|  |  |
| --- | --- |
|  | **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**

**Вариант 3**

Название предмета: Типы и структуры данных

Студент: Баринов Никита

Группа: ИУ7-31Б

*2021г.*

**Описание условия задачи**

Найти самый длинный простой путь в графе.

**Техническое задание**

***Описание исходных данных***

Программа предоставляет пользователю выбрать различные действия с графом, которые предоставлены в меню.

Пункты меню:

1 — Добавление вершины

2 — Добавлении ребра

3 — Удалении ребра

4 — Вывод графа на экран

5 — Нахождение самого длинного простого пути

0 — Выход из приложения.

***Описание результата программы***

При запуске программы происходит чтение заранее заготовленного в файле графа. При выборе 5 пункта на экран выводится в графическом виде граф с отмеченным на нем самым длинным простым путем. Также выводится его суммарная длина. При выборе пунктов меню, отвечающих за изменение строения графа, граф меняется автоматически, но для изменения графического отображения оного, нужно еще раз выбрать соответствующие пункты меню.

***Описание задачи, реализуемой программой***

Программа выполняет обработку графа и поиск самого длинного простого пути. Для обеспечения графического вывода графа, программа описывает граф на языке DOT.

***Описание алгоритма***

Для каждой из вершин проводится следующий алгоритм.

Начиная с некоторой вершины v0, ищется ближайшая смежная ей вершина v, для которой в свою очередь осуществляется поиск в глубину до тех пор, пока не встретится ранее просмотренная вершина, или не закончится список смежности вершины v (то есть вершина полностью обработана). Если нет новых вершин, смежных с v, то вершина v считается использованной, идет возврат в вершину, из которой попали в вершину v, и процесс продолжается до тех пор, пока не получим v = v0. При просмотре используется список.

Сложность алгоритма – O(n^3)

***Способ обращения к программе***

Способ обращения к программе — консольный.

Компиляция: make app.out

Имя программы: app.out

Как запустить: ./app.out <имя файла>

Дальнейшие инструкции будут выведены после запуска.

***Описание возможных аварийных ситуаций и ошибок пользователя***

* Ошибка выделения памяти для хранения
* Неверный пункт меню.
* Некорректный ввод.

#define ALLOC\_ERROR -1

#define VERTEX\_NUM\_INPUT\_ERROR -2

#define WEIGHT\_INPUT\_ERROR -3

Во всех указанных случаях, кроме ошибки выделения памяти, программа вернет сообщение об ошибке и вернется в режим выбора пункта меню.

***Входные данные программы***

* Текстовый файл (.txt)
* Команды пользователя
* Ребра и вершины

***Выходные данные программы***

* Текстовый файл (.txt)
* Вывод в консоль – длина пути, сообщения для пользователя.

**Описание внутренних структур данных**

**1) Граф(основан на матрице смежности)**

typedef struct graph graph\_t;

struct graph

{

    int \*\*matrix;

    int \*visited;

    int vertex\_num;

};

**2) Односвязный список**

typedef struct node node\_t;

struct node

{

    int data;

    node\_t \*next;

};

**Подпрограммы и их описание**

// Создание графа

graph\_t \*create\_graph(int vertex\_num);

// Удаление графа

void delete\_graph(graph\_t \*graph);

// Ввод графа

int input\_graph(graph\_t \*\*graph, FILE \*f);

// Вывод графа в dot файл

void graphMatrixMakeDot(graph\_t \*src, char \*fileName, node\_t \*res);

// Вывод dot файла в графичесоком виде

void graphMatrixOpenDotFile(char \*fileName);

// Нахождение самого длинного простого пути в графе

node\_t \*longest\_simple\_path(graph\_t \*graph, int \*len);

// Нахождение самого длинного простого пути исходящего из заданной вершины

int longest\_simple\_path\_from\_vertex(graph\_t \*graph, int vertex, node\_t \*\*path);

// Добавление ребра

int add\_new\_path(graph\_t \*graph);

// Удаление ребра

int delete\_path(graph\_t \*graph);

// Добавление вершины

void add\_new\_vertex(graph\_t \*graph);

// Функции работы со списком

node\_t \*create\_node(int data);

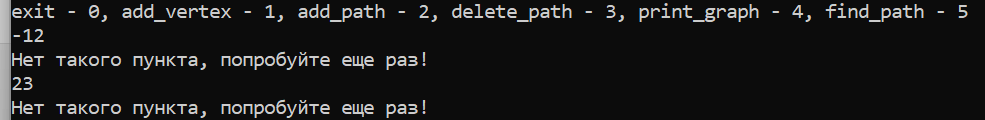
void delete\_node(node\_t \*node);

node\_t \*append\_list(node\_t \*head, node\_t \*node);

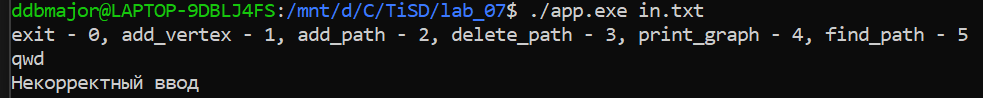
void delete\_list(node\_t \*head);

**Тестирование**

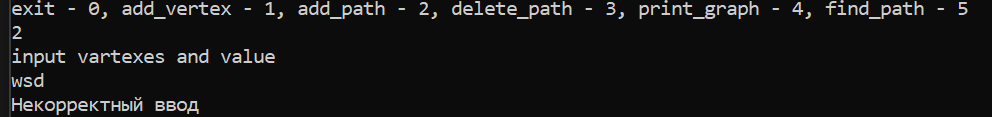
1. Ввод неверного пункта меню – “Нет такого пункта, попробуйте еще раз!”



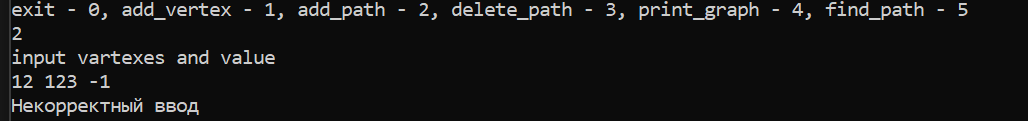
1. Некорректный ввод при выборе пункта меню(слово) - “Нет такого пункта, попробуйте еще раз!”

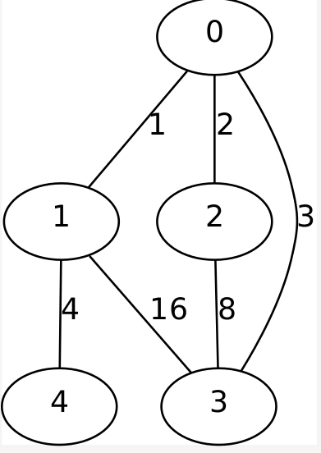
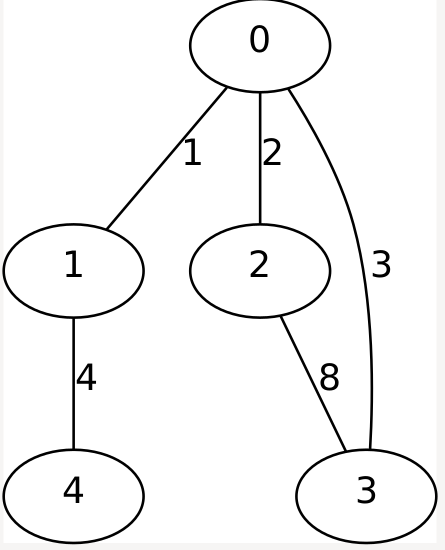
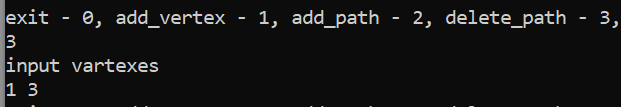


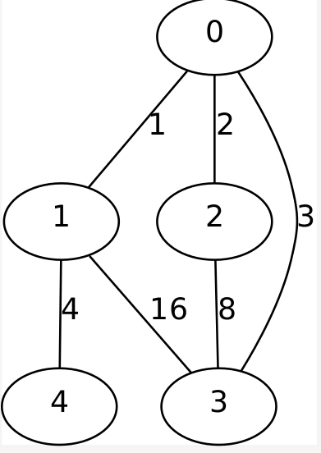
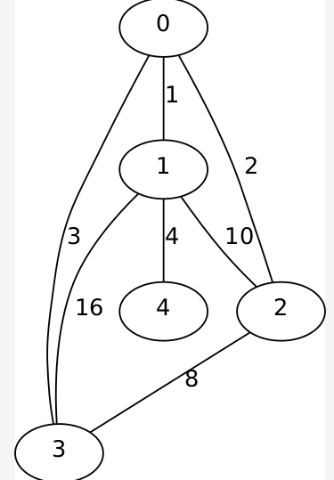
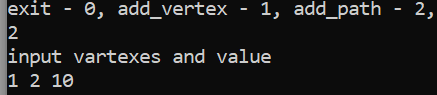
1. Неверный ввод при добавлении ребра(слово) – “Некорректный ввод ”



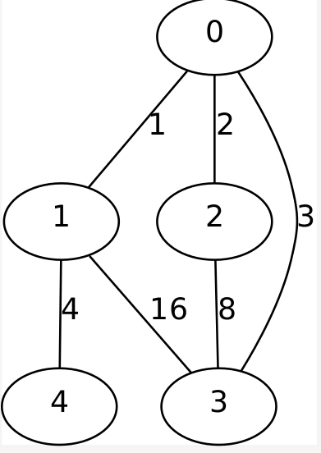
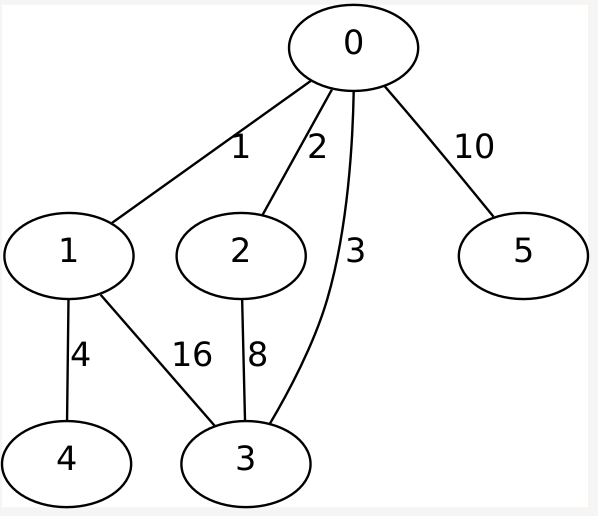
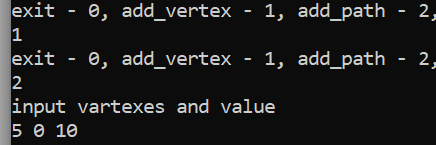
1. Попытка добавить отрицательную по номеру вершину - “Некорректный ввод ”



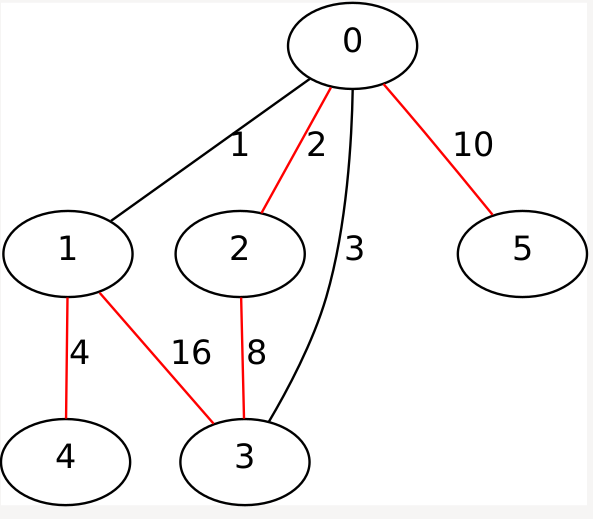
1. Удаление уже существующего ребра — “Успешно”
2. Добавление нового ребра — “Успешно”

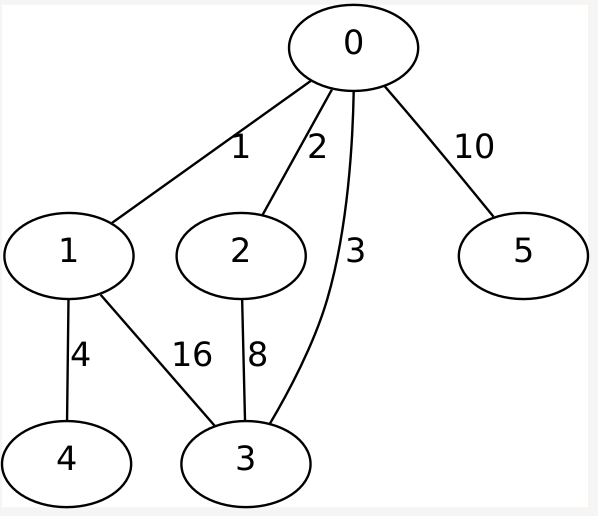


1. Добавление вершины — Успешно



1. Нахождение самого длинного простого пути





Вывод

В отличие от задачи кратчайшего пути, которая может быть решена за полиномиальное время на графах без циклов с отрицательным весом, задача нахождения самого длинного пути является NP-трудной и не может быть решена за полиномиальное время для произвольных графов.

NP-трудность невзвешенной задачи поиска самого длинного пути можно показать, сведя задачу к поиску гамильтонова пути.

Для поиска кратчайших путей используются алгоритмы Дейкстры, Беллмана-Форда, Флойда-Уоршалла. Но для нахождения самого длинного простого пути готового эффективного алгоритма нет, поэтому мной был выбран следующий способ.

Для реализации поставленной задачи был выбран алгоритм поиска в глубину с запретом посещения уже посещенных вершин, так как он позволяет решить в точности поставленную задачу.

Для хранения графа была выбрана матрица смежности, так как с помощью нее удобнее работать при реализации алгоритма поиска простого пути, минусы такого способа – большой занимаемый объем памяти.

Пример практического применения алгоритма – найти в городе такой путь, пройдя по которому, возможно посетить максимальное кол-во достопримечательностей, не повторяясь.

1. **Ответы на вопросы**

1. Что такое граф?  
Граф – конечное множество вершин и соединяющих их рёбер. Если пары ребра имеют направление, то граф называется ориентированным; если ребро имеет вес, то граф называется взвешенным.  
2. Как представляются графы в памяти?  
Существуют различные методы представления графов в программе. Матрица смежности B(n\*n) – элемент b[i,j]=1, если существует ребро, связывающее вершины i и j, и =0, если ребра не существует.  
Список смежностей – содержит для каждой вершины из множества вершин V список тех вершин, которые непосредственно связаны с ней. Входы в списки смежностей могут храниться в отдельной таблице, либо же каждая вершина может хранить свой список смежностей.  
3. Какие операции возможны над графами?  
Основные операции над графами: обход вершин и поиск различных путей: кратчайшего пути от вершины к вершине; кратчайшего пути от вершины ко всем остальным; кратчайших путей от каждой вершины к каждой; поиск эйлерова пути и гамильтонова пути, если таковые есть в графе, удаление и добавление вершин, удаление и добавления ребер.  
4. Какие способы обхода графов существуют?  
Один из основных методов проектирования графовых алгоритмов – поиск в глубину. Начиная с некоторой вершины v0, ищется ближайшая смежная ей вершина v, для которой в свою очередь осуществляется поиск в глубину до тех пор, пока не встретится ранее просмотренная вершина, или не закончится список смежности вершины v (то есть вершина полностью обработана). Если нет новых вершин, смежных с v, то вершина v считается использованной, идет возврат в вершину, из которой попали в вершину v, и процесс продолжается до тех пор, пока не получим v = v0. При просмотре используется стек.  
Поиск в ширину – обработка вершины V осуществляется путём просмотра сразу всех «новых» соседей этой вершины, которые последоватеьно заносятся в очередь просмотра.   
5. Где используются графовые структуры?  
Графовые структуры могут использоваться в задачах, в которых между элементами могут быть установлены произвольные связи, необязательно иерархические. Наиболее распространенным является использование графов при решении различных задач о путях, будь то построение коммуникационных линий между городами или прокладка маршрута на игровом поле.  
6. Какие пути в графе Вы знаете?  
Путь в графе, проходящий через каждое ребро ровно один раз, называется эйлеровым путём; путь может проходить по некоторым вершинам несколько раз – в этом случае он является непростым. Путь, проходящий через каждую вершину ровно один раз, называется гамильтоновым путём. Как эйлеров, так и гамильтонов путь могут не существовать в некоторых графах. Простой путь – путь, в котором нет повторов вершин, не простой путь – путь, в котором есть повторы вершин.  
7. Что такое каркасы графа?  
Каркас графа – подграф без циклов, в который входят все вершины графа, и некоторые (не обязательно все) его рёбра.