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

Название предмета: Типы и структуры данных

Студент: Мицевич Максим Дмитриевич

Группа: ИУ7-31Б

2020г.

I. Описание условия задачи

Заданы две системы двухсторонних дорог с одним и тем же множеством городов (железные и шоссейные дороги). Найти минимальный по длине путь из города A в город B, который может проходить как по железной так и по шоссейной дорогам, и места пересадок содного вида транспорта на другой на этом пути.

II. Техническое задание

1. *Описание исходных данных*

Программа ожидает: выбор одного из пунктов меню:

1 — чтение графа

2 — вывод графа

3 — поиск графа

а также выбор дальнейших указаний, в зависимости от выбранного пункта меню

Формат ввода:

Для ввода графа изначально требуется ввести размер графа, затем выбрать способ ввода из файла или из стандартного потока ввода, далее вводятся все ребра графа в формате:

<тип дороги> <первый узел> <второй узел> <длина пути>

окончанием считается ввод минус единицы.

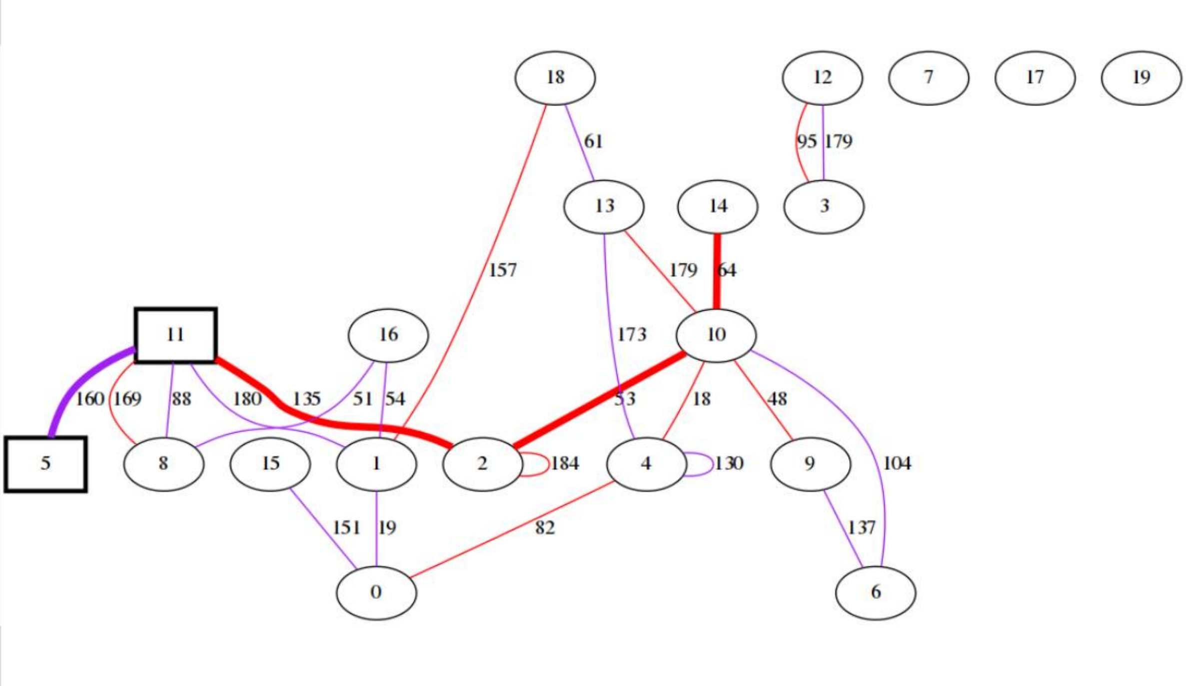
2. *Описание результата программы*

Результатом работы программы являются:

1. Граф в графическом виде
2. Маршрут с пересадками в графическом виде

Формат вывода:

Красным цветом обозначаютя дороги первого типа, филоетовым — второго. Если маршрут проходит по рербру, то оно рисуется жирным цветом. Если вершина является начальной или в ней требуется пересадка, то она рисуется в форме прямоугольника и обводится жирным цветом.



3. ***Описание задачи, реализуемой программой***

Программа находит кратчайший путь от начальной вершины до конечной и показывает вершины, в которых нужно совершить пересадку или сообщает, что такого пути нет.

4. ***Способ обращения к программе***

Способ обращения к программе - консольный. Дальнейшие инструкции будут выведены после запуска.

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

- ошибки открытия файла
- отказ операционной системы выделить память
- ввод несуществующей вершины
- отрицательный путь

Во всех указанных случаях программа сообщит об ошибке

III. Описание внутренних структур данных

Граф хранится в виде таблицы списков смежности. В каждом узле графа хранится информация требуемая для работы алгоритма по поиску кратчайшего пути.

```
typedef struct
{
    int arived;
    size_t from;
    size_t edge_type;
    size_t length;
}way_inf_t;
```

arived показывает была посещена вершина или еще нет, from — из какой вершины попали в текущую, edge_type — по ребру какого типа попали в эту вершину, length — путь от начальной вершины до текущей

```
typedef struct list_node lnode_t;

struct list_node
{
    size_t id;
    size_t cost;
    lnode_t *next;
};
```

Узел списка смежности. Id — до какой вершины идет дуга, cost — вес дуги, next — указатель на следующий элемент списка.

```
typedef struct graph_node gnode_t;

struct graph_node
{
    way_inf_t way;
    lnode_t *near_roads[ROAD_TYPES];
};
```

way — состояние о пути данного узла, near_rodes — массив указателей на списки смежности разных типов дорог

```
typedef struct
{
    gnode_t *graph;
    size_t size;
}graph_t;
```

То есть граф хранится в виде таблицы рёбер. Такое представление наиболее эффективно по времени и по памяти по сравнению, например, с таблицей смежности, ведь обычно граф не является полным, а значит таблица смежности стала бы сильно разреженной.

IV. Описание алгоритма

Для нахождения минимального пути в графе воспользуемся алгоритмом Дейкстры.

1. Сначала инициализируем нужные нам данные: ставим бесконечную длину пути во все вершины, кроме начальной, в начальную ставим 0, все вершины обозначаем, как непосещенные.
2. Просматриваем непосещенные вершины и ищем вершину с наименьшим путем до нее. Если такой вершины нет или путь до нее равен бесконечности, то выходим из алгоритма.
3. Выполняем релаксацию всех вершин, смежных с выбранной в пункте 2 по всем смежным ребрам (если путь до релаксируемой вершины больше суммы пути до вершины из пункта 2 и веса ребра, то меняем длину пути до релаксируемой вершины на эту сумму, а также запоминаем вершину, из которой пришли в релаксируемую и тип дороги, по которой это сделали). Помечаем вершину из пункта два, как посещенную, возвращаемся к пункту 2.
4. Длина пути в финальной вершине будет длиной кратчайшего пути из начальной вершины в нее (если эта длина будет равна бесконечности, то такого пути не существует). Так как в пункте 3 мы запоминали вершины, из которых мы пришли в ту или иную, и тип ребра, по которому это удалось сделать, можно восстановить через какие вершины проходит кратчайший путь и в каких узлах требуется пересадка.

Оценка по времени

Исходя из описания алгоритма, мы можем сделать вывод, что сложность алгоритма составляет: O(v * v).

Оценка по памяти

Память выделяется под каждую вершину и под каждую дугу (ребро — 2 дуги)

V. **Выводы по проделанной работе**

Для решения задачи был выбран алгоритм Дейкстры. Сравним этот алгоритм с другим алгоритмом поиска кратчайшего пути — алгоритмом Форда. Временная сложность алгоритма Дейкстры v * v, в то время как сложность алгоритма Форда — v * e, что может дать выигрыш во времени на графе, у которого много ребер. В отличие от алгоритма Форда, алгоритм Дейкстры не может обрабатывать ребра с отрицательным весом, но в условиях данной задачи это не критично, так как вес ребра является расстоянием между двумя пунктами, следовательно, не может быть отрицательным. Также стоит отметить, что алгоритм Дейкстры прост в реализации и позволяет находить не только длину кратчайшего пути, но и через какие вершины этот путь должен проходить.

VI. **Ответы на вопросы**

1. Что такое граф?

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

2. Как представляются графы в памяти?

Графы могут быть представлены в виде таблицы смежности, списков достижимых вершин из каждой вершины, массива ребёр.

3. Какие операции возможны над графами?

- поиск вершин в графе
- поиск кратчайших путей от Vk до Vm
- поиск Эйлерова пути
- поиск Гамильтонова пути
- поиск кратчайших путей между всеми вершинами

4. Какие способы обхода графов существуют?

Поиск (обход) в глубину, в ширину.

5. Где используются графовые структуры?

Схемы авиалиний, схемы дорог.

6. Какие пути в графе Вы знаете?

Простой путь, гамильтонов, эйлеров, замкнутый.

7. Что такое каркасы графа?

Каркас графа - подграф данного графа, с тем же числом вершин, что и у исходного графа. Он получается из исходного графа удалением максимального числа рёбер, входящих в циклы, но без нарушения связности графа.