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. С. Гаврилов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

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.

Формат входных данных
В первой строке заданы 1 n 105 и 1 m 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.

1 Описание и исходный код

Находить компоненты связности будем, выполняя поиск в глубину. Записываем пройденные узлы в массив, указатель на который передается в функцию обхода. Пройденные вершины отмечаем и не начинаем обход из них. Полученные компоненты отсортируем с помощью поразрядной сортировки.

Класс узла графа:

```
1
2  struct vertCon {
3      bool marked;
4      std::vector<int> connectsTo;
5
6      vertCon() {
7          marked = false;
8      }
9  };
```

функция обхода компоненты связности:

```
1
2  void passComponent(int start, std::vector<vertCon>& vs, std::vector<int>* constructed)
       {
3
4      constructed->push_back(start);
5
6      vs[start].marked = true;
7
8      for (int i = 0; i < (int)vs[start].connectsTo.size(); ++i) {
9          if (!vs[vs[start].connectsTo[i]].marked) {
10             passComponent(vs[start].connectsTo[i], vs, constructed);
11         }
12     }
13 }
```

Основная функция:

```
1  int main() {
2      auto tm1 = std::chrono::steady_clock::now();
3      int vertCount, pathCount;
4      std::cin >> vertCount >> pathCount;
5
6      std::vector<vertCon> verts(vertCount + 1);
7
8      int vertA, vertB;
9      for (int i = 0; i < pathCount; ++i) {
10         std::cin >> vertA >> vertB;
11
12         verts[vertB].connectsTo.push_back(vertA);
13         verts[vertA].connectsTo.push_back(vertB);
```

```
14      }
15
16      std::vector<std::vector<int>*> components;
17
18      for (int i = 1; i < vertCount + 1; ++i) {
19
20          if (verts[i].marked) {
21              continue;
22          }
23
24          std::vector<int>* component = new std::vector<int>();
25          passComponent(i, verts, component);
26
27          components.push_back(component);
28      }
29
30      std::cerr << "comps sum: " << components.size() << std::endl;
31      for (size_t i = 0; i < components.size(); ++i) {
32          radixSort(*(components[i]));
33          for (size_t j = 0; j < components[i]->size(); ++j) {
34              printf("%d ", (*(components[i]))[j]);
35          }
36          printf("\n");
37
38          delete components[i];
39      }
40      auto tm2 = std::chrono::steady_clock::now();
41      auto delt = tm2 - tm1;
42      std::cerr << "test complete| " << std::chrono::duration_cast<std::chrono::
            milliseconds>(delt).count()<< "ms" <<std::endl;
43  }
```

2 Тест производительности

Все проходы по графу выполняются за v действий, где v – число вершин в графе, так как отмеченные вершины больше не посещаются. Поразрядная сортировка линейна, выполняется за O(n).

Итак, алгоритм должен иметь линейную асимптотику. Проверим:

```
max@max-Swift:~/Рабочий стол/ДА/lab9/DA_lab_9-main/simple$ ./gen.exe 100000
>testx
100000
max@max-Swift:~/Рабочий стол/ДА/lab9/DA_lab_9-main/simple$ ./lab9.exe <testx
>res
test complete| 428ms
max@max-Swift:~/Рабочий стол/ДА/lab9/DA_lab_9-main/simple$ ./gen.exe 200000
>testx
200000
max@max-Swift:~/Рабочий стол/ДА/lab9/DA_lab_9-main/simple$ ./lab9.exe <testx
>res
test complete| 853ms
max@max-Swift:~/Рабочий стол/ДА/lab9/DA_lab_9-main/simple$
```

Асимптотика линейная.

3 Выводы

В ходе выполнения этой лабораторной работы я на практике увидел, сколь пагубное влияние на производительность могут оказывать копирование при передаче и возврате структур из/в функцию. Долгое время я не мог уложиться в ограничение по времени именно из-за возврата целого вектора рекурсивной функцией, в то время как запись найденных вершин в один вектор, на который в функцию передается лишь ссылка выполнялась куда быстрее. Сам алгоритм достаточно прост и в то же время эффективен, думаю, опыт работы с ним всяко будет полезен.

Список литературы

[1] *Компонента связности графа — Википедия*
URL: https://ru.wikipedia.org/wiki/Компонента_связности_графа (дата обращения: 11.03.2021)

[2] *Использование обхода в глубину для поиска компонент сильной связности — Викиконспекты.*
URL: https://neerc.ifmo.ru/wiki/index.php?title=Использование_обхода_в_глубину_для
(дата обращения: 14.03.2021)