Санкт-Петербургский политехнический университет Петра великого

Институт машиностроения, материалов и транспорта

Высшая школа автоматизации и робототехники

Курсовая проект

Дисциплина: Программирование на языках высокого уровня

Тема: Топологическая сортировка

Разработал:

студент гр. 3331506/70401 Кондратченко О.О.

Преподаватель Ананьевский М.С.

Санкт-Петербург

2020

**Оглавление**

[Введение 3](#_Toc42447300)

[Постановка задачи 3](#_Toc42447301)

[Описание алгоритма 3](#_Toc42447302)

[Реализация алгоритма 4](#_Toc42447303)

[Анализ алгоритма 5](#_Toc42447304)

[Сложность алгоритма 5](#_Toc42447305)

[Численный анализ алгоритма 5](#_Toc42447306)

[Область применения 6](#_Toc42447307)

[Заключение 7](#_Toc42447308)

[Список литературы 8](#_Toc42447309)

[Приложение 1 – Исходный код 9](#_Toc42447310)

# Введение

## Постановка задачи

Топологическая сортировка (для направленного ациклического графа) – линейное упорядочивание вершин так, что для каждого направленного ребра UV вершина U предшествует V в упорядочивании. Топологическая сортировка для графа невозможна, если граф не является направленным ацикличным.

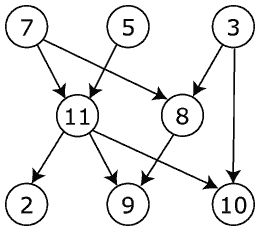


Рисунок 1 – Бесконтурный ориентированный граф (не отсортирован)

Для графа (рисунок 1) существует несколько согласованных последовательностей его вершин, которые могут получены при помощи данной сортировки.

Результатом топологической сортировки данного графа может быть следующие последовательности.

* 7, 5, 11, 3, 8, 2, 9, 10
* 3, 7, 5, 8, 11, 10, 9, 2

## Описание алгоритма

Топологическая сортировка в данной реализации использует обход графа в глубину, но при этом используется временный стек для хранения отсортированного списка, где начало стека соответствует последнему элементу.

Мы не печатаем вершину сразу, мы сначала рекурсивно вызываем *dfs* для всех смежных вершин, а затем помещаем ее в стек. По мере заполнения стека он образует отсортированную последовательность вершин, начиная с конца.

При этом стоит обратить внимание на то, что вершина помещается в стек только тогда, когда все смежные вершины уже находятся в стеке.

На выходе алгоритм лишь выдает топологически отсортированную последовательность вершин, вынимая элементы из стека с конца.

# Реализация алгоритма

Алгоритм реализован на языке C++.

Был создан *Graph* класс, в котором

* Имеется ключ Vertex (количество вершин), и list adj\_list (указатель смежности для вершин).

Реализованы методы:

* *addEdge* (создание графа, где добавляется узел и связь с вершинами), *printEdge* (вывод отсортированных вершин), *topologicalSort* (топологическая сортировка) и *dfs* (обход в глубину), два последних которые являются ключевыми функциями топологической сортировки.

На выходе алгоритм лишь выдает топологически отсортированную последовательность вершин, т.к на практике в большинстве случаев этого достаточно – лишь знать правильную последовательность “действий”.

# Анализ алгоритма

## Сложность алгоритма

Сложность такого алгоритма соответствует сложности алгоритма поиска в глубину, то есть O(m+n), где n – число вершин, m – число ребер. Это доказывается тем, что при проходе в глубину алгоритм лишь единожды проходит через единственную вершину, помечая ее как посещенную.

В большинстве случаев количество ребер в графе гораздо больше количества вершин. Поэтому определяющим фактором в сложности алгоритма является количество ребер графа.

## Численный анализ алгоритма

Посчитаем время заполнения нашего дерева различным количеством данных. Численные результаты приведены в таблице 1.

Таблица 1 -

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Количество узлов | 0 | 1000 | 10000 | 100000 | 200000 | 400000 | 800000 | 1600000 | 3200000 |
| Время, мс | 0 | 0,5 | 5 | 50 | 100 | 200 | 400 | 800 | 1600 |

На рисунке 2 приведен график времени выполнения в зависимости от количества узлов в графе:

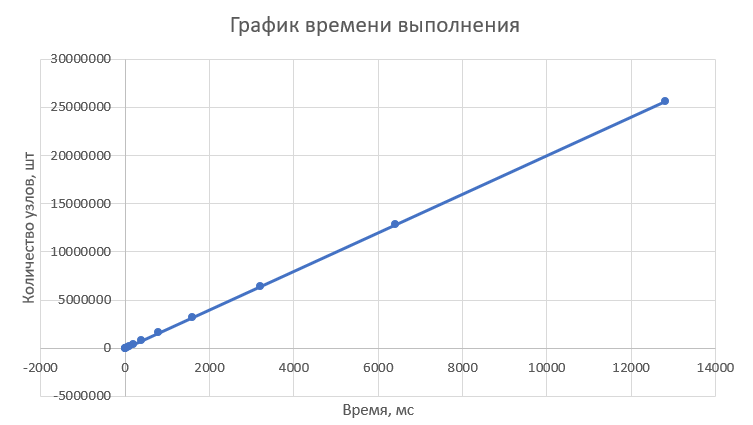


Рисунок 2 – Численный анализ

# Область применения

Топологическая сортировка применяется в самых разных ситуациях, например при создании параллельных алгоритмов, когда по некоторому описанию алгоритма нужно составить граф зависимостей его операций и, отсортировав его топологически, определить, какие из операций являются независимыми и могут выполняться параллельно (одновременно).

Примером использования топологической сортировки может служить создание карты сайта, где имеет место древовидная система разделов.

# Заключение

В ходе выполнения курсовой работы было рассмотрена топологическая сортировка, реализация алгоритма которого была выполнена на языке С++.

Так же был проверен анализ сложности алгоритма и его численный анализ, была рассмотрена область применения данной сортировки и ее особенности.

# Список литературы

1. *Левитин А. В.* Глава 5. Метод уменьшения размера задачи: Топологическая сортировка // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 220–224. — 576 с. — ISBN 978-5-8459-0987-9
2. *Рафгарден Тим.* Совершенный алгоритм. Графовые алгоритмы и структуры данных. - СПб.: Питер, 2019. - 256 с.: ил. - (Серия «Бибnиотека программиста») — ISBN 978-5-4461-1272-2

# **Приложение** 1 – Исходный код

