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 №8 ПО ДИСЦИПЛИНЕ:
ТИПЫ И СТРУКТУРЫ ДАННЫХ

Графы

Студент **Павлов Д. В.**

Группа **ИУ7-33Б**

Вариант 7

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

Студент _____ **Павлов Д. В.**

Преподаватель _____ **Никульшина Т. А.**

2024

1. Описание условия задачи.

Обработать графовую структуру в соответствии с заданным вариантом. Обосновать выбор необходимого алгоритма и выбор структуры для представления графов. Ввод данных осуществить на усмотрение программиста. Результат выдать в графической форме. Задана система двусторонних дорог. Для каждой пары городов найти длину кратчайшего пути между ними.

2. Описание технического задания.

Задана система двусторонних дорог. Для каждой пары городов найти длину кратчайшего пути между ними.

Входные данные.

Строка: путь к файлу

Выходные данные.

Сообщение об успехе или ошибке выполнения пункта, кратчайшие пути для каждой пары вершин, изображение графа.

Способ обращения к программе.

Запускается через терминал командой *./app*

Возможные аварийные ситуации и ошибки пользователя.

Некорректный ввод данных. Ошибка выделения памяти.

3. Описание внутренних структур данных.

Структура графа.

```
typedef struct
{
   int **adjacency_matrix; // Матрица смежности
   int **next; // Матрица следующих вершин в пути
   size_t size; // Количество вершин в графе
} graph_t;
```

Интерфейс для работы со структурами.

Для работы с деревьями

```
// Очистка памяти, выделенной под граф
void graph_free(graph_t *graph);

// Выделение памяти под граф
int graph_alloc(graph_t *graph);
```

```
// Считывание графа
int fread_graph(char *file_name, graph_t *graph);

// Поиск кратчайших путей для каждой пары вершин
void find_shortest_paths(graph_t *graph);

// Вывод кратчайшего пути для каждой пары вершин
void print_shortest_paths(graph_t graph);

// Вывод графа графически
int open_dot_img(const char *file_name, graph_t graph);
```

4. Описание алгоритма.

Используется алгоритм Флойда-Уоршалла для поиска кратчайших путей и матрица смежности для представления графа.

Описание поиска кратчайших путей:

1. Алгоритм использует три вложенных цикла:
    - k - промежуточная вершина, через которую может проходить путь
    - i - начальная вершина пути
    - j - конечная вершина пути
2. Если все условия выполнены:
    - вычисляется длина пути через вершину k: new_dist = расстояние(i → k) + расстояние(k → j)
    - если новый путь короче существующего (new_dist < adjacency_matrix[i][j])
        - обновляем длину пути (adjacency_matrix[i][j] = new_dist)

Алгоритм перебирает все возможные промежуточные вершины k и проверяет, можно ли сократить путь из i в j, если пойти через вершину k. Если такой более короткий путь найден, информация о нем сохраняется.

5. Пример работы программы.

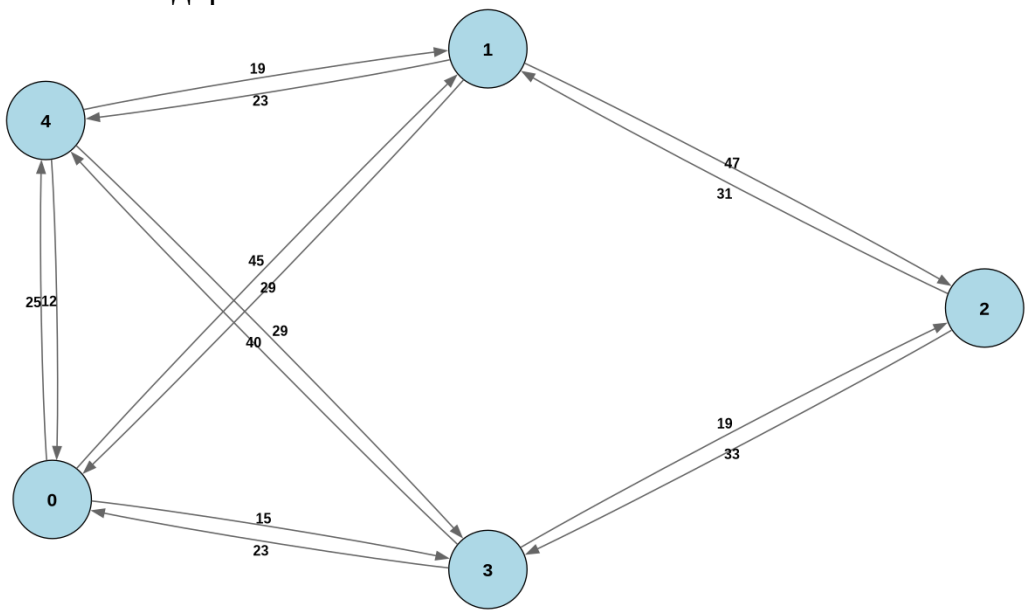
1. Неверный ввод файла

```
Введите название файла: non_exist.txt
Ошибка при чтении файла. Файл не найден или он записан неверно.

Process finished with exit code 3
```

3. Пустой ввод.

```
Введите название файла:
Ошибка ввода имени файла.

Process finished with exit code 3
```

2. Правильный ввод файла.



Кратчайший путь от 0 до 1: 0 -> 4 -> 1 = 44 Кратчайший путь от 0 до 2: 0 -> 3 -> 2 = 34 Кратчайший путь от 0 до 3: 0 -> 3 = 15 Кратчайший путь от 0 до 4: 0 -> 4 = 25 Кратчайший путь от 1 до 0: 1 -> 0 = 29 Кратчайший путь от 1 до 2: 1 -> 2 = 47 Кратчайший путь от 1 до 3: 1 -> 0 -> 3 = 44 Кратчайший путь от 1 до 4: 1 -> 4 = 23 Кратчайший путь от 2 до 0: 2 -> 3 -> 0 = 56 Кратчайший путь от 2 до 1:	Кратчайший путь от 2 до 3: 2 -> 3 = 33 Кратчайший путь от 2 до 4: 2 -> 1 -> 4 = 54 Кратчайший путь от 3 до 0: 3 -> 0 = 23 Кратчайший путь от 3 до 1: 3 -> 2 -> 1 = 50 Кратчайший путь от 3 до 2: 3 -> 2 = 19 Кратчайший путь от 3 до 4: 3 -> 4 = 40 Кратчайший путь от 4 до 0: 4 -> 0 = 12 Кратчайший путь от 4 до 1: 4 -> 1 = 19 Кратчайший путь от 4 до 2: 4 -> 0 -> 3 -> 2 = 46 Кратчайший путь от 4 до 3:

2 -> 1 = 31	4 -> 0 -> 3 = 27

6. Выводы по проделанной работе.

Для поставленной задачи, а именно поиск кратчайших путей между всеми вершинами, подходит представление графа в виде матрицы смежности и алгоритм Флойда-Уоршалла.

Выбор алгоритма Флойда-Уоршалла обоснован следующими преимуществами:

1. Требует только одного запуска для нахождения всех кратчайших путей, в отличие от алгоритма Дейкстры, который нужно запускать для каждой вершины
2. Более эффективен для плотных графов
3. Прост в реализации и дает наглядный код
4. Позволяет легко восстанавливать найденные пути
5. Временная сложность $O(v^3)$

Выбор матрицы смежности как структуры данных обусловлен:

1. Алгоритм Флойда-Уоршалла изначально разработан для работы с матрицей смежности
2. Обеспечивает константное время доступа к любому ребру $O(1)$
3. Естественно поддерживает модификацию весов при поиске кратчайших путей
4. Пространственная сложность $O(v^2)$

Пример реальной задачи: Система маршрутизации для транспортной компании:

- Вершины - города или складские центры
- Рёбра - дороги между ними
- Веса - расстояния или время в пути
- Двунаправленные рёбра с разными весами учитывают особенности маршрута (пробки в одном направлении, платные участки и т.д.)
- Необходимо быстро находить оптимальные маршруты между любыми двумя точками.

Контрольные вопросы.

1. Что такое граф?

Граф – это конечное множество вершин и соединяющих их рёбер, т.е.:
G = {V, E}, где V – конечное непустое множество вершин; E – множество ребер (пар вершин).

2. Как представляются графы в памяти?

Основные представления графа в памяти: матрица смежности, матрица инцидентности, список рёбер, список смежности.

3. Какие операции возможны над графами?

Над графами возможны следующие операции:

- поиск кратчайшего пути от одной вершины к другой (если он есть);
- поиск кратчайшего пути от одной вершины ко всем другим;
- поиск кратчайших путей между всеми вершинами;
- поиск эйлерова пути (если он есть);
- поиск гамильтонова пути (если он есть).
- добавление/удаление вершины.
- добавление/удаление ребра.

4. Какие способы обхода графов существуют?

Основные способы обхода графов: поиск в ширину (BFS – Breadth First Search) и поиск в глубину (DFS – Depth First Search).

5. Где используются графовые структуры?

Графовые структуры обычно используются там, где нужно представить и анализировать связи между объектами.

6. Какие пути в графе Вы знаете?

В графе может быть простой, замкнутый, эйлеров или гамильтонов путь.

7. Что такое каркасы графа?

Каркасом, или остовным деревом для графа называется связный подграф этого графа, содержащий все вершины графа и не имеющий циклов.