**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»

# ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

# «ГРАФЫ»

**по курсу «Типы и структуры данных»**

Студент: Чепиго Дарья Станиславовна

Группа: ИУ7-34Б

Студент _____ Чепиго Д.С

*подпись, дата* *фамилия, и.о.*

Преподаватель _____ Силантьева А.В.

*подпись, дата* *фамилия, и.о.*

Оценка _____

## Условие задачи

## Вариант 2

Определить, является ли связным заданный граф.

## Техническое задание

***Входные данные***

- Целое число от 0 до 5 – пункт меню

- Числа , требуемые в меню программы

Для сборки программы существует makefile. Программа «app.exe» запускается через консоль, после чего можно увидеть меню программы, состоящее из 6 пунктов

*Пункты меню:*

*1 - ввести матрицу смежности*

*2 - считать матрицу смежности из файла*

*3 - вывести матрицу смежности*

*4 - вывести граф*

*5 - определить является ли граф связным*

*0 - выход из программы*

***Выходные данные:***

- Количество вершин в графе

- Матрица смежности

- Результат проверки графа на связность

- Граф в представлении через программу Graphviz

***Описание алгоритма***

1. Пользователь выбирает пункт меню

2. Пользователь вводит количество вершин графа

3. Пользователь вводит матрицу смежности

4. Пользователь может вывести на экран матрицу смежности, граф или определить является ли граф связным

Для определения является ли граф связным используется идея , что если каждый узел может быть достигнут из вершины v, и каждый узел может достичь v, то граф связан. Для второй проверки нам нужно поменять все дуги графа на обратные, то есть транспонировать матрицу смежности. Для определение достижимости из одной вершины остальных используется алгоритм поиска в ширину, в который добавляется массив с данными о посещаемости вершин. Если после выполнения поиска в ширину какая-то вершина остается не посещенной — граф  не связан. Таким образом алгоритм определения связности графа выглядит следующим образом:

1. Проверяем на связность исходный граф

2. Если какая-то вершина недостижима —  граф не связан

3. Проверяем на связность обратный граф

4. Если какая-то вершина недостижима — граф не связан

Алгоритм поиска в ширину с использованием матрицы смежности имеет алгоритмическую сложность O(n^2). Учитывая, что мы этот алгоритм используем дважды, получается O(2*n^2). Также для проверки на связность нам необходимо дважды просматривать массив посещенных  вершин размера n, следовательно сложность алгоритма получается O(2*n^2 + 2n)

следовательно остается O(n^2). Что касается памяти, то мне приходится хранить две матрицы для моего алгоритма — матрицу смежности и обратную к ней. Поэтому в данном случае на больших данных использование матрицы смежности занимает меньше памяти и проще в использовании, в сравнении со списками, поэтому я и выбрала эту структуру данных.

Прямым применением алгоритма проверки на связность является теория сетей. Например, все компьютеры, включенные в сеть Интернет, образуют связный граф, и хотя отдельная пара компьютеров может быть не соединена напрямую  от каждого компьютера можно передать информацию к любому другому (есть путь из любой вершины графа в любую другую).

***Аварийные ситуации:***

- Выбор несуществующего пункта меню

- Ошибка выделения памяти

- Неверное количество вершин графа

- Неверная длина пути между вершинами графа

***Описание структур данных***

*Структура для реализации графа:*

```
typedef struct graph_struct graph_struct_t;

struct graph_struct
{
    int size;               // количество вершин в графе

    int **matrix;           // матрица смежности
    int **reverse_matrix;    // обратная матрица смежности
};
```

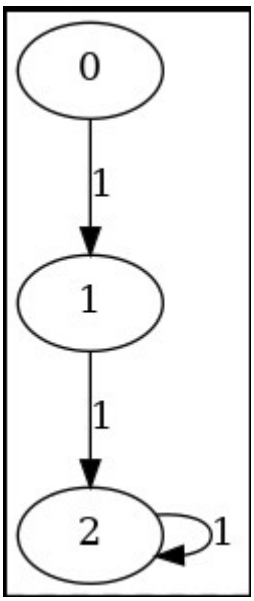
Структура для реализации очереди, используемой в поиске в ширину:

```
typedef struct queue_node_struct queue_node_t;

struct queue_node_struct
{
     int data;              // данные узла
     queue_node_t *next;    // указатель на следующий элемент
};

typedef struct queue_struct queue_struct_t;

struct queue_struct
{
     queue_node_t *start; // указатель на начало очереди
     queue_node_t *end;   // указатель на конец очереди
};
```

**Пример работы программы**

Пункт 1.Ввод матрицы смежности:
0 1 0
0 0 1
0 0 1

Граф выглядит следующим образом:



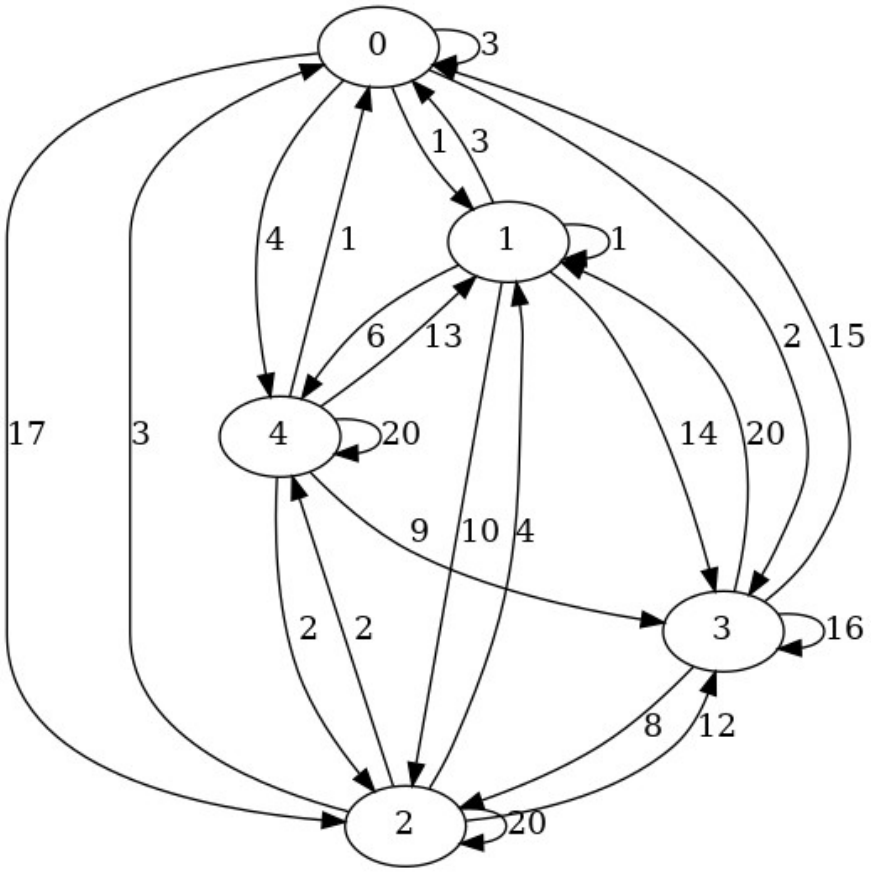

Данный граф не является связным. Если бы мы не делали проверку на связность обратного графа, то он бы считался связным, так как мой алгоритм начинается с 0 вершины.

Пункт 2. Генерация матрицы смежности:

3 1 17 2 4
3 1 10 14 6
3 4 20 12 2
15 20 8 16 0
1 13 2 9 20

Граф:

Является связным




**Тестирование**

*Позитивные тесты.*

| Входные данные | Действия программы | Выходные данные |
|---|---|---|
| Пункт меню 1<br>*Ввод матрицы смежности* | Корректная работа программы<br>Создание матрицы смежности и обратной к ней | Ожидание следующего ключа |

| | | |
|---|---|---|
| Пункт меню 2<br>Ввод корректного числа — кол-во вершин в графе | Корректная работа программы<br>Создание матрицы смежности и обратной к ней | Ожидание следующего ключа |
| Пункт меню 3<br>Если матрица смежности существует | Корректная работа программы<br>Вывод на экран матрицы смежности | Ожидание следующего ключа |
| Пункт меню 3/4/5<br>Если матрицы смежности не существует | Корректная работа программы<br>Вывод на экран информации, что надо ввести матрицу смежности в 1-2 пунктах | Ожидание следующего ключа |
| Пункт 4<br>Если матрица смежности существует | Корректная работа программы<br>Вывод на экран png изображения графа | Ожидание следующего ключа |
| Пункт 5<br>Если матрица смежности существует | Корректная работа программы<br>Вывод информации о связности графа | Ожидание следующего ключа |

*Негативные тесты*

| Входные данные | Действия программы | Выходные данные |
|---|---|---|
| Ввод числа 10 | Информация о неверном ключе меню | Завершение программы |
| Пункт 1 ввод отрицательных чисел или букв | Информация о неверном количестве вершин в графе или неверном | Завершение программы |

| | | |
|---|---|---|
| | размере пути между вершинами | |

## Контрольные вопросы

*1. Что такое граф?*

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

G = < V,E >, где V – конечное непустое множество вершин; E – множество ребер (пар вершин).

*2. Как представляются графы в памяти?*

Либо с помощью матрицы смежности, либо с помощью списка смежности.

*3. Какие операции возможны над графами?*

Вывод графа, обход графа, поиск в графе.

*4. Какие способы обхода графов существуют?*

Обход в ширину и обход  в глубину.

*5. Где используются графовые структуры?*

В системах навигации и дорог, в теории сетей.

*6. Какие пути в графе Вы знаете?*

Эйлеров путь, Гамильтонов путь, непростой путь.

*7. Что такое каркасы графа?*

Это деревья, в которые входят все вершины графа и некоторые его рёбра.