Национальный исследовательский технологический университет «МИСИС»

Отчёт

по дисциплине «Комбинаторика и теория графов»

на тему «Представления графов в компьютере».

Выполнил:

студент группы БИВТ-23-4

Дарьютин Даниил Денисович

Репозиторий:

https://github.com/DaniilDarjutin/MISIS\_Graph\_Dariutin.git

Москва, 2024

**Содержание**

1. **Введение**
2. **Теоретическое описание алгоритмов и структур данных**
3. **Описание алгоритмов и операций**
4. **Описание реализации и процесс тестирования**
5. **Анализ временной сложности**
6. **Заключение**
7. **Выводы**

**1. Введение**

В данной работе рассматривается разработка программного обеспечения для представления и работы с ориентированным взвешенным графом. Графы являются одной из основных структур данных, широко применяемых в различных областях, таких как анализ сетей, алгоритмы маршрутизации, анализ данных и моделирование сложных систем. Основной задачей данной работы является создание набора классов на языке программирования C#, обеспечивающих представление графа с использованием следующих структур данных:

* Список рёбер (Edge List)
* Список дуг (Arc List)
* Список смежности (Adjacency List)
* Список пучков дуг (Incidence List)

Кроме того, реализуются операции вставки, удаления и поиска вершин и рёбер, что позволяет гибко работать с графом, изменяя его структуру в процессе выполнения программы.

**Цель работы:**

* Разработать набор классов на языке C# для представления графа с различными структурами данных.
* Реализовать основные операции работы с графом (вставка, удаление, поиск).
* Провести анализ временной и пространственной сложности операций.

**Задачи работы:**

1. Изучить различные способы представления графов и выбрать наиболее подходящие для реализации.
2. Разработать архитектуру программы с использованием принципов объектно-ориентированного программирования.
3. Реализовать и протестировать основные операции для работы с графами.
4. Провести сравнительный анализ эффективности различных структур представления графа.

**2. Теоретическое описание представления графов**

Граф — это структура данных, состоящая из множества вершин (узлов) и множества рёбер (связей), которые соединяют пары вершин. Графы могут быть ориентированными или неориентированными. В данной работе рассматривается ориентированный взвешенный граф, в котором каждому ребру приписан вес, представляющий стоимость, расстояние или любую другую метрику.

Существует несколько способов представления графов в памяти компьютера, каждый из которых имеет свои преимущества и недостатки. Рассмотрим основные из них.

**2.1 Список рёбер (Edge List)**

Список рёбер представляет собой упорядоченный список всех рёбер графа. Каждое ребро хранится как кортеж, включающий начальную вершину, конечную вершину и вес.

Пример:

* Граф с вершинами **A, B, C** и рёбрами:

(A, B, 5)

(B, C, 10)

(C, A, 2)

Преимущества:

* Простая реализация.
* Подходит для хранения графов с небольшим числом рёбер (разреженных графов).

Недостатки:

* Поиск соседних вершин занимает время O(E), где E — количество рёбер.
* Низкая эффективность для плотных графов.

**2.2 Список дуг (Arc List)**

Список дуг — это расширенный вариант списка рёбер, в котором ребро рассматривается как направленная дуга. Для каждой дуги указывается начальная и конечная вершина, а также вес дуги.

Пример:

* Граф с дугами:

(A → B, вес 5)

(B → C, вес 10)

(C → A, вес 2)

Преимущества:

* Хорошо подходит для ориентированных графов.
* Позволяет легко учитывать направление дуги и её вес.

Недостатки:

* Трудоёмкий поиск соседей (аналогично списку рёбер).

**2.3 Список смежности (Adjacency List)**

Список смежности представляет собой массив списков, где каждая вершина связана со списком всех её соседних вершин и весов рёбер, соединяющих их.

Пример:

* Вершина A: [(B, 5)]
* Вершина B: [(C, 10)]
* Вершина C: [(A, 2)]

Преимущества:

* Эффективное хранение разреженных графов.
* Быстрый доступ к соседним вершинам (O(1) при наличии указателей на список).

Недостатки:

* Неудобно работать с плотными графами.
* Требует дополнительной памяти для указателей.

**2.4 Список пучков дуг (Incidence List)**

Список пучков дуг хранит для каждой вершины список инцидентных ей рёбер (пучков дуг), что позволяет легко находить все рёбра, исходящие из данной вершины.

Пример:

* Вершина A: [(A → B, вес 5)]
* Вершина B: [(B → C, вес 10)]
* Вершина C: [(C → A, вес 2)]

Преимущества:

* Удобен для работы с ориентированными графами.
* Позволяет быстро находить все исходящие и входящие рёбра.

Недостатки:

* Может требовать больше памяти по сравнению с другими структурами.
* Сложнее реализовать, чем список смежности.

**2.5 Сравнительный анализ**

|  |  |  |  |
| --- | --- | --- | --- |
| Структура | Преимущества | Недостатки | Временная сложность поиска соседей |
| Список рёбер | Простота, малое использование памяти для разреженных графов | Низкая эффективность поиска соседей | O(E) |
| Список дуг | Подходит для ориентированных графов | Трудоёмкий поиск соседей | O(E) |
| Список смежности | Эффективность для разреженных графов, быстрый доступ к соседям | Дополнительная память для указателей | O(1) |
| Список пучков дуг | Удобен для ориентированных графов, быстрый доступ к рёбрам | Высокие требования к памяти | O(1) |

Таким образом, для разреженных графов наиболее подходящим является список смежности, а для графов с большим количеством дуг выгодно использовать список пучков дуг.

**3. Описание алгоритмов и операций**

Для работы с графами, представленными различными структурами данных, необходимо реализовать следующие базовые операции:

* Вставка вершины.
* Вставка ребра.
* Удаление вершины.
* Удаление ребра.
* Поиск вершины.
* Поиск ребра.

Каждая операция имеет свою сложность, зависящую от выбранной структуры данных. Рассмотрим эти операции подробнее для каждой из структур.

**3.1 Вставка вершины**

При вставке новой вершины в граф необходимо добавить её в основную структуру, представляющую вершины графа. Это может быть список вершин или массив указателей (в зависимости от реализации).

Алгоритм:

1. Проверить, существует ли уже вершина с таким идентификатором.
2. Если вершина не найдена, добавить её в соответствующую структуру данных.
3. Инициализировать пустые списки смежности для новой вершины (если используется список смежности или пучков дуг).

Временная сложность:

* O(1) для списка смежности или списка пучков дуг, так как добавление элемента в конец списка занимает константное время.

**3.2 Вставка ребра**

При вставке ребра необходимо обновить информацию как в списке рёбер (или дуг), так и в списке смежности или пучков дуг (если они используются).

Алгоритм:

1. Проверить, существуют ли обе вершины, соединяемые ребром.
2. Проверить, существует ли уже ребро между данными вершинами.
3. Добавить новое ребро в список рёбер.
4. Обновить список смежности и список пучков дуг.

Временная сложность:

* O(1) для списка смежности.
* O(E) для списка рёбер, если требуется проверка на наличие дубликатов.

**3.3 Удаление вершины**

Удаление вершины требует обновления всех структур, хранящих информацию о вершинах и рёбрах, связанных с данной вершиной.

Алгоритм:

1. Найти вершину в основной структуре.
2. Удалить вершину из списка вершин.
3. Удалить все рёбра, связанные с данной вершиной, из списка рёбер, списка смежности и списка пучков дуг.

Временная сложность:

* O(V + E), где V — количество вершин, E — количество рёбер, так как требуется пройти по всем рёбрам, чтобы найти инцидентные.

**3.4 Удаление ребра**

Удаление ребра предполагает удаление его из всех структур, где оно было сохранено (список рёбер, список дуг, список смежности, список пучков дуг).

Алгоритм:

1. Найти ребро в списке рёбер или дуг.
2. Удалить ребро из списка рёбер.
3. Обновить список смежности и список пучков дуг, удалив соответствующее ребро.

Временная сложность:

* O(E) для поиска и удаления ребра.

**3.5 Поиск вершины**

Поиск вершины осуществляется путём проверки её наличия в списке вершин.

Алгоритм:

1. Проверить, существует ли вершина с заданным идентификатором в списке вершин.

Временная сложность:

* O(V) для линейного списка вершин.
* O(1) для хеш-таблицы или словаря.

**3.6 Поиск ребра**

Поиск ребра заключается в проверке наличия дуги между двумя заданными вершинами.

Алгоритм:

1. Проверить, существуют ли обе вершины.
2. Пройти по списку рёбер и найти ребро, соединяющее данные вершины.

Временная сложность:

* O(E) для списка рёбер.
* O(1) для списка смежности при наличии указателя на нужное ребро.

**4. Описание реализации и процесс тестирования**

**4.1 Используемые инструменты**

Для реализации задачи использовался язык программирования C# и среда разработки Visual Studio. C# был выбран благодаря встроенной поддержке коллекций (List, Dictionary), что упрощает реализацию структур данных для представления графов. Кроме того, C# обладает мощным набором инструментов для отладки и тестирования, что позволило провести полноценное тестирование разработанных классов.

**4.2 Реализация классов**

Разработанная система включает следующие основные классы:

* Graph — базовый класс графа, включающий методы вставки и удаления вершин и рёбер.
* EdgeListGraph — класс, представляющий граф в виде списка рёбер.
* AdjacencyListGraph — класс, представляющий граф в виде списка смежности.
* ArcListGraph — класс, представляющий граф в виде списка дуг.
* AdjacencyBundleGraph — класс, представляющий граф в виде списка пучков дуг.

Каждый из классов реализует базовые методы работы с графами, такие как добавление, удаление и поиск вершин и рёбер. В основе классов лежат коллекции C# (List и Dictionary), что позволяет эффективно управлять элементами графа.

**4.3 Особенности реализации**

При реализации классов была учтена возможность представления графа в разных форматах. Это позволило использовать полиморфизм для создания единого интерфейса работы с графами, независимо от их внутреннего представления. Основные методы включают:

* InsertVertex(id) — добавляет новую вершину в граф.
* InsertEdge(from, to, weight) — добавляет новое ребро или дугу в граф.
* RemoveVertex(id) — удаляет вершину из графа, а также все связанные с ней рёбра.
* RemoveEdge(from, to) — удаляет ребро между заданными вершинами.
* FindVertex(id) — проверяет наличие вершины в графе.
* FindEdge(from, to) — проверяет наличие ребра между двумя вершинами.

Эти методы реализованы в каждом классе в зависимости от используемой структуры данных.

**4.4 Тестирование**

Для проверки корректности работы разработанных классов были проведены следующие тесты:

1. **Тест на вставку вершин:**

* Создаётся пустой граф.
* Вставляется несколько вершин с уникальными идентификаторами.
* Проверяется наличие всех вставленных вершин методом FindVertex.

1. **Тест на вставку рёбер:**

* Создаётся граф с несколькими вершинами.
* Вставляются рёбра с различными весами.
* Проверяется наличие рёбер методом FindEdge.

1. **Тест на удаление вершин:**

* Создаётся граф с несколькими вершинами и рёбрами.
* Удаляется одна из вершин.
* Проверяется отсутствие удалённой вершины и всех рёбер, связанных с ней.

1. **Тест на удаление рёбер:**

* Создаётся граф с несколькими вершинами и рёбрами.
* Удаляется одно из рёбер.
* Проверяется отсутствие удалённого ребра методом FindEdge.

1. **Тест на поиск вершин и рёбер:**

* Создаётся граф с несколькими вершинами и рёбрами.
* Проверяется корректность поиска вершин и рёбер методами FindVertex и FindEdge.

**4.5 Анализ тестирования**

Результаты тестов показали, что все методы работают корректно для разных типов графов. Были протестированы графы с различным количеством вершин и рёбер, а также случаи, когда граф не содержит ни одной вершины или ребра. Все тесты были успешно пройдены, что подтверждает корректность реализованных алгоритмов.

**5. Анализ временной сложности**

Для оценки эффективности реализованных методов был проведён анализ временной и пространственной сложности. Рассмотрим основные методы и их сложность:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Метод** | |  | | --- | | **Список рёбер** |  |  | | --- | |  | | |  | | --- | | **Список дуг** |  |  | | --- | |  | | |  | | --- | | **Список смежности** |  |  | | --- | |  | | | **Список пучков дуг** | | --- |  |  | | --- | |  | |
| |  | | --- | | Вставка вершины |  |  | | --- | |  | | |  | | --- | | O(1) | | O(1) | O(1) | O(1) |
| |  | | --- | | Вставка ребра |  |  | | --- | |  | | O(E) | O(E) | O(1) | O(1) |
| Удаление вершины | O(V + E) | O(V + E) | O(V + E) | O(V + E) |
| Удаление ребра | O(E) | O(E) | O(1) | O(1) |
| Поиск вершины | O(V) | O(V) | O(1) | O(1) |
| Поиск ребра | O(E) | O(E) | O(1) | O(1) |

**Заключение:**

* Список рёбер и список дуг имеют более высокую сложность для операций с рёбрами, так как они требуют полного перебора.
* Список смежности и список пучков дуг имеют лучшую производительность, особенно для операций вставки и поиска рёбер.

**6. Заключение**

В данной работе были рассмотрены различные способы представления ориентированного взвешенного графа в виде четырёх структур данных: список рёбер, список дуг, список смежности и список пучков дуг. Для каждой структуры данных были реализованы основные операции: вставка, удаление и поиск вершин и рёбер. Также был проведён анализ временной сложности операций и выполнено тестирование.

Реализация показала, что использование различных структур данных позволяет оптимизировать работу алгоритмов в зависимости от характеристик графа. Например, для разреженных графов оптимально подходит список смежности, а для мультиграфов — список пучков дуг.

Таким образом, предложенные реализации покрывают широкий спектр задач и позволяют эффективно работать с графами различной сложности.

**7. Выводы**

1. Реализованы четыре различных структуры данных для представления графа.
2. Проведён анализ временной сложности операций, который подтвердил теоретические оценки.
3. Тестирование показало корректность реализации основных операций, а также эффективность структур данных для разных случаев использования.
4. Использование абстрактного класса Graph позволило унифицировать интерфейс и сделать код расширяемым.
5. Работа может быть использована в дальнейшем для разработки алгоритмов на графах, таких как поиск кратчайшего пути, обходы в ширину и глубину.