|  |  |
| --- | --- |
|  | **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)** |

Отчет по лабораторной работе №7

«Графы»

Студент Ву Минь Куанг

Группа ИУ7И – 34Б

Преподаватель Силантьева А. В.

*2021 г.*

**Цель работы**

Реализовать алгоритмы обработки графовых структур: поиск различных путей, проверку связности, построение остовых деревьев минимальной стоимости.

**Условие задачи**

Обработать графовую структуру в соответствии с указанным вариантом задания. Обосновать выбор необходимого алгоритма и выбор структуры для представления графов. Ввод данных – на усмотрение программиста. Результат выдать в графической форме.

**Техническое задание**

В системе двусторонних дорог за проезд каждой дороги взимается некоторая пошлина. Найти путь из города A в город B с минимальной величиной S+P, где S - сумма длин дорог пути, а P - сумма пошлин проезжаемых дорог

**Допущения**:

* Все надписи выводятся на английском языке
* Названия городов – числа
* Пользователь не может ввести значение выше, чем половина от максимального целого
* Индексация ведется с 0

**Входные данные**

Целое число – количество вершин

Пути (из какого города в какой), расстояния между ними, пошлины дорог

Из какого города в какой требуется найти минимальный путь

**Выходные данные**

Построенный граф, минимальный путь

**Структуры данных**

В программе описывается единственна структур данных для матрицы стоимостей:

*struct table\_t*

*{*

*int size;*

*int \*\*matrix;*

*};*

где

* **int size** – размер матрицы
* **int \*\*matrix** – элементы матрицы

**Описание алгоритма**

У пользователя спрашивается количество вершин n в будущем графе. Затем выделяется память под матрицу с количеством строк n, а столбцов – в два раза больше, так как задача усложняется тем, что помимо длины пути есть значение пошлины. То есть заполняется по две ячейки сразу. Таким образом, получается матрица стоимостей, она симметрична, так как граф неориентированный, информация про одно ребро идет парами. Если путь отсутствует, то роль бесконечности в матрице выполняет значения половины от максимального целого. Для нахождения кратчайшего пути использовался алгоритм Дейкстры. Перед этим создается вспомогательная матрица стоимостей, но уже квадратная. Каждый ее элемент является суммой пары длины + пошлина. Алгоритм Дейкстры позволяет найти расстояние от одной вершины до всех других. Изначально рассматриваем всех соседей исходной вершины. Кратчайшие расстояния – это и будет прямыми путями к ним. Запоминаем их. Исходная вершина отмечается посещенной. Далее рассматриваем ближайшую не посещенную вершину и смотрим на ее соседей. Если вершина уже была посещена, то в расчет ее не берем. Находим с минимальной меткой и проверяем будет ли путь через текущую вершину короче, чем уже указанный. Если да, то заменяем. Таким образом проходим по всем не посещенным вершинам и в итоге получаем метки на каждой вершине (если она достижима) и среди них уже можем выбрать нужную. Потом нужно получить сам путь. Движемся в обратную сторону от конечной вершины. Смотрим на метки соседей и отнимаемой от текущей вес ребра по отдельности. Если получившееся значение совпало с меткой, то именно отсюда был осуществлен переход. Двигаемся так до тех пор, пока не вернемся в исходную вершину. В результате получаем массив целочисленных значений, в котором содержатся вершины минимального пути. После нахождения граф записывается в текстовый файл на языке dot и строится изображение графа при помощи GraphViz, в консоль выводится время работы, объем занимаемой памяти и информация о минимальном пути. Также на графическом представлении минимальный путь отмечен красным цветом.

В проекте содержится 10 файлов: main.c, main.h, funcs.c, funcs.h, graph.c, graph.h, messages.c, free\_messages.h, table.c, table.h. Обращение происходит с помощью стандартных потоков ввода / вывода (через консоль).

**Описание функций**

*main.c*

* *int main(void)*

Считывание данных происходит через консоль, поэтому входных данных нет, выходные – код ошибки

*messages.c*

* *void greeting(void)*

Входные, выходные данные - отсутствуют

Выводит приветственную фразу на экран

*table.c*

* *struct table\_t \*create\_table(int size, int count)*

Входные данные – количество вершин в графе; число, от которого зависит, будет ли удвоенное количество столбцов

Выходные данные – указатель на выделенную область матрицу

Выделяет память под указанное количество элементов для матрицы

* *int fill\_table(struct table\_t \*table)*

Входные данные – указатель на матрицу стоимостей

Выходные данные –код ошибки

Заполняет матрицу стоимостей, считывая при этом данные из stdin

* *void free\_table(struct table\_t \*table)*

Входные данные – указатель на матрицу стоимостей

Выходные данные – отсутствуют

Освобождает память из-под матрицы

* *void print\_table(FILE \*flow, struct table\_t \*table, int count)*

Входные данные – поток, указатель на матрицу стоимостей, число, от которого зависит, какое количество столбцов

Выходные данные – отсутствуют

Производит печать в указанный поток

* struct table\_t \*copy\_table(struct table\_t \*table)

Входные данные – указатель на матрицу стоимостей

Выходные данные – указатель на созданную матрицу

Копирует элементы из матрицы в новую, при этом складывая длину и пошлину и записывая в одну ячейку сумму

* *int \*deicstra(struct table\_t \*table, int from, int to, int \*m, int \*amount)*

Входные данные – указатель на матрицу стоимостей, пункт «из», пункт «в», указатель на количество вершин в найденном минимальном пути

Выходные данные – указатель на массив числе, которые показываются минимальный путь

Осуществляет алгоритм Дейкстры

*graph.c*

* *int find(int \*way, int k, int i, int j)*

Входные данные – указатель на массив чисел, проверяемые вершины

Выходные данные – признак

Проверяет, являются ли значения двух рядом расположенных элементов в массиве равными указанным

* *void export\_to\_dot(struct table\_t table, int \*way)*

Входные данные – указатель на матрицу стоимостей, указатель на массив чисел, в который входи минимальный путь

Выходные данные – отсутствуют

Записывает граф на языке dot в файл, строит граф с помощью GraphViz и запускает полученную картинку

*efficiency.c*

* *uint64\_t tick(void)*

Входные данные – отсутствуют

Выходные данные – число типа uint64\_t (описывается в заголовочном файле stdint.h)

Находит количество тактов процессора

* *int check\_number(int \*num, int left, int right)*

Входные данные – число, левая граница, правая граница

Выходные данные – код ошибки

Проверяет, является ли считанное число числом и входит ли оно в указанный нтервал.

**Тесты**

**Негативные тесты**

* ***Аварийные тесты при указании числа вершин***

**Число является отрицательным**

*Input amount of vertexes: -1*

*Invalid value*

**Нулевое число**

*Input amount of vertexes: 0*

*Invalid value*

**Введено не число**

*Input amount of vertexes: a*

*Invalid value*

* ***Аварийные тесты при указании значения ребер***

**Первая вершина отрицательна**

*Input amount of vertexes: 5*

*Input connections in graph (format "from to length duty"). To finish write "-1" (index from 0):*

*-2 2 1 1*

*Invalid value*

**Вторая вершина отрицательна**

*Input amount of vertexes: 5*

*Input connections in graph (format "from to length duty"). To finish write "-1" (index from 0):*

*2 -2 1 1*

*Invalid value*

**Значение вершины больше максимально возможного**

*Input amount of vertexes: 5*

*Input connections in graph (format "from to length duty"). To finish write "-1" (index from 0):*

*2 5 1 1*

*Invalid value*

**Нулевая длина дороги**

*Input amount of vertexes: 5*

*Input connections in graph (format "from to length duty"). To finish write "-1" (index from 0):*

*2 3 0 2*

*Invalid value*

**Отрицательная длина дороги**

*Input amount of vertexes: 5*

*Input connections in graph (format "from to length duty"). To finish write "-1" (index from 0):*

*2 3 -1 2*

*Invalid value*

**Длина дороги больше максимально возможного**

*Input amount of vertexes: 5*

*Input connections in graph (format "from to length duty"). To finish write "-1" (index from 0):*

*2 3 1073741824 1*

*Invalid value*

**Нулевая пошлина**

*Input amount of vertexes: 5*

*Input connections in graph (format "from to length duty"). To finish write "-1" (index from 0):*

*2 3 2 0*

*Invalid value*

**Отрицательная пошлина**

*Input amount of vertexes: 5*

*Input connections in graph (format "from to length duty"). To finish write "-1" (index from 0):*

*2 3 1 -2*

*Invalid value*

**Пошлина больше максимально возможного**

*Input amount of vertexes: 5*

*Input connections in graph (format "from to length duty"). To finish write "-1" (index from 0):*

*2 3 2 1073741824*

*Invalid value*

* ***Аварийные тесты при указании искомого пути***

**Указанный город – не число**

*Input vertex 'from' and vertex 'to': a 0*

*Invalid vertex*

*Input vertex 'from' and vertex 'to': 0 a*

*Invalid vertex*

**Указанный город – отрицательный**

*Input vertex 'from' and vertex 'to': -2 0*

*Invalid vertex*

*Input vertex 'from' and vertex 'to': 0 -2*

*Invalid vertex*

**Указанный город – больше, чем максимально возможный**

*Input vertex 'from' and vertex 'to': 6 0*

*Invalid vertex*

*Input vertex 'from' and vertex 'to': 0 6*

*Invalid vertex*

* ***Нет ни одного пути между вершинами, указанными для поиска***

*Input amount of vertexes: 4*

*Input connections in graph (format "from to length duty"). To finish write "-1" (index from 0):*

*0 1 2 4*

*1 2 4 3*

*-1*

*Input vertex 'from' and vertex 'to': 0 3*

*I can't find minimum way from 0 to 3*

*Time of working algorithm: 2940*

Memory for table: 224

**Позитивные тесты**

* ***Корректные данные, поиск минимального пути***

*Hello! This progtam can read your graph, build and find minimum way from one vertex to second*

*Input amount of vertexes: 6*

*Input connections in graph (format "from to length duty"). To finish write "-1" (index from 0):*

*0 1 1 1*

*1 2 2 1*

*2 3 11 10*

*3 5 8 2*

*3 4 4 1*

*4 0 3 1*

*0 2 4 1*

*4 2 10 1*

*-1*

*Input vertex 'from' and vertex 'to': 0 5*

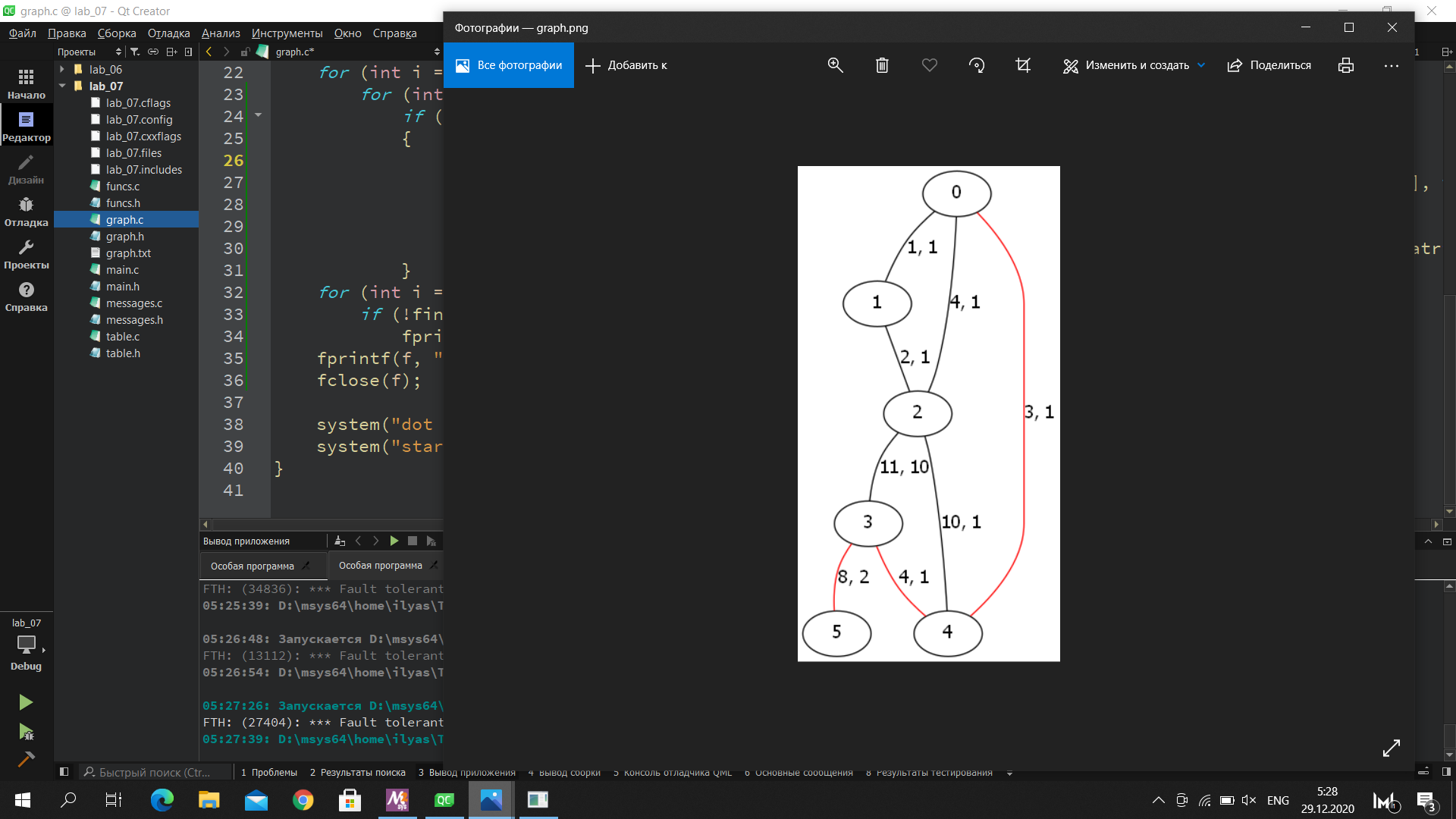
*Minimum way = 19*

*0 -> 4 -> 3 -> 5*

*Time of working algorithm: 7098*

*Memory for table: 464*

Графическое представление графа (красным помечан минимальный путь):



**Вывод**

Проведем анализ эффективности выбранного алгоритма Дейкстры. Изменяя количество вершин и ребер, проследим, как изменяется время выполнения и объем занимаемой памяти. Время измеряем в тактах процессора, память – в байтах.

|  |  |  |
| --- | --- | --- |
| Количество вершин / количество ребер | Время | Память |
| 2 / 1 | 5061 | 80 |
| 4 / 1 | 6069 | 224 |
| 4 / 3 | 6384 | 224 |
| 4 / 6 | 6678 | 224 |
| 6 / 1 | 6510 | 464 |
| 6 / 3 | 8757 | 464 |
| 6 / 6 | 11046 | 464 |
| 10 / 1 | 7245 | 1232 |
| 10 / 3 | 8169 | 1232 |
| 10 / 6 | 15981 | 1232 |

Сделаем некоторый вывод: очевидно, что граф в виде матрицы получился не таким эффективным по памяти, так как требуется дополнительная таблица для нахождения кратчайшего пути (для удобного применения алгоритма поиска кратчайшего пути) и при этом каждая таблица является симметричной, что занимает лишнюю память. Однако к плюсам такого представления можно отнести удобство использования и относительно быстрый доступ к элементам, так как он происходит по индексу. Память зависит только от количества вершин.

На время же влияет не только количество вершин, но и ребер также. Причем при малых значениях это несильно отражается на времени, тогда как при больших разница видна больше. То есть чем больше граф связан, тем быстрее растет время поиска.

Алгоритм Дейкстры выбран оправданно, так как он выполняет поставленную задачу, является удобным при использовании на выбранном представлении графа (матрице стоимостей), при малых количествах не является затратным.

**Контрольные вопросы**

1. **Что такое граф?**

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

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

1. **Как представляются графы в памяти?**

Графы в памяти могут представляться различными способами. Например, матрица стоимостей, матрица смежности, список смежности

1. **Какие операции возможны над графами?**

К основным операциями над графовыми структурами можно отнести:

* поиск кратчайшего пути от одной вершины к другой (если он есть);
* поиск кратчайшего пути от одной вершины ко всем другим;
* поиск кратчайших путей между всеми вершинами;
* поиск эйлерова пути (если он есть);
* поиск гамильтонова пути (если он есть).

1. **Какие способы обхода графов существуют?**

Существует два основных способа обхода графа: поиск в ширину (breadth first search, BFS), поиск в глубину (depth first search, DFS)

1. **Где используются графовые структуры?**

Графовые структуры используются в тех случаях, когда между элементами устанавливается некоторая связь. В качестве примера можно привести генеалогическое древо, блок-схема, схемы авиаперелетов, лабиринт и др.

1. **Какие пути в графе Вы знаете?**

Бывают пути: Эйлеров, Гамильтонов, кратчайший путь между всеми вершинами / от одной вершины к другой.

1. **Что такое каркасы графа?**

Каркас графа – дерево, в которое входят все вершины графа, и некоторые его ребра.