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

Студент **Тюликов М.В.**

Группа **ИУ7-33Б**

Предмет **Типы и структуры данных**

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

Студент _____ **Тюликов М.В.**

Преподаватель _____ **Барышникова М. Ю.**

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

Входные данные

- Команда (число) из меню программы
- Текстовый файл

Выходные данные

- Файл в DOT формате
- Файлы png
- Эйлеров путь

Возможные аварийные ситуации

- Некорректная команда
- Некорректный файл
- Ошибка памяти

Описание внутренних структур данных

```
typedef enum
{
   NO_PATH = 1,
   LIST_ALLOC_ERROR,
   LIST_IS_NULL,
   READ_ERROR,
   STRING_TOO_SMALL,
   STRING_READ_ERROR,
   STRING_TOO_BIG,
   STACK_OVER_ERROR,
   STACK_DELETE_ERROR
} errors_t;

typedef struct graph
```

```
{
    int matrix[SIZE][SIZE];
    int size;
} graph_t;

typedef struct
{
    int *m;
    size_t count;
} stack_dyn_t;
```

Константы

```
#define SIZE 100
```

Используемые функции

```
//Функция для считывания графа из файла
int read_from_file(graph_t *g);

//Функция для считывания команды с клавиатуры
int read_command(char *tmp, int *command);

//Функция поиска Эйлерова пути
int get_euler_path(graph_t *g);

//Функция вывода графа в виде картинки
int open_graph_img_src(graph_t *g);

//Функция построения остового дерева
int minimum_spanning_tree(graph_t *g);

//Функция проверки связности графа
int is_graph_connected(graph_t *g);

//Функция распечатки стека
int print_stack(stack_dyn_t *st_1);

//Функция взятия элемента из стека
int pop_m(stack_dyn_t *st_1, int *tmp);

//Функция добавления элемента в стек
int push_m(stack_dyn_t *st_1, int data);
```

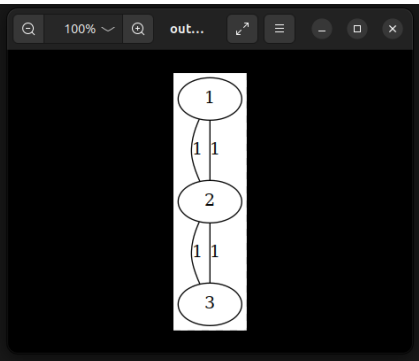
Меню программы

Программа обрабатывает нужную команду:

```
Меню команд, выберите пункт, который вас интересует.
1: Вывести граф.
2: Вывести эйлеров путь.
3: Проверить связность.
4: Построить остовое дерево минимальной стоимости.
5: Считать граф из файла.
0: Выход.
```

Пример работы программы

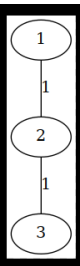
```
Меню команд, выберите пункт, который вас интересует.
1: Вывести граф.
2: Вывести эйлеров путь.
3: Проверить связность.
4: Построить остовое дерево минимальной стоимости.
5: Считать граф из файла.
0: Выход.
>1
Меню команд, выберите пункт, который вас интересует.
1: Вывести граф.
2: Вывести эйлеров путь.
3: Проверить связность.
4: Построить остовое дерево минимальной стоимости.
5: Считать граф из файла.
0: Выход.
```




```
Меню команд, выберите пункт, который вас интересует.
1: Вывести граф.
2: Вывести эйлеров путь.
3: Проверить связность.
4: Построить остовое дерево минимальной стоимости.
5: Считать граф из файла.
0: Выход.
>3
Граф связный.
Меню команд, выберите пункт, который вас интересует.
1: Вывести граф.
2: Вывести эйлеров путь.
3: Проверить связность.
4: Построить остовое дерево минимальной стоимости.
5: Считать граф из файла.
0: Выход.
>
```

```
Меню команд, выберите пункт, который вас интересует.
1: Вывести граф.
2: Вывести эйлеров путь.
3: Проверить связность.
4: Построить остовое дерево минимальной стоимости.
5: Считать граф из файла.
0: Выход.
>2
1
2
3
2
1
Время работы алгоритма(в наносекундах): 50430
Меню команд, выберите пункт, который вас интересует.
1: Вывести граф.
2: Вывести эйлеров путь.
3: Проверить связность.
4: Построить остовое дерево минимальной стоимости.
5: Считать граф из файла.
0: Выход.
>
```

```
Меню команд, выберите пункт, который вас интересует.
1: Вывести граф.
2: Вывести эйлеров путь.
3: Проверить связность.
4: Построить остовое дерево минимальной стоимости.
5: Считать граф из файла.
0: Выход.
>4
Общая стоимость остовного дерева: 2
Меню команд, выберите пункт, который вас интересует.
1: Вывести граф.
2: Вывести эйлеров путь.
3: Проверить связность.
4: Построить остовое дерево минимальной стоимости.
5: Считать граф из файла.
0: Выход.
>
```



Описание алгоритма

В программе используется представление графа в виде матрицы смежности, в виду простоты реализации, удобства представления в графическом виде, а также скорости работы.

Сначала проверим, существует ли эйлеров путь. Затем найдём все простые циклы и объединим их в один - это и будет эйлеровым циклом. Если граф таков, что эйлеров путь не является циклом, то, добавим недостающее ребро, найдём эйлеров цикл, потом удалим лишнее ребро.

Чтобы проверить, существует ли эйлеров путь, нужно воспользоваться следующей теоремой. Эйлеров цикл существует тогда и только тогда, когда степени всех вершин чётны. Эйлеров путь существует тогда и только тогда, когда количество вершин с нечётными степенями равно двум (или нулю, в случае существования эйлерова цикла).

Кроме того, конечно, граф должен быть достаточно связным (т.е. если удалить из него все изолированные вершины, то должен получиться связный граф).

Сложность этого алгоритма, является линейной относительно числа рёбер.

```
stack St;
в St кладём любую вершину (стартовая вершина);
пока St не пустой
        пусть V - значение на вершине St;
        если степень(V) = 0, то
                добавляем V к ответу;
                снимаем V с вершины St;
        иначе
                находим любое ребро, выходящее из V;
                удаляем его из графа;
                второй конец этого ребра кладём в St;
```

Выводы по проделанной работе

В ходе выполнения лабораторной работы были получены навыки работы с типом данных граф, а также получено представление о том, в каких целях он используется. Был получен опыт работы с Эйлеровыми путями, в жизни данный алгоритм может быть полезен для транспортировочных компаний, которым необходимо проложить оптимальный маршрут, проходящий по всем заданным дорогам.

Контрольные вопросы

- Что такое граф?

  Граф – это конечное множество вершин и ребер, соединяющих их.

- Как представляются графы в памяти?

  Графы в памяти могут представляться различным способом. Один из видов представления графов – это матрица смежности B(n*n); В этой матрице элемент b[i,j]=1, если ребро, связывающее вершины Vi и Vj существует и b[i,j]=0, если ребра нет. У неориентированных графов матрица смежности всегда симметрична. Во многих случаях удобнее представлять граф в виде так называемого списка смежностей. Список смежностей содержит для каждой вершины из множества вершин V список тех вершин, которые непосредственно связаны с этой вершиной.

- Какие операции возможны над графами?

  Над графами можно выполнять операции добавления/удаления вершин и ребер, поиска кратчайшего пути, определения связности графа.

- Какие способы обхода графов существуют?

  Существуют различные способы обхода графов, такие как обход в ширину (BFS) и обход в глубину (DFS).

- Где используются графовые структуры?

Графовые структуры используются в различных областях, включая информационные технологии, транспортные сети, социальные сети, биоинформатику и т. д.

- Какие пути в графе Вы знаете?

  В графе можно найти различные типы путей, такие как кратчайший путь, цикл, простой путь и др.

- Что такое каркасы графа?

  Каркас графа - это подграф, содержащий все вершины и некоторые из ребер исходного графа, образующий дерево без циклов. Каркасы используются, например, для поиска минимального остовного дерева взвешенного графа.