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

Студент **Паншин Сергей Константинович**

Группа **ИУ7-33Б**

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

Студент _____ **Паншин С. К.**

Преподаватель _____ **Никульшина Т. А.**

Преподаватель _____ **Барышникова М. Ю.**

Оценка _____

2023 г.

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.

Техническое задание

Найти все вершины графа, к которым от заданной вершины можно добраться по пути не длиннее A.

Входные данные:

Пункт меню (число от 0 до 9):

```
Menu:
    1) Init graph
    2) Add edge
    3) Show graph
    4) Is graph connected
    5) Change start node
    6) Find nodes, that path len from start node <= a
    7) Print paths to n node from start node
    8) Build minimum cost spanning trees
    9) Time comparation
    0) Quit program
```

Также путь к файлу, номера узлов и веса ребер, максимальная длина.

Выходные данные:

Картинка графа, номера узлов.

Возможные аварийные ситуации:

Некорректный ввод: пункта меню, некорректный путь, не корректный номер узла.

Способ обращения к программе

В папке с программой запустить команду *make run*.

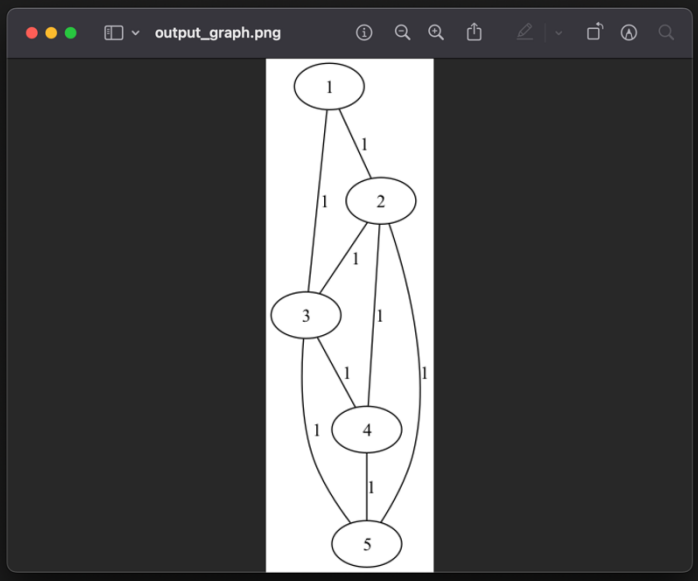
Структуры данных

```
// граф
typedef struct graph
{
    int n;
    int **paths;
    int edge_count;
} graph_t;

// пути
typedef struct path
{
    int n;
    int start_node;
    int *lens;
    int *nodes_marked;
    int *path;
} path_t;

// инициализация пути
// вход: указатель на указатель на путь, размер
// выход: код возврата
int path_init(path_t **path, int n);

// очистка пути
// вход: указатель на указатель на путь
void path_free(path_t **path);

// инициализация графа
// вход: указатель на указатель на граф, размер
// выход: код возврата
int graph_init(graph_t **g, int n);

// очистка графа
// вход: указатель на указатель на граф
void graph_free(graph_t **g);

// добавления ребра в граф
// вход: указатель на граф, вершины ребра и его вес
// выход: код возврата
int graph_add_edge(graph_t *g, int v, int u, int len);

// граф в дот формат
// вход: указатель на файл, имя, указатель на граф
void graph_to_dot(FILE *f, const char *graph_name, graph_t *g);
```

```
// чтение ребра
// вход: размер графа, указатели на вершины ребра и его вес
void read_edge(int n, int *v, int *u, int *len);

// чтение графа из файла
// вход: указатель на указатель на граф, указатель на файл
// выход: код возврата
int graph_read_from_file(graph_t **g, FILE *file);


// поиск всех минимальных путей в графе
// вход: указатель на указатель на граф, указатель на указатель на
путь, стартовая вершина
// выход: код возврата
int dijkstra(graph_t *g, path_t **path, int start_node);

// вершины, до которых путь не более a
// вход: указатель на путь, макс длина
// выход: вектор вершин
vector_t *get_nodes_len_le_a(path_t *path, int a);

// путь из стартовой вершина в заданную
// вход: указатель на путь, номер вершины
// выход: вектор вершин
vector_t *get_path_to_t(path_t *path, int to);ъ

// построение минимального остового дерева
// вход: указатель на граф
void minimum_spanning_tree(graph_t *g);
```

Пример работы

```
FILE LOADED SUCCESS!
------------------
Menu:
    1) Init graph
    2) Add edge
    3) Show graph
    4) Is graph connected
    5) Change start node
    6) Find nodes, that path len from start node <= a
    7) Print paths to n node from start node
    8) Build minimum cost spanning trees
    9) Time comparation
    0) Quit program
------------------
Choose menu item (0-9):
3
------------------
Menu:
    1) Init graph
    2) Add edge
    3) Show graph
    4) Is graph connected
    5) Change start node
    6) Find nodes, that path len from start node <= a
    7) Print paths to n node from start node
    8) Build minimum cost spanning trees
    9) Time comparation
    0) Quit program
------------------
Choose menu item (0-9):
6
Enter max len number (> 0):
1
Nodes that path len from 1 <= 1
2 3
------------------
Menu:
    1) Init graph
    2) Add edge
    3) Show graph
    4) Is graph connected
    5) Change start node
    6) Find nodes, that path len from start node <= a
    7) Print paths to n node from start node
    8) Build minimum cost spanning trees
    9) Time comparation
    0) Quit program
------------------
```



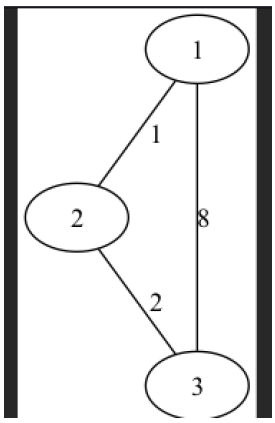
Примечание: при попытке создать не связанный граф, не связанные вершины не отображаются.

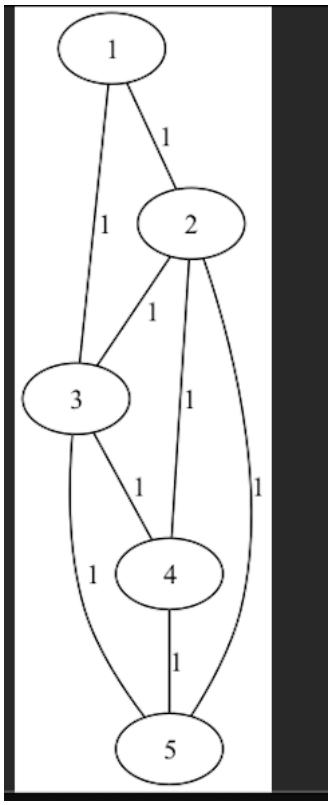
Замеры

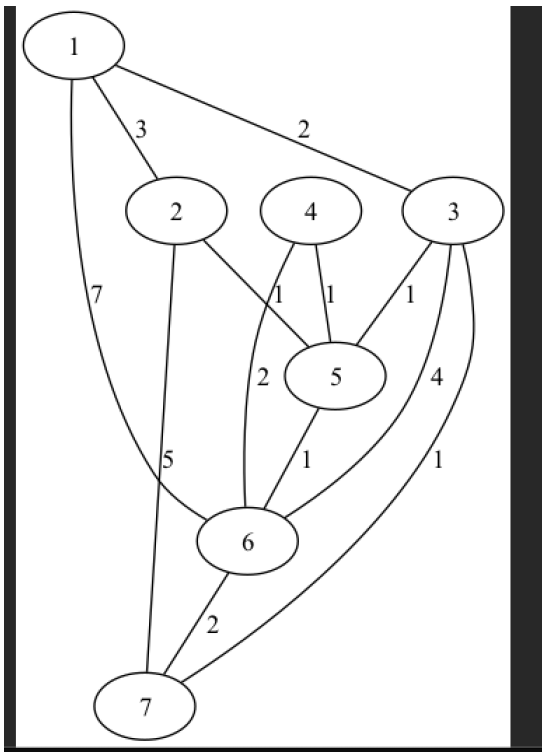
Производится 2000 итераций при замерах, на графах, представленных ниже.

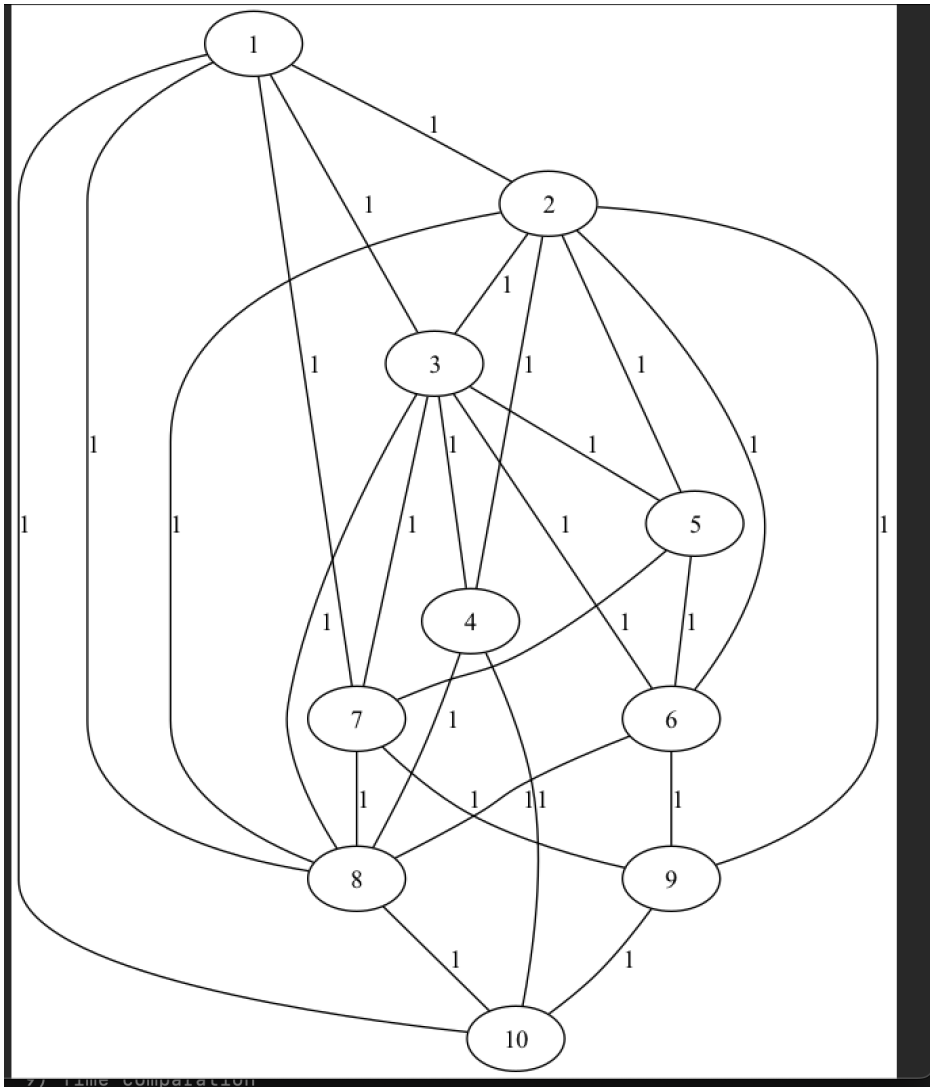
PATH FIND ANALYZE

Node count	Time(ns)	Size (bytes)
3	412	20
5	671	28
7	861	36
10	1487	48


1
1
2
8
2
3


1
1
1
2
1
3
1
1
1
4
1
5


1
3
2
2
4
3
7
1
1
1
2
5
4
5
1
1
6
2
7


1
1
2
1
1
3
1
1
1
1
1
1
1
1
5
1
1
1
4
1
1
7
1
6
1
1
1
1
1
8
9
1
1
10

Выводы по проделанной работе

Для поиска кратчайшего пути используется алгоритм Дейкстры. Он подходит под условие задачи, так как в графе не может быть отрицательных весов. Алгоритм Дейкстры не подходит для несвязного графа, и так же не подходит для графа, где есть отрицательные веса у дуг. Но, так как в моей задаче таких случаев быть не может, то данный алгоритм идеально подходит для решения моей задачи. На поиск вершины будет затрачиваться O(n) операций, а на одну релаксацию — O(1) операций, и итоговая асимптотика алгоритма составляет: O(n^2 + m). Для хранения матрицы используется матрица смежности, так как её удобно использовать для алгоритмов поиска пути.

Реальная задача в которой можно использовать реализованную программу – это посмотреть до каких пунктов может доехать курьер за путь не длинее N километров.

Контрольные вопросы

Что такое граф?

Граф – конечное множество вершин и соединяющих их ребер;

G = <V, E>. Если пары E (ребра) имеют направление, то граф называется ориентированным; если ребро имеет вес, то граф называется взвешенным.

Как представляются графы в памяти?

С помощью матрицы смежности или списков смежности.

Какие операции возможны над графами?

Обход вершин, поиск различных путей, исключение и включение вершин.

Какие способы обхода графов существуют?

Обход в ширину и обход в глубину

Где используются графовые структуры?

Графовые структуры могут использоваться в задачах, в которых между элементами могут быть установлены произвольные связи, необязательно иерархические.

Какие пути в графе Вы знаете?

Эйлеров путь, непростой путь, гамильтонов путь.

Что такое каркасы графа?

Каркас графа – дерево, в которое входят все вершины графа, и некоторые (необязательно все) его рёбра.