|  |  |
| --- | --- |
| Изображение выглядит как герб, эмблема, символ, нашивка  Автоматически созданное описание | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

# КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2 ПО ДИСЦИПЛИНЕ:**

**ТИПЫ И СТРУКТУРЫ ДАННЫХ**

***«Графы»***

Студент **Демирел Э.А.**

**Вариант 8**

Группа **ИУ7-31Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

|  |  |
| --- | --- |
| Студент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Демирел Э.А.** |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Барышникова М.Ю.** |

**2024**

**Условие задачи**

Обработать графовую структуру в соответствии с указанным вариантом задания. Обосновать выбор необходимого алгоритма и выбор структуры для представления графов. Предложить вариант реальной задачи, для решения которой можно использовать разработанную программу.

Ввод данных – на усмотрение программиста. Результат выдать в графической форме.

Задана система двусторонних дорог. Найти два города и соединяющий их путь, который проходит через каждую из дорог системы только один раз

**Техническое задание**

**Входные данные:**

Целое число – действие, выбранное пользователем (от 0 до 12).

Количество узлов (вершин) графа.

Названия городов.

Рёбра, с указанием узлов, которые они соединяют.

**Выходные данные:**

В результате операций можно получить изображение (файл формата .png), содержащий визуальное представление полученного графа. На изображении может быть указан найденный путь.

Временные затраты и затраты по памяти на некоторые действия выводятся на экран.

**Описание задачи:**

Программа обязана мочь строить граф из городов, введённых пользователем, находить «Эйлеров путь», визуализировать полученный граф.

**Способ обращения к программе:**

Обращение к программе происходит через запуск исполняемого файла app.exe. После запуска пользователем с клавиатуры вводятся желаемые действия.

**Аварийные ситуации:**

Все аварийные ситуации, связанные с неправильным вводом пользователя, обработаны. Единственными аварийными ситуациями может быть ошибка выделения памяти и ошибка создания dot файла и изображения, зачастую из-за отсутствия на компьютере пользователя утилиты Graphwiz.

**Используемые структуры данных**

typedef struct AdjListNode

{

int dest;

struct AdjListNode \*next;

} AdjListNode;

typedef struct AdjList

{

AdjListNode \*head;

} AdjList;

AdjListNode – узел списка смежности, dest – идентификатор смежной вершины, next – адрес следующего узла.

AdjList – список смежности, где head – «голова» списка.

#define MAX\_VERTICES 1000

#define MAX\_NAME\_LEN 50  
  
typedef struct Graph

{

int numVertices;

char vertexNames[MAX\_VERTICES][MAX\_NAME\_LEN];

AdjList \*array;

} Graph;

Graph – структура графа, где numVertices – количество вершин в графе, vertexNames – имена вершин, а array – массив списков смежности.

**Алгоритм программы**

Данная программа реализует управление графами с использованием списков смежности. Она предоставляет пользователю текстовое меню для выполнения различных операций, таких как добавление вершин и рёбер, поиск Эйлова пути, визуализация графа с помощью Graphviz и выход из программы.

Инициализация и отображение меню:

При запуске программы отображается меню с доступными опциями.

Программа ожидает ввода от пользователя для выбора одной из доступных опций (1-6).

**Обработка выбора пользователя:**

В зависимости от выбранной опции, программа выполняет соответствующую операцию:

1. Ввести количество городов (вершин)
2. Ввести названия городов
3. Ввести дороги (ребра)
4. Найти Эйлов путь
5. Визуализировать граф
6. Выход из программы

**Цикл работы программы:**

После выполнения выбранной операции программа возвращается к отображению меню.

Цикл продолжается до тех пор, пока пользователь не выберет опцию выхода.

**Ввод количества городов (вершин)**

1. Если граф уже существует, освобождается память, занимаемая предыдущим графом.
2. Если ранее был найден Эйлов путь, он также освобождается.
3. Программа запрашивает у пользователя ввести количество городов (вершин).
4. Пользователь вводит целое положительное число, не превышающее 1000.
5. Вызывается функция создания графа, которая:
6. Выделяет память для структуры шграфа.
7. Инициализирует массив списков смежности для каждой вершины.
8. Устанавливает количество вершин.
9. Программа выводит сообщение о успешной установке количества городов.

**Ввод названия городов**.

1. Если граф не создан (опция 1 не выполнена), выводится сообщение об ошибке.
2. Программа запрашивает у пользователя ввод названий городов по одному.
3. Максимальная длина имени ограничена 50 символами.
4. Пользователь вводит название города для каждой вершины.
5. Введённое название сохраняется в массиве vertexNames по соответствующему индексу.
6. После успешного ввода всех названий выводится соответствующее сообщение.

**Ввод дорог (рёбер)**

1. Если граф не создан или названия городов не введены, выводится сообщение об ошибке.
2. Вызывается функция удаления предыдущих рёбер, которая:
3. Проходит по всем вершинам графа.
4. Освобождает память, занятую существующими рёбрами.
5. Происходит освобождение ранее найденного Эйлова пути:
6. Программа запрашивает у пользователя ввести количество дорог (ребер).
7. Программа запрашивает у пользователя ввод двух названий городов, между которыми проводится дорога для каждой дороги.
8. Проверяется существование введённых городов в графе.
9. Если города найдены, вызывается функция добавления рёбер.
10. Проверяется, существует ли уже ребро между этими городами.
11. Если нет, добавляется ребро в оба направления (неориентированный граф).
12. Если да, выводится сообщение о существующем ребре и пропускается добавление.
13. Если города не найдены или ввод некорректен, пользователю предлагается повторить ввод.
14. После успешного добавления всех дорог выводится соответствующее сообщение.

**Поиск Эйлова пути**

Эйлов путь: Путь в графе, проходящий через каждое ребро ровно один раз.

1. Если граф не создан или не содержит вершин, выводится сообщение об ошибке.
2. Освобождение ранее найденного пути
3. Подсчитывается общее количество рёбер в графе.
4. Поиск Эйлова пути:
5. Создаётся копия графа для модификации.
6. Используется стек для хранения пути.
7. Алгоритм последовательно проходит по рёбрам, удаляя их из копии графа и записывая путь.
8. В конце проверяется, соответствует ли найденный путь количеству рёбер + 1.
9. Если соответствует, путь считается Эйлеровым и инвертируется для корректного порядка.
10. Если нет, путь не существует.
11. Возврат результата
12. Если путь найден, возвращается массив индексов вершин, представляющий путь, и размер пути.
13. Если путь не найден, возвращается пустое значение.
14. Если Эйлеров путь найден, программа выводит последовательность городов.
15. Если нет, выводится сообщение о его отсутствии.

**Визуализация графа**

1. Если граф не создан, выводится сообщение об ошибке.
2. Вызов функции создания файла для Graphwiz
3. Открывается файл с именем, указанным пользователем (например, graph\_output.dot).
4. В файл записываются определения узлов и рёбер:
5. Все вершины, включая изолированные (без рёбер), явно добавляются в файл DOT.
6. Все рёбра добавляются без дублирования.
7. Если Эйлов путь найден, соответствующие рёбра пути выделяются красным цветом и увеличенной толщиной линии.
8. Закрытие файла DOT.
9. С помощью команды dot из Graphviz создаётся изображение графа в формате PNG (например, graph\_output.png).
10. Если команда выполняется успешно, выводится сообщение о создании файла изображения.
11. Если возникает ошибка при выполнении команды, выводится соответствующее сообщение.
12. После успешной генерации изображения выводится сообщение с именем созданного файла.

**Примеры работы программы**

1. Ввод количества узлов.

Программа для поиска Эйлерова пути в невзвешенном графе, представленном в виде списка смежности.

--- Меню программы ---

1. Ввести количество городов (вершин)

2. Ввести названия городов

3. Ввести дороги (ребра)

4. Найти Эйлов путь

5. Визуализировать граф

6. Выход

Выберите опцию (1-6): 1

Введите количество городов (вершин): 3

Количество городов установлено на 3 за 0.000000000000 c.

Памяти потребовалось: 50020.00 байт.

2. Ввод названия городов (на основании п.1)

Выберите опцию (1-6): 2

Введите названия городов по одному. Максимальная длина имени: 49 символов.

Город 1: London

Город 2: Paris

Город 3: Moscow

Названия городов успешно введены.

3. Ввод рёбер (на основании п.1 и п.2)

Выберите опцию (1-6): 3

Существующие дороги удалены.

Введите количество дорог (ребер): 3

Введите дороги в формате 'город1 город2' по одной на строку:

Дорога 1: London Paris

Дорога 2: Paris Moscow

Дорога 3: Moscow London

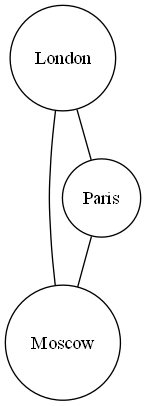
Дороги успешно введены; время потребовавшееся для этого: 0.000000000000 секунд.

4. Визуализация графа (на основании п. 1, 2 и 3)

Выберите опцию (1-6): 5

Файл изображения сгенерирован: graph\_output.png

graph\_output.png:



5. Поиск Эйлерова пути (на основании п. 1, 2 и 3)

Выберите опцию (1-6): 4

Найден Эйлов путь:

London -> Paris -> Moscow -> London

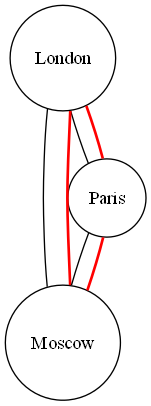
Время поиска Эйлерова пути: 0.000000000000 секунд.

6. Визуализация Эйлерова пути (на основании п.5) (красным)

Выберите опцию (1-6): 5

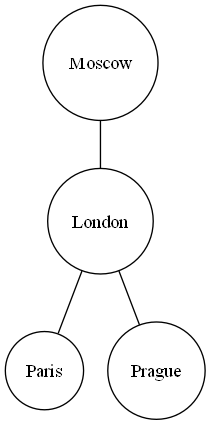
Файл изображения сгенерирован: graph\_output.png

graph\_output.png:



7. Поиск Эйлерова пути там, где он невозможен

Изначальный граф:



Выберите опцию (1-6): 4

Эйлов путь не существует в данной системе дорог.

Время поиска Эйлерова пути: 0.000000000000 секунд.

8. Ввод не числа в меню

Выберите опцию (1-6): ывфыф

Неверный ввод. Пожалуйста, введите число от 1 до 6.

9. Неправильный ввод в ребрах

Выберите опцию (1-6): 3

Существующие дороги удалены.

Введите количество дорог (ребер): 3

Введите дороги в формате 'город1 город2' по одной на строку:

Дорога 1: фывфыв

Неверный формат ввода. Ожидалось два названия городов.

**Пример прикладного использования реализованной программы**

Примером прикладного использования реализованной программы может служить построение маршрутов, например, туристических, между городами, чтобы при этом проехать все дороги (возможно, с какими-то дополнительными несущественными остановками), но при этом проезжать каждую дорогу только один раз для экономии топлива и финансов.

**Контрольные вопросы**

**0. Почему была выбрана реализация графа списком смежности?**

1. Экономия памяти

Разрежённые графы: Списки смежности требуют O(n + m) памяти, где n - количество вершин, а m - количество рёбер. Это значительно эффективнее по сравнению с матрицей смежности, особенно когда граф разрежён (мало рёбер).

2. Быстрый перебор соседних вершин

Легко и быстро перебираются все смежные вершины для заданной вершины, что важно для алгоритмов обхода (DFS, BFS) и поиска путей.

3. Гибкость при динамических изменениях

Добавление и удаление рёбер: Рёбра можно добавлять и удалять без значительных накладных расходов на перераспределение памяти, как это требуется при использовании матрицы смежности.

4. Удобство для большинства алгоритмов

Многие алгоритмы графов оптимально работают со списками смежности, обеспечивая лучшую производительность и упрощая реализацию.

**1. Что такое граф?**

Граф - это фундаментальная структура данных в информатике и математике, состоящая из множества вершин (узлов) и множества рёбер (связей) между ними. Графы используются для моделирования различных систем и процессов, где объекты представлены вершинами, а их взаимодействия - рёбрами.

**2. Как представляются графы в памяти?**

Существует несколько способов представления графов в памяти компьютера. Наиболее распространённые из них:

**Матрица смежности**

Представляет граф в виде двумерной матрицы. Если граф содержит n вершин, то создаётся матрица размера n x n. Элемент matrix[i][j] равен 1 (или весу ребра), если существует ребро между вершинами i и j, иначе 0.

**Список смежности**

Представляет граф в виде массива списков. Каждый элемент массива соответствует вершине и содержит список вершин, смежных с этой вершиной.

**Список рёбер**

Граф представлен списком всех рёбер. Каждый элемент списка содержит пару (или тройку, если веса) вершин, соединённых ребром.

**3. Какие операции возможны над графами?**

С графами можно выполнять множество операций, в зависимости от целей и задач. Основные операции включают:

1. Добавление и удаление вершин
2. Добавление и удаление рёбер
3. Поиск и проверка существования
4. Обход графа
5. Поиск путей
6. Определение связности
7. Определение компонентов связности
8. Определение циклов
9. Визуализация графа
10. Преобразования графов

**4. Какие способы обхода графов существуют?**

Обход графа — это процесс посещения всех его вершин и рёбер в определённом порядке. Существуют два основных метода обхода графов:

1. Обход в глубину (DFS - Depth-First Search)

Начинается с выбранной вершины. Исследует как можно глубже вдоль каждой ветви перед возвратом. Использует стек (можно реализовать рекурсивно).

2. Обход в ширину (BFS - Breadth-First Search)

Начинается с выбранной вершины. Исследует все соседние вершины на текущем уровне перед переходом к вершинам следующего уровня. Использует очередь.

**5. Где используются графовые структуры?**

* Компьютерные сети: Маршрутизация и топология.
* Социальные сети: Анализ связей между пользователями.
* Навигационные системы: Поиск маршрутов.
* Базы данных: Графовые базы данных для хранения взаимосвязанных данных.

**6. Какие пути в графе Вы знаете?**

* Путь: Последовательность вершин, где каждая пара соединена ребром.
* Цикл: Путь, начинающийся и заканчивающийся в одной вершине.
* Эйлеров путь: Проходит через каждое ребро ровно один раз.
* Гамильтонов путь: Проходит через каждую вершину ровно один раз.
* Кратчайший путь: Путь с минимальным количеством рёбер или минимальной суммой весов.

**7. Что такое каркасы графа?**

Каркас графа (остовное дерево) — подграф, включающий все вершины исходного графа и минимальное количество рёбер для поддержания связности.

**Вывод**

Была выбрана реализация графа списком смежности, так как она эффективно использует память для разрежённых графов, обеспечивает быстрый перебор соседних вершин и предоставляет гибкость при динамическом добавлении или удалении рёбер. Эти преимущества делают список смежности оптимальным выбором для нашей программы по работе с графами.