Санкт-Петербургский Государственный Университет

Факультет Прикладной математики – процессов управления

Лабораторная работа по курсу «Алгоритмы и анализ сложности» на тему «Эмпирический анализ алгоритма Косарайю поиска компонент сильной связности в орграфе»

Автор: Петров Михаил Иванович

333 группа ФИИТ

Содержание

Краткое описание алгоритма

В алгоритме Косарайю для нахождения сильных компонент сначала выполняется поиск в глубину в обратном порядке и сохраняются номера вершин в порядке обхода. Затем выполняется метод поиска в глубину (DFS), при этом порядок посещения не посещённых вершин определяется обратным порядком их последовательности, сохраненной при обходе в обратном порядке.

В данном алгоритме деревья в лесе DFS определяют сильные компоненты так же, как они определяют связные компоненты в случае неориентированных графов, то есть две вершины принадлежат одной и той же компоненте сильной связности тогда и только тогда, когда они принадлежат одному и тому же дереву этого леса.

Определения:

Сильно связный орграф – ориентированный граф, для любой вершины которого найдется ориентированный путь в нее из любой другой его вершины.

Компонента сильной связности в орграфе - максимальный по включению сильно связный подграф.

Математический анализ алгоритма

Пусть **G = (V,E)** – ориентированный граф.

Пусть орграф **G** представлен в виде списков смежности.

Метод состоит из двух процедур поиска в глубину (на орграфе **G** и на транспонированном орграфе, содержащем те же ребра, но противоположно направленные, что эквивалентно поиску в глубину в обратном порядке на графе **G**). Временная сложность каждой из данных процедур при выбранном способе хранения будет составлять **Θ(|V|+|E|)**.

Для хранения потребуются списки смежности исходного и транспонированного графов, количество элементов в которых равно **|V|+|E|**, вектор с индексами обратного порядка обхода, вектор, содержащий информацию о том, просмотрена ли вершина, вектор, содержащий номера сильных компонент для каждой вершины, состоящие из **|V|** элементов.

Следовательно, метод Косарайю потребует линейных затрат времени и пространства памяти.

В худшем случае (при насыщенных графах) оценка будет **Θ(|V|2)**.

Пусть граф представлен в виде матриц смежности. Сложность каждой из процедур поиска в глубину при выбранном способе хранения будет составлять **Θ(|V|2)** и, следовательно, время работы алгоритма будет квадратично.

Характеристики входных данных

При эмпирическом анализе алгоритма в качестве входных данных предполагались ориентированные графы, представленные в виде списков смежности. Каждой вершине соответствовал список из тех вершин, которые были связаны с данной исходящими из нее ребрами.Характеристиками при генерации списков было количество вершин орграфа. Количество вершин в графе (n) изменялось от 100 до 1000 с шагом в 100 вершин.

Для измерения трудоемкости алгоритма определялось время выполнения программы в миллисекундах. При каждом значении параметра n из заданного промежутка алгоритм Косарайю выполнялся по 10 раз для каждого случая. После чего время работы делилось на 10 с целью определения среднего времени работы для данных конкретного размера. (Измерение времени производилось с помощью функций языка программирования C++)

Код программы (реализация на языке C++)

```cpp
#include <functional>
#include <iostream>
#include <ostream>
#include <vector>

template<typename T>
std::ostream& operator<<(std::ostream& os, const std::vector<T>& v) {
    auto it = v.cbegin();
    auto end = v.cend();

    os << "[";
    if (it != end) {
        os << *it;
        it = std::next(it);
    }
    while (it != end) {
        os << ", " << *it;
        it = std::next(it);
    }
    return os << "]";
}

std::vector<int> kosaraju(std::vector<std::vector<int>>& g) {
    auto size = g.size();
    std::vector<bool> vis(size);
    std::vector<int> l(size);
    auto x = size;
    std::vector<std::vector<int>> t(size);

    std::function<void(int)> visit;
    visit = [&](int u) {
        if (!vis[u]) {
            vis[u] = true;
            for (auto v : g[u]) {
                visit(v);
                t[v].push_back(u);
            }
            l[--x] = u;
        }
    };

    for (int i = 0; i < g.size(); ++i) {
        visit(i);
    }
    std::vector<int> c(size);

    std::function<void(int, int)> assign;
    assign = [&](int u, int root) {
        if (vis[u]) {
            vis[u] = false;
            c[u] = root;
            for (auto v : t[u]) {
                assign(v, root);
            }
        }
    };

    for (auto u : l) {
        assign(u, u);
    }

    return c;
}

std::vector<std::vector<int>> g = {
    {1},
    {2},
    {0},
    {1, 2, 4},
    {3, 5},
    {2, 6},
    {5},
    {4, 6, 7},
};

int main() {
    using namespace std;

    cout << kosaraju(g) << endl;

    return 0;
}
```

Реализация генератора

(реализация на языке C++)

```cpp
std::vector<std::vector<int>> graph(int n) {
    std::vector<std::vector<int>> g(0);
    int s, m, t = 0;
    srand(time(0));
    for (int i = 0; i < n; ++i) {
        std::vector<int> p(0);
        t = 0;
        m = rand() % n;
        for (int j = 0; j < m; ++j) {
            s = rand() % n;
            if (j == 0) {
                p.push_back(s);
            }
            if (j > 0) {
                if (s > p[t]) {
                    p.push_back(s);
                    ++t;
                }
            }
        }
        g.push_back(p);
    }
    return g;
}
```

Функция генерирует орграф, представленный в виде списков смежности.

Входным параметром данной функции является число вершин графа.

Вычислительный эксперимент

В результате вычислительного эксперимента была составлена таблица, показывающая зависимость трудоёмкости алгоритма от количества вершин в сгенерированном орграфе.

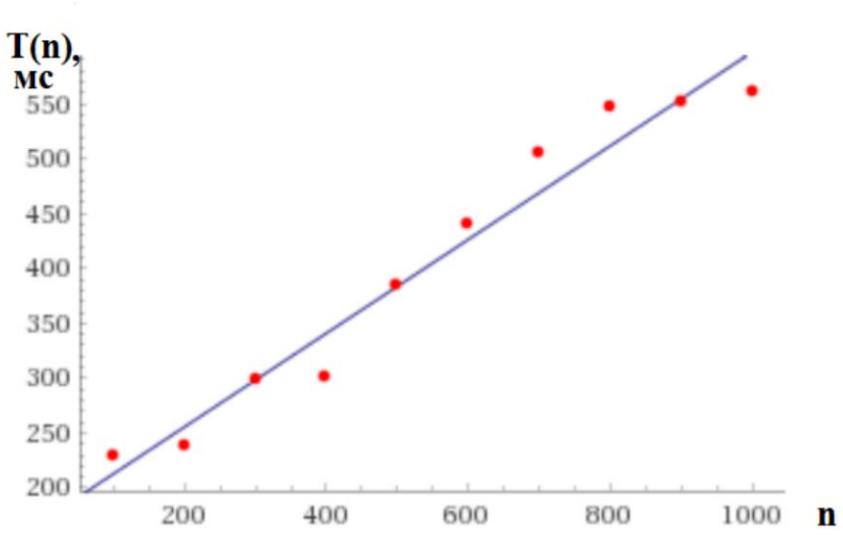
При каждом значении числа вершин графа из промежутка от 100 до 1000 с шагом 100 алгоритм выполнялся по 10 раз. Полученное время работы алгоритма делилось на 10 с целью определения средней трудоемкости при конкретных входных данных.

Полученная в результате эксперимента таблица:

n	100	200	300	400	500	600	700	800	900	1000
T(n), мс	229	238	298	302	384	441	504	546	551	560

Аппроксимируем и с помощью WolframAlpha получаем:

$$T(n) = 0.428667 * n + 169.533$$



По анализу алгоритм соответствует оценке Θ(n).

Список литературы

1. **Фундаментальные алгоритмы на C++ Алгоритмы на графа**х
   (Автор: Седжвик Р. Издательство: СПб.: ДиаСофтЮП Год издания: 2002)

2. http://e-maxx.ru/algo/strong_connected_components
   (Источник с описанием алгоритмов)

3. http://rosettacode.org/wiki/Kosaraju
   (Источник с реализацией алгоритмов)

Характеристика использованной вычислительной среды и оборудования:

Среда разработки: Visual Studio 2017.

Операционная система: MacOS Catalina.

Процессор: 2,3 GHz 2-ядерный процессор Intel Core i5

Оперативная память: 8 ГБ.