



**Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное 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***

**Вариант № 12**

**Студент: Бондарева В. А.**

**Группа: ИУ7-34Б**

**Преподаватель: Никульшина Т. А.**

**2025 г.**

## Условие задачи

Реализовать алгоритмы обработки графовых структур: поиск различных путей, проверка связности, построение остовых деревьев минимальной стоимости. 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.

Условие по варианту №12: задана система двусторонних дорог. Найти множество городов, расстояние от которых до выделенного города (столицы) больше, чем Т.

## Описание ТЗ

### Исходные данные

Исходными данными программы являются:

1. Строка, содержащая единственное целое положительное число, являющееся пунктом меню.

Пользователю предоставляются следующие опции меню:

- |  |
|--|
| <ul style="list-style-type: none"><li>0. Выход</li><li>1. Создать граф</li><li>2. Уничтожить граф</li><li>3. Настроить граф вручную</li><li>4. Считать граф из файла</li><li>5. Задать столицу</li><li>6. Найти кратчайшее между двумя городами</li><li>7. Найти дальние города</li><li>8. Проверить связность графа</li><li>9. Вывести граф</li></ul> |
|--|

Листинг 1. Опции меню.

2. Строка, содержащая имя файла, над которым необходимо произвести какие-либо операции.
3. Строка, содержащая слово (название города), над которым необходимо произвести какие-либо операции, например, добавить в граф.
4. Строка, содержащее число (длина дороги), над которым необходимо произвести какие-либо операции.

### Результат

Результатом выполнения программы являются:

1. Вывод меню для выбора опции (главное меню или подменю для ручной настройки графа).

2. Текущее состояние графа в виде формата «png».
3. Таблицы, отображающие результат выполнения операций.
4. Сообщения о статусе выполнения программы и ошибках.

#### *Способ обращения к программе*

Пользователь обращается к программе при помощи исполняемого файла app.exe.

#### *Возможные аварийные ситуации и ошибки пользователя*

Все возможные аварийные ситуации и ошибки пользователя описаны типом status\_t, включающим в себя коды возврата программы:

1. ERR\_IO: ошибка ввода/вывода данных программы.
2. ERR\_RANGE: неверно указано количество чего-либо.
3. ERR\_MEM: ошибка при работе с памятью.
4. ERR\_ARGS: ошибка при работе с аргументами функции.
5. ERR\_FILE: ошибка при работе с файлом.
6. ERR\_NO\_PATH: ошибка для случая, когда искомого пути не существует.
7. ERR\_NOT\_FOUND: ошибка при отсутствии необходимого элемента.
8. ERR\_GRAPH\_DOESNT\_EXIST: ошибка инициализации графа (граф не был инициализирован).
9. ERR\_INVALID\_GRAPH\_DATA: ошибка существования не валидных данных графа.
10. ERR\_ALREADY\_EXISTS: необходимый элемент уже существует.
11. ERR\_INVALID\_POINTER: ошибка работы с указателем.

### **Описание внутренних структур данных**

```
typedef struct
{
    char **cities_names;
    char *capital;
    size_t **roads;
    size_t cities_quantity;
    size_t max_vertices_quantity;
} graph_t;

size_t *distances;
```

Листинг 2. Внутренние структуры данных.

В данной лабораторной работе единственной необходимой структурой является структура графа graph\_t. Она содержит следующие поля:

- cities\_names: массив указателей на названия городов
- capital: столица графа, обязательно содержится в cities\_names

- roads: матрица смежности, содержащая элементы типа size\_t. Представляет из себя матрицу размеров cities\_quantity на cities\_quantity, где значение длины дороги из города с индексом n до города с индексом m (в cities\_names) содержится в ячейке матрицы roads[n][m]

- cities\_quantity: количество существующих в графе дорог

- max\_vertices\_quantity: максимальное количество вершин графа (т.е. максимальная его вместимость, граф не расширяется)

Массив расстояний distances хранит в себе расстояния от столицы до каждого города, создается в функции dijkstra\_graph и не уничтожается по ее завершению. Таким образом массив дистанций можно использовать в дальнейшей обработке.

В листинге №3 представлены сигнатуры основных функций.

```
/** @brief Функция для ввода графа из файла (с помощью файлового потока
filestream). */
status_t input_graph_from_file(graph_t *graph, FILE *filestream);

/** @brief Функция для добавления города на граф. */
status_t add_city_to_graph(graph_t *graph, const char *city);

/** @brief Функция для добавления дороги на граф. */
status_t add_road_to_graph(graph_t *graph, size_t index_city_1, size_t
index_city_2, size_t distance_1_to_2, size_t distance_2_to_1);

/** @brief Функция для удаления дороги из графа. */
status_t remove_road_from_graph(graph_t *graph, size_t index_city_1, size_t
index_city_2);

/** @brief Функция для удаления города с графа. */
status_t remove_city_from_graph(graph_t *graph, const char *city);

/** @brief Функция для перевода графа (в его представлении структурой)*/
status_t export_graph_to_dot_file(graph_t *graph, const char *filename);

/** @brief Функция для поиска кратчайших расстояний между городами. */
status_t dijkstra_graph(graph_t *graph, size_t src, size_t **dist);

/** @brief Функция для поиска городов, расстояние до которых от указанного
города. */
status_t find_cities_farther_than_t_distance(graph_t *graph, size_t
distance_t, size_t **far_cities, size_t *count);

/** @brief Функция для проверки связности графа. */
status_t is_graph_connected(const graph_t *graph, bool *is_connected);

/** @brief Функция для обновления столицы графа. */
status_t set_graph_capital(graph_t *graph, const char *capital);

/** @brief Функция для получения индексов двух городов по их названиям. */
status_t get_cities_indexes(graph_t *graph, const char *city_1, const char
*city_2, size_t *indx1, size_t *indx2);

/** @brief Функция для инициализации графа. */
status_t init_graph(graph_t *graph, size_t cities_quantity);

/** @brief Функция для уничтожения (освобождения) графа. */
status_t free_graph(graph_t *graph);
```

## Алгоритм

Помимо основного алгоритма программы (main.c) основополагающими алгоритмами для реализации заданного функционала являются:

- алгоритм поиска кратчайших расстояний между городами (алгоритм Дейкстры)
- алгоритм проверки связности графа
- алгоритм поиска городов, расстояние до которых от указанного города больше, чем расстояние T

Рассмотрим каждый алгоритм.

1. Проверяется корректность входных данных.
2. Выделяется память под массив расстояний и массив для пометки посещенных вершин.
3. Инициализируется массив расстояний: расстояние до источника равно 0, до всех остальных — бесконечность. Все вершины помечаются непосещенными. Массив расстояний является результирующим и не уничтожается при завершении работы алгоритма.
4. В цикле (количество итераций равно числу вершин минус находится непосещенная вершина с минимальным расстоянием и помечается посещенной).
5. Для этой вершины проверяются все возможные соседи: если сосед не посещен, есть дорога до него, и новое расстояние через текущую вершину меньше старого — обновляется расстояние до соседа.
6. Освобождается выделенная память.

Листинг 4. Алгоритм Дейкстры.

1. Проверяется корректность входных данных.
2. Выделяется память под массив посещенных вершин.
3. Выполняется поиск в глубину (DFS) начиная с нулевой вершины, все достижимые вершины отмечаются посещенными.
4. Проверяется, все ли вершины посещены: если да - флаг is\_connected устанавливается в true, иначе в false.
5. Освобождается выделенная память.

Листинг 5. Алгоритм проверки связности.

1. Проверяется корректность входных данных.
2. Ищется индекс столицы в графе.
3. Запускается алгоритм Дейкстры от столицы, вычисляются расстояния до всех городов, результат сохраняется в distances.
4. Выделяется память под массив дальних городов.
5. Перебираются все города: если расстояние до города больше заданного и не бесконечность - индекс города добавляется в массив far\_cities, счётчик таких городов увеличивается.
6. Освобождается выделенная память.

Листинг 6. Алгоритм нахождения дальних городов.

## Тесты

### *Позитивные тесты*

Номер теста	Описание	Ожидаемый результат
1	Корректный ввод пункта меню.	<div> <div>7. Найти дальние города</div> <div>8. Проверить связность графа</div> <div>ФУНКЦИИ ВЫВОДА</div> <div>9. Вывести граф</div> </div> <p>Выберите пункт меню: 3</p>
2	Корректное считывание графа из файла.	<div>9. Вывести граф</div> <p>Выберите пункт меню: 4 Введите имя файла с графом: data/graph1.txt Все прошло хорошо :)</p> <div>М Е Н У</div>
3	Корректное добавление города в граф.	<div>3. Удалить город</div> <div>4. Удалить дорогу</div> <p>Выберите пункт меню: 1 Введите название нового города: SvobodaPopygaiam Все прошло хорошо :)</p> <div>М Е Н У</div>
4	Корректное добавление дороги.	<div>4. Удалить дорогу</div> <p>Выберите пункт меню: 2 Введите город №1: SvobodaPopygaiam Введите город №2: Saratov Введите длину дороги из SvobodaPopygaiam в Saratov: 100500 Введите длину дороги из Saratov в SvobodaPopygaiam: 1 Все прошло хорошо :)</p> <div>М Е Н У</div>
5	Корректное нахождение дальних городов.	<div>Выберите пункт меню: 7</div> <p>Введите расстояние T, будем искать места столь отдаленные: 20</p> <div> <div>ГОРОДА, СТОЛЬ ОТДАЛЕННЫЕ</div> <div> Столица: Moscow  0. NewYork  1. Saratov  2. SvobodaPopygaiam </div> </div> <p>Все прошло хорошо :)</p>

Таблица 1. Позитивные тесты.

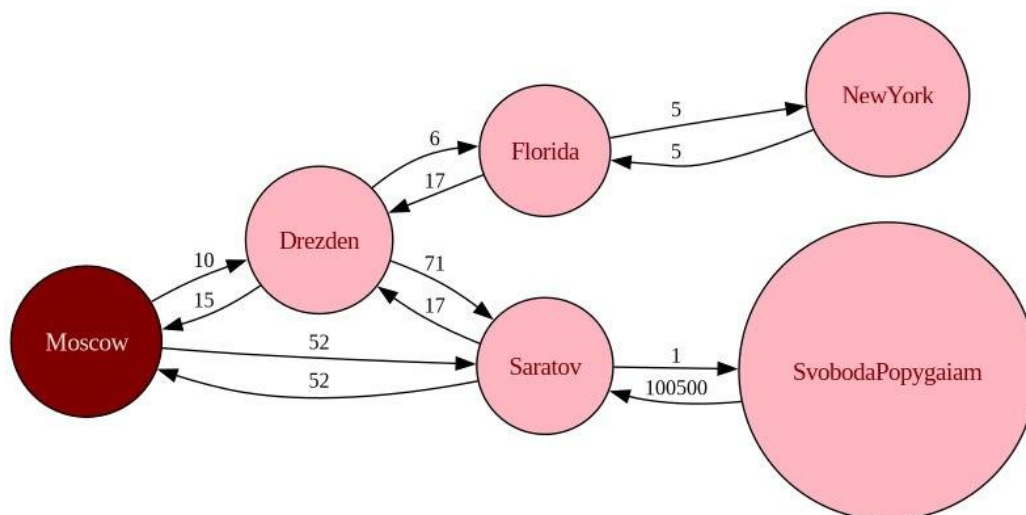


Рис. 1. Граф после действий, описанных в позитивных тестах.

## Негативные тесты

Номер теста	Описание	Ожидаемый результат
1	Некорректный ввод пункта меню.	 <p>6. Найти кратчайшее между двумя городами 7. Найти дальние города 8. Проверить связность графа</p> <p>ФУНКЦИИ ВЫВОДА</p> <p>9. Вывести граф</p> <p>Выберите пункт меню: one Выберите пункт меню: Выберите пункт меню: Выберите пункт меню:</p>
2	Попытка удаления города из пустого графа.	 <p>3. Удалить город 4. Удалить дорогу</p> <p>Выберите пункт меню: 3 Введите название удаляемого города: Baumanka Произошла ошибка: ничего не найдено</p> <p>М E N U</p> <p>0. Выход</p>
3	Попытка удаления дороги из пустого графа.	 <p>4. Удалить дорогу</p> <p>Выберите пункт меню: 4 Введите город №1: first Введите город №2: second Введите длину дороги из first в second: 10 Введите длину дороги из second в first: 20 Произошла ошибка при работе с аргументами функции</p> <p>М E N U</p>
4	Попытка произведения операций над несуществующим графом.	 <p>9. Вывести граф</p> <p>Выберите пункт меню: 6 Граф не существует. Попробуйте инициализировать граф (пункт меню №1)</p> <p>М E N U</p> <p>0. Выход</p>
5	Ошибка при работе с файлом.	 <p>9. Вывести граф</p> <p>Выберите пункт меню: 4 Введите имя файла с графом: datadata.txt Произошла ошибка при работе с файлом</p> <p>М E N U</p> <p>0. Выход</p>

Таблица 2. Негативные тесты.

## Обоснования

Алгоритм Дейкстры был выбран для решения данной задачи, так как в ее контексте система дорог является взвешенным графом. Данный алгоритм эффективно находит кратчайшие расстояния от одной вершины до всех остальных, что прямо соответствует требованию условия - определить, какие города находятся дальше заданного расстояния  $T$ . По сравнению с другими алгоритмами (например, Флойда-Уоршелла, который находит расстояние между всеми парами вершин, что избыточно для данной задачи) он менее требователен к памяти и является оптимальным выбором.

Матрица смежности как структура данных для хранения системы дорог удовлетворяет требованиям задачи, таким как поддержка ориентированности и разных весов в противоположных направлениях (матрица смежности позволяет хранить разные значения для  $roads[i][j]$  и  $roads[j][i]$ ) и большая скорость доступа к элементам матрицы.

## Контрольные вопросы

### 1. Что такое граф?

Граф – это конечное множество вершин и ребер, соединяющих их, т. е.:  $G = \langle V, E \rangle$ , где  $V$  – конечное непустое множество вершин;  $E$  – множество ребер (пар вершин). Если пары  $E$  (ребра) имеют направление, то граф называется ориентированным (орграф), если иначе – неориентированный (неорграф). Если в пары  $E$  входят только различные вершины, то в графе нет петель. Если ребро графа имеет вес, то граф называется взвешенным. Степень вершины графа равна числу ребер, входящих и выходящих из нее (инцидентных ей). Неорграф называется связным, если существует путь из каждой вершины в любую другую.

### 2. Как представляются графы в памяти?

Одним из видов представления графов в памяти является матрица смежности. Матрица смежности — это квадратная матрица размером  $n$  на  $n$ , где  $b[i][j] = 1$  при наличии ребра между вершинами  $V_i$  и  $V_j$ , и  $0$  — при отсутствии. Для неориентированных графов такая матрица симметрична.

### 3. Какие операции возможны над графами?

Над графами возможно произвести следующие операции:

- поиск кратчайшего пути от одной вершины к другой (если он есть);
- поиск кратчайшего пути от одной вершины ко всем другим (если таковые есть);
- поиск кратчайших путей между всеми вершинами;
- поиск эйлера пути (если он есть);
- поиск гамильтонова пути (если он есть);
- добавление вершины графа;
- добавление дороги на граф;
- удаление вершины графа;
- удаление дороги между двумя вершинами;

### 4. Какие способы обхода графа существуют?

- Обход в глубину (DFS): начиная с выбранной вершины, алгоритм уходит как можно дальше по первому доступному ребру, пока не упрётся в тупик; затем возвращается назад и пробует следующее ребро. Использует стек для сохранения пути.

- Обход в ширину (BFS): начиная с выбранной вершины, алгоритм сначала посещает всех ее непосредственных соседей, затем соседей соседей и так далее по уровням. Использует очередь для обработки вершин, идеален для поиска кратчайшего пути в невзвешенном графе и проверки достижимости.

5. *Где используются графовые структуры?*

В построении маршрутов и навигации, для анализа связей между пользователями в социальных сетях и разработки рекомендационных систем, для анализа маршрутизации пакетов данных, в разработке искусственного интеллекта (игровая индустрия), моделировании физики, а также в разработке баз данных.

6. *Какие пути в графе Вы знаете?*

Существуют следующие пути в графе:

- простой путь: путь без повторяющихся вершин;
- кратчайший путь: путь с минимальным суммарным весом рёбер (или минимальным количеством рёбер в невзвешенном графе);
- эйлеров путь: путь, проходящий через каждое ребро графа ровно один раз;
- гамильтонов путь: путь, проходящий через каждую вершину графа ровно один раз.

7. *Что такое каркасы в графе?*

Каркас графа - это подграф, который содержит все вершины исходного графа и является деревом. Он включает в себя все вершины исходного графа и связывает эти вершины в единую структуру без лишних связей.