|  |  |
| --- | --- |
|  | **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»

**Лабораторная Работа №8 «Графы»**

**Вариант №7**

Студент **Поляков Андрей Игоревич** Группа **ИУ7-32Б**

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

|  |  |
| --- | --- |
| Студент | **Поляков А.И.** |
| Проверяющий |  |
|  |  |

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Описание условия задачи

# Обработать графовую структуру в соответствии с заданным вариантом. Обосновать выбор необходимого алгоритма и выбор структуры для представления графов. Результат выдать в графической форме.

# Техническое задание

## Исходные данные:

#### Пункты меню, выраженные целыми числами 0-9. Внутри некоторых пунктов вводятся числа.

## **Матрица смежности:**

## 1) Ввести граф

## 2) Вывести матрицу

## 3) Вывести граф

## 4) Найти минимальные пути

## **Список смежности**

## 5) Ввести граф

## 6) Вывести список

## 7) Вывести граф

## 8) Найти минимальные пути

## 9) Сравнение структур

## 0) Выход

## Результат:

#### Граф, матрица смежности, список смежности, результаты анализа производительности

## Описание задачи:

## Для каждой пары вершин графа найти длину кратчайшего пути между ними.

## Способ обращения к программе:

Запуск с помощью ./app.exe

## Аварийные ситуации и ошибки:

1. Некорректная команда
2. Не введена матрица
3. Не введен список
4. Ошибка выделения памяти

# Описание внутренних структур данных

Матрица смежности

**typedef struct**

**{**

**int\*\* matrix;**

**int vertices;**

**} graph\_mtr\_t;**

1. matrix – матрица
2. vertices – размер матрицы (количество вершин)

Список смежности

**typedef struct node**

**{**

**int vertex;**

**int weight;**

**struct node\* next;**

**} node\_t;**

**typedef struct**

**{**

**node\_t\*\* adjacency\_list;**

**int vertices;**

**} graph\_list\_t;**

node – узел списка

1. vertex – связанная вершина
2. weight – длина пути
3. next – следующий узел

graph\_list\_t - Список

1. adjacency\_list – массив списков для каждой из вершин
2. vertices – размер (кол-во вершин)

# Описание алгоритмов

1. Отобразить пользователю список команд и дождаться ввода номера нужной команды.
2. Для создания графа у пользователя поочередно запрашиваются длины путей, которые затем вводятся в матрицу или список смежности в зависимости от выбранной команды.
3. Команда вывода матрицы/списка выводит структуру.
4. Команда вывода графа создает dot файл и вызывает graphwiz.
5. Команда сравнения структур, генерирует случайный граф размером от 10 до 510, а затем засекает время поиска путей. Все замеры производятся путем многочисленных запусков с ожиданием RSE < 5. Затем производится анализ затраченного времени и памяти.

# Алгоритм поиска кратчайших путей

Алгоритм Флойда-Уоршелла используется для нахождения кратчайших путей между всеми парами вершин во взвешенном ориентированном графе.

Алгоритм Флойда-Уоршелла выполняется в следующих шагах:

1. **Инициализация**: Создаю копию D матрицы/списка смежностей. Если между вершинами нет ребра, то значение принимается равным бесконечности.
2. **Обновление расстояний**: Для каждой пары вершин (i, j) проверяем, можно ли улучшить текущее расстояние между ними, используя промежуточную вершину k. Для этого сравниваем значение D[i][j] с суммой D[i][k] и D[k][j]. Если сумма меньше, чем текущее значение D[i][j], то обновляем D[i][j] этой суммой. Повторяем шаг 2 для всех возможных промежуточных вершин k от 1 до n.
3. **Получение кратчайших путей**: После завершения алгоритма, матрица D содержит кратчайшие расстояния между всеми парами вершин.

Алгоритм Флойда-Уоршелла выполняется за O(n3) операций, где n – количество вершин в графе. Это делает его эффективным для небольших графов, но может быть менее эффективным для графов с большим количеством вершин.

# Тестовые данные

## Позитивные тесты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Тест** | **Входные данные** | | | | **Результат** |
| **Связный граф** | | | | | |
| Ввод связного графа | количество вершин графа: 3  1 - 2: 1  1 - 3: 2  2 - 3: 3 | | | | Z:\home\andrey05\Pictures\Screenshot_20231219_000417.png |
| Получить матрицу смежности графа | количество вершин графа: 3  1 - 2: 1  1 - 3: 2  2 - 3: 3 | | | | Z:\home\andrey05\Pictures\Screenshot_20231219_000546.png |
| Получить список смежности графа | количество вершин графа: 3  1 - 2: 1  1 - 3: 2  2 - 3: 3 | | | | Z:\home\andrey05\Pictures\Screenshot_20231219_000706.png |
| Найти минимальные пути | количество вершин графа: 3  1 - 2: 1  1 - 3: 2  2 - 3: 3 | | | | Z:\home\andrey05\Pictures\Screenshot_20231219_000957.png |
| **Несвязный граф** | | | | | |
| Ввод несвязного графа | | количество вершин графа: 3  1 - 2: 0  1 - 3: 0  2 - 3: 3 | | Z:\home\andrey05\Pictures\Screenshot_20231219_001514.png | |
| Получить матрицу смежности графа | | количество вершин графа: 3  1 - 2: 0  1 - 3: 0  2 - 3: 3 | | Z:\home\andrey05\Pictures\Screenshot_20231219_001642.png | |
| Получить список смежности графа | | количество вершин графа: 3  1 - 2: 0  1 - 3: 0  2 - 3: 3 | | Z:\home\andrey05\Pictures\Screenshot_20231219_001733.png | |
| Найти минимальные пути | | количество вершин графа: 3  1 - 2: 0  1 - 3: 0  2 - 3: 3 | | Z:\home\andrey05\Pictures\Screenshot_20231219_001824.png | |
| **Кратчайший путь – не прямой** | | | | | |
| Кратчайший путь в графе – не прямой | | | Z:\home\andrey05\Pictures\Screenshot_20231219_002024.png | Z:\home\andrey05\Pictures\Screenshot_20231219_002054.png | |
| Анализ производительности | | | - | Результаты анализа | |

## Негативные тесты

|  |  |  |
| --- | --- | --- |
| Вывод невведенного графа | - | Граф не введен |
| Вывод невведенной матрицы | - | Граф не введен |
| Вывод невведенного списка | - | Граф не введен |
| Некорректная команда | abc | Неверная команда |

# Замеры

Программа генерирует случайный граф размером от 10 до 510, а затем засекает время поиска путей. Все замеры производятся путем многочисленных запусков с ожиданием RSE < 5. Затем производится анализ затраченного времени и памяти.

Размер графа: 10

Реализация с помощью матрицы:

| Время, нс | Кол-во итераций | RSE

| 6976.53 | 30 | 3.68

Занимаемая память - 400 байт

Реализация с помощью списка:

| Время, нс | Кол-во итераций | RSE

| 22173.10 | 20 | 2.98

Занимаемая память - 1440 байт

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Разность производительности = 15196.57 нс

Реализация с помощью списка дольше на 217.824039 %

Разность занимаемой памяти = 1040 байт

Реализация с помощью списка больше на 260.000000 %

Размер графа: 110

Реализация с помощью матрицы:

| Время, нс | Кол-во итераций | RSE

| 3520893.77 | 70 | 4.40

Занимаемая память - 48400 байт

Реализация с помощью списка:

| Время, нс | Кол-во итераций | RSE

| 5171694.70 | 20 | 3.01

Занимаемая память - 191840 байт

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Разность производительности = 1650800.93 нс

Реализация с помощью списка дольше на 46.885849 %

Разность занимаемой памяти = 143440 байт

Реализация с помощью списка больше на 296.363636 %

Размер графа: 210

Реализация с помощью матрицы:

| Время, нс | Кол-во итераций | RSE

| 21734911.70 | 10 | 0.15

Занимаемая память - 176400 байт

Реализация с помощью списка:

| Время, нс | Кол-во итераций | RSE

| 29593145.90 | 10 | 0.52

Занимаемая память - 702240 байт

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Разность производительности = 7858234.20 нс

Реализация с помощью списка дольше на 36.154894 %

Разность занимаемой памяти = 525840 байт

Реализация с помощью списка больше на 298.095238 %

Размер графа: 310

Реализация с помощью матрицы:

| Время, нс | Кол-во итераций | RSE

| 67937167.90 | 10 | 0.19

Занимаемая память - 384400 байт

Реализация с помощью списка:

| Время, нс | Кол-во итераций | RSE

| 93987158.30 | 10 | 1.83

Занимаемая память - 1532640 байт

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Разность производительности = 26049990.40 нс

Реализация с помощью списка дольше на 38.344240 %

Разность занимаемой памяти = 1148240 байт

Реализация с помощью списка больше на 298.709677 %

Размер графа: 410

Реализация с помощью матрицы:

| Время, нс | Кол-во итераций | RSE

|156777068.80 | 10 | 0.18

Занимаемая память - 672400 байт

Реализация с помощью списка:

| Время, нс | Кол-во итераций | RSE

|206093558.00 | 10 | 0.27

Занимаемая память - 2683040 байт

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Разность производительности = 49316489.20 нс

Реализация с помощью списка дольше на 31.456443 %

Разность занимаемой памяти = 2010640 байт

Реализация с помощью списка больше на 299.024390 %

Размер графа: 510

Реализация с помощью матрицы:

| Время, нс | Кол-во итераций | RSE

|304284850.80 | 10 | 0.09

Занимаемая память - 1040400 байт

Реализация с помощью списка:

| Время, нс | Кол-во итераций | RSE

|400158256.20 | 10 | 0.14

Занимаемая память - 4153440 байт

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Разность производительности = 95873405.40 нс

Реализация с помощью списка дольше на 31.507781 %

Разность занимаемой памяти = 3113040 байт

Реализация с помощью списка больше на 299.215686 %

# Результаты анализа:

По результатам анализа ясно, что наиболее подходящая для данной задачи структура – матрица, так как она выигрывает по скорости и по памяти. В алгоритме Флойда-Уоршелла, на каждом шаге, для каждой пары вершин (i, j) проверяется, можно ли улучшить текущее расстояние между ними, используя промежуточную вершину k. Это требует прямого доступа к весам ребер между каждой парой вершин, что удобно для матрицы смежности.

# Ответы на вопросы

1. Что такое граф?

Граф - это абстрактная математическая структура, представляющая собой множество вершин (или узлов), соединенных рёбрами (или дугами). Графы используются для моделирования отношений между объектами.

2. Как представляются графы в памяти?

Графы могут быть представлены различными способами в памяти компьютера. Некоторые из распространенных представлений включают матрицы смежности и списки смежности. В матрице смежности используется двумерный массив для хранения информации о связях между вершинами, а в списке смежности каждая вершина имеет список своих соседей.

3. Какие операции возможны над графами?

Операции над графами включают добавление и удаление вершин и рёбер, проверку наличия ребра между двумя вершинами, нахождение соседей вершины, обход графа и нахождение кратчайших путей.

4. Какие способы обхода графов существуют?

Обходы графов могут быть в глубину (DFS - Depth-First Search) и в ширину (BFS - Breadth-First Search). DFS исследует как можно глубже в структуру графа, прежде чем возвращаться, в то время как BFS идет по уровням, исследуя вершины на текущем уровне перед переходом к следующему.

5. Где используются графовые структуры?

Графовые структуры широко используются в различных областях, таких как информатика, транспортное планирование, социальные сети, биоинформатика, графовые базы данных, анализ сетей и др.

6. Какие пути в графе Вы знаете?

В графе могут существовать различные типы путей, включая простой путь (без повторяющихся вершин), цикл (замкнутый путь), путь между двумя вершинами, кратчайший путь (с минимальной суммой весов рёбер) и др.

7. Что такое каркасы графа?

Каркас графа - это подграф, который включает в себя все вершины и некоторое подмножество рёбер исходного графа, образующее дерево. Каркасы часто используются в алгоритмах поиска минимального остовного дерева, где стремятся найти подграф с минимальной суммой весов рёбер, охватывающий все вершины исходного графа.

# Вывод

Результаты экспериментов показали, что для поставленной задачи оптимальнее всего использовать матрицу смежностей и алгоритм Флойда-Уоршелла. Данную программу можно использовать, когда задана система двусторонних дороги для каждой пары городов найти длину кратчайшего пути между ними