Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Курсовая работа

по дисциплине «Объектно-ориентированное программирование»

на тему «Алгоритм Косараю»

Выполнил студент Игнатьев В. В.
гр. 3331506/20101

Преподаватель Ананьевский М. С.
доцент

Санкт-Петербург

2025

Оглавление

Введение

Ориентированные графы являются фундаментальным инструментом для моделирования отношений в различных областях: от анализа социальных сетей и веб-структур до оптимизации баз данных и систем автоматизированного проектирования. Одной из ключевых задач при работе с такими графами является поиск сильно связных компонентов (Strongly Connected Components, SCC) — подмножеств вершин, где каждая вершина достижима из любой другой в пределах этого подмножества. Выявление SCC позволяет глубже понять структуру графа, упростить его анализ и решать прикладные задачи, такие как обнаружение циклов, кластеризация данных или проверка корректности распределённых систем.

Среди алгоритмов для поиска сильно связных компонентов особое место занимает алгоритм Косараю (Kosaraju's algorithm), предложенный в 1978 году. Его популярность обусловлена простотой реализации, высокой эффективностью и наглядной логикой, основанной на двух обходах графа в глубину (DFS). В отличие от методов Тарьяна или алгоритма поиска компонент через матрицу достижимости, подход Косараю не требует сложных структур данных и демонстрирует линейную временную сложность, что делает его применимым для работы с крупномасштабными графами.

Целью данной курсовой работы является изучение алгоритма Косараю, его теоретических основ, практической реализации и областей применения. В рамках исследования поставлены следующие задачи:

1. Рассмотреть понятие сильно связных компонентов и их значение в теории графов.

2. Изучить принципы работы алгоритма Косараю, включая инверсию графа и стратегию двойного обхода в глубину.

3. Провести сравнительный анализ с другими методами поиска SCC (например, алгоритмом Тарьяна).

4. Реализовать алгоритм на практике и протестировать его на различных примерах.

1. Теоретические сведения

Сильно связный компонент (Strongly Connected Component, SCC) в ориентированном графе — это максимальное подмножество вершин, все вершины внутри которого взаимно достижимы. Это свойство делает SCC ключевым элементом для анализа структуры графа, так как они позволяют декомпозировать граф на независимые «блоки» сильной связности.

Например, в графе, представленном на рис. 1, можно выделить четыре SCC:

- 3,5,6 (взаимно достижимые вершины),
- 1, 2, 4 (изолированные вершины).

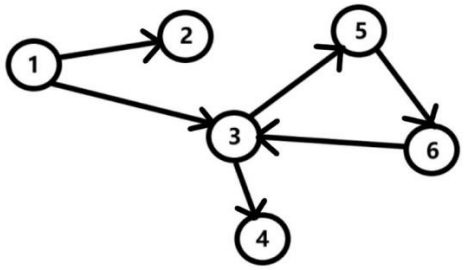

Рисунок 1 – Ориентированный граф

Сильно связные компоненты обладают свойством транзитивности: если вершина u принадлежит SCC S_2, а вершина v — SCC S_2, и существует путь из u в v, то не может существовать пути из v в u. Это позволяет строить конденсацию графа — новый граф, где каждый SCC представлен одной вершиной, а рёбра отражают связи между компонентами. Конденсация всегда является ациклическим ориентированным графом (DAG), что упрощает анализ исходной структуры.

Выявление SCC имеет широкое практическое применение:

1. Веб-аналитика: Поиск SCC используется для кластеризации веб-страниц, образующих «сильно связные ядра» в структуре интернета.

2. Распределённые системы: SCC помогают обнаруживать циклы в системах синхронизации или тупики (deadlocks).

3. Социальные сети: Анализ сообществ, где пользователи взаимно связаны.

4. Компиляторы: Оптимизация кода через выявление циклов в графах зависимостей.

Помимо алгоритма Косараю, существуют другие методы поиска сильно связных компонентов:

1. Алгоритм Тарьяна: Основан на одном обходе в глубину (DFS) с использованием стека и меток для отслеживания компонентов.

2. Алгоритм Габова (Path-based strong component algorithm): также использует DFS, но опирается на разделение графа на пути.

3. Алгоритм через матрицу достижимости: позволяет найти SCC через возведение матрицы смежности в степень.

Ключевые различия между алгоритмами:

- Сложность реализации: Алгоритм Косараю проще, чем Тарьяна или Габова, так как требует только двух обходов DFS.

- Потребление памяти: Алгоритм Тарьяна эффективнее по памяти, но менее интуитивен.

- Параллелизация: Алгоритм Косараю легче адаптировать для параллельных вычислений благодаря разделению этапов.

Формально, SCC можно описать через отношение эквивалентности на множестве вершин V:

- Рефлексивность: Каждая вершина достижима сама из себя.

– Симметричность: если *u* достижима из *v*, то *v* достижима из *u*.

– Транзитивность: если *u* достижима из *v*, а *v* — из *w*, то *u* достижима из *w*.

Разбиение графа на SCC соответствует разложению этого отношения на классы эквивалентности. Конденсация графа G в DAG G' может быть использована для решения задач топологической сортировки, поиска источников и стоков, что особенно полезно в задачах планирования.

2. Алгоритм Косараю

2.1. Историческая справка

Алгоритм был предложен в 1978 году индийским учёным Самиром Косараю (по другим данным — независимо открыт Майклом Шариром). Несмотря на простоту, он остаётся одним из самых популярных методов поиска SCC благодаря линейной сложности и минимальным требованиям к памяти.

2.2. Описание алгоритма

Алгоритм состоит из трёх этапов:

1. **Первый проход DFS**: выполняется обход в глубину исходного графа, вершины добавляются в стек в порядке завершения их обработки.
2. **Транспонирование графа**: создается граф с обратными направлениями всех ребер.
3. **Второй проход DFS**: Вершины обрабатываются в порядке, определенном стеком (от верхушки к основанию) на транспонированном графе. Каждый новый DFS обнаруживает одну сильно связанную компоненту.

2.3 Сравнение с алгоритмом Тарьяна

- Интуитивность: Косараю проще для понимания, так как разделяет этапы.

- Производительность: на практике алгоритм Тарьяна может быть быстрее из-за одного обхода DFS, но разница незначительна.

3. Практическая реализация и анализ

Программа, выполняющая алгоритм Косараджу:

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

using namespace std;

class Graph {
    int V; // Количество вершин
    vector<vector<int>> adj; // Список смежности

    // Рекурсивная функция для DFS
    void DFSUtil(int v, vector<bool>& visited, vector<int>& component) {
        visited[v] = true;
        component.push_back(v);

        for (int u : adj[v]) {
            if (!visited[u]) {
                DFSUtil(u, visited, component);
            }
        }
    }

public:
    Graph(int V) : V(V), adj(V) {}

    // Добавление ребра в граф
    void addEdge(int v, int w) {
        adj[v].push_back(w);
    }

    // Получение транспонированного графа
    Graph getTranspose() {
        Graph g(V);
        for (int v = 0; v < V; v++) {
            for (int u : adj[v]) {
                g.addEdge(u, v);
            }
        }
        return g;
    }

    // Заполнение стека вершинами в порядке завершения обработки
```

```
    void fillOrder(int v, vector<bool>& visited, stack<int>& Stack) {
        visited[v] = true;

        for (int u : adj[v]) {
            if (!visited[u]) {
                fillOrder(u, visited, Stack);
            }
        }

        // Все достижимые из v вершины обработаны, добавляем v в стек
        Stack.push(v);
    }

    // Основная функция для нахождения SCC
    vector<vector<int>> getSCCs() {
        stack<int> Stack;
        vector<bool> visited(V, false);

        // Заполняем стек в порядке завершения обработки вершин
        for (int i = 0; i < V; i++) {
            if (!visited[i]) {
                fillOrder(i, visited, Stack);
            }
        }

        // Создаем транспонированный граф
        Graph gr = getTranspose();

        // Снова помечаем все вершины как не посещенные
        fill(visited.begin(), visited.end(), false);

        vector<vector<int>> sccs;

        // Обрабатываем вершины в порядке, определенном стеком
        while (!Stack.empty()) {
            int v = Stack.top();
            Stack.pop();

            if (!visited[v]) {
                vector<int> component;
                gr.DFSUtil(v, visited, component);
                sccs.push_back(component);
            }
        }

        return sccs;
    }
};

int main() {
    // Создаем граф
    Graph g(5);
    g.addEdge(1, 0);
    g.addEdge(0, 2);
    g.addEdge(2, 1);
    g.addEdge(0, 3);
    g.addEdge(3, 4);

    // Находим сильно связанные компоненты
    vector<vector<int>> sccs = g.getSCCs();
```

```
    // Выводим результат
    cout << "Strongly Connected Components:\n";
    for (const auto& scc : sccs) {
        for (int v : scc) {
            cout << v << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Заключение

Алгоритм Косараю остаётся эталонным методом поиска SCC благодаря балансу между простотой и эффективностью. Его применение актуально в задачах анализа данных, веб-графов и распределённых систем. Несмотря на конкуренцию со стороны алгоритма Тарьяна, он сохраняет популярность в образовании и промышленности. Дальнейшие исследования могут быть направлены на оптимизацию для работы в распределённых средах и обработку динамически изменяющихся графов.

Список литературы

1. Филиппов А. Н. РАЗВИТИЕ МЕТОДА НУМЕРАЦИИ ЗНАЧЕНИЙ //Программирование. – 2010. – Т. 36. – №. 3. – С. 54-68. Белов, В. В.
2. Хадиев К. Р., Сафина Л. И. Квантовый алгоритм для нахождения кратчайшего пути в ациклическом ориентированном графе //Вестник Московского университета. Серия 15. Вычислительная математика и кибернетика. – 2019. – №. 1. – С. 48-52
3. Блюмин С. Л., Жбанова Н. Ю., Сысоев А. С. Ориентированные ациклические графы: моделирование метаграфами //СОВРЕМЕННЫЕ ИННОВАЦИИ В НАУКЕ И ТЕХНИКЕ. – 2020. – С. 40-45.
4. Рындин А. А., Саргсян Э. Р. ПРОГНОЗИРОВАНИЕ ПОВЕДЕНИЯ СЕТИ ПЕРЕДАЧИ ДАННЫХ В СИСТЕМЕ МОНИТОРИНГА ТЕЛЕКОММУНИКАЦИОННЫХ СЕТЕЙ НА ОСНОВЕ МОДИФИЦИРОВАННОГО АЛГОРИТМА КОСАРАЙЮ //Вестник Воронежского государственного технического университета. – 2020. – Т. 16. – №. 3. – С. 20-26.