; i++) {
    size_t from, to;
    cin >> from >> to;
    gr.AddEdge(from, to);
  }

  print(gr.Components());

  return 0;
}
```

3 Консоль

```
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab9$ ./main.out
5 4
1 2
2 3
1 3
4 5
1 2 3
4 5
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab9$ ./main.out
5 4
1 1
1 1
1 1
1 1
1
2
3
4
5
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab9$
```

4 Тест производительности

Для тестов я использвал утилиту gnuplot для построения графиков зависимости времени работы программы от величины числа. Так же для сравнения использовал библиотеку chrono для замера времени.

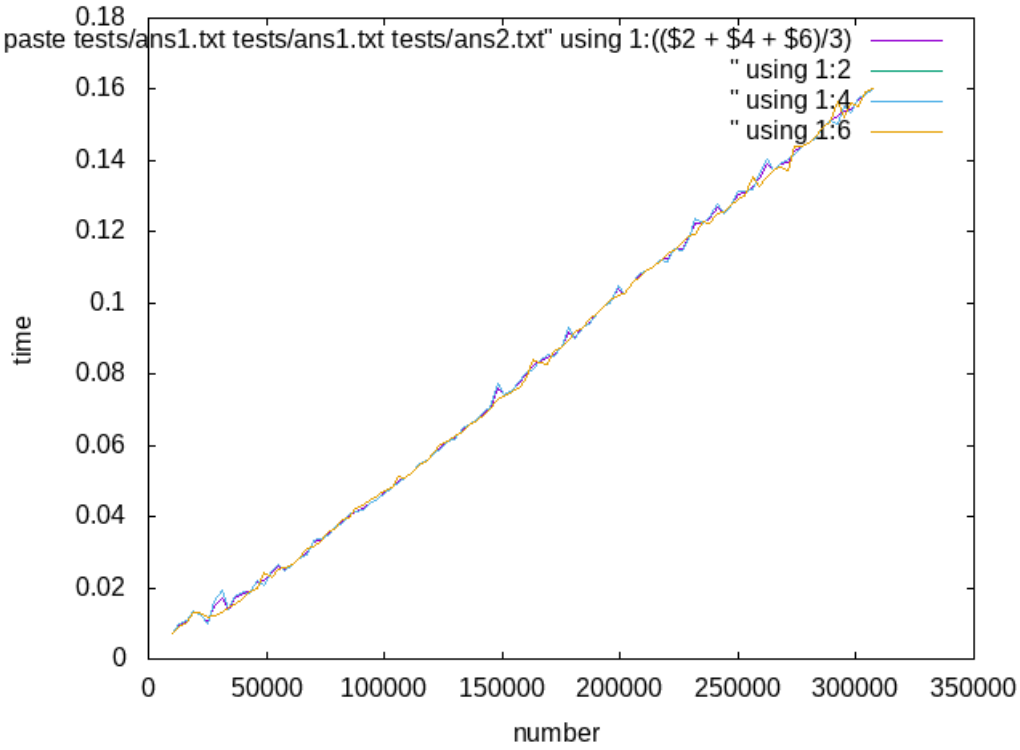

Рис. 1: График времени работы поиска компонент связности от суммы количества вершин и ребер

Как мы можем заметить на Рис. 1 время работы алгоритма линейно зависит от суммы колличества вершин и ребер.

5 Выводы

В ходе девятой лабораторной работы я познакомился с графами и их представлениями в языках программирования. Вспомнил, что такое компоненты связности графа и написал программу, которая с помощью обхода в ширину выводит их на экран. Проблем при написании лабораторной работы не было.

Список литературы

[1] *Поисковик - Google.*
URL: `https://www.google.com/`

[2] *Сайт с подробной документацией библиотек C++*
URL: `https://en.cppreference.com/`

[3] *О компонентах связности и как их найти*
URL: `https://ru.wikipedia.org/wiki/Компонента_связности_графа`