|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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)** |

**Дисциплина “Типы и структуры данных”**

**Лабораторный практикум №7**

**по теме: «обработка графов»**

Выполнил студент: \_\_*Клименко Алексей Константинович\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*фамилия, имя, отчество*

Группа: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*ИУ7-35Б*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил, к.п.н.: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*подпись, дата*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2020 г.*

**Цель работы**

Получить навыки применения графовых структур, освоить несколько способов представления графов в программе, реализовать алгоритмы обработки графовых структур: поиск различных путей, проверку связности, построение остовых деревьев минимальной стоимости.

**описание условия 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.

Задана система двусторонних дорог. Найти множество городов, расстояние от которых до выделенного города ( столицы ) больше, чем Т.

Результат выдать в графической форме.

**Техническое задание**

**Исходные данные**

Исходными данными являются названия городов и расстояния между ними, считанные из файла.

**Формат файла с данными:** Каждая строка в файле состоит из двух названий городов и целого числа, записанных через пробел.

Примеры корректных входных файлов:

|  |  |  |
| --- | --- | --- |
| 1 2 10 | A B 13  B C 2  C A 7  A D 8  E F 4 | Новосибирск Омск 651  Омск Томск 905  Новосибирск Томск 259  Омск Тюмень 625 |

**Результат**

Результатом работы программы является обработка и отображение неориентированного графа. Поиск и выделение городов, удаленных более чем на заданное расстояние от указанного города.

Выбранный город помечается зеленым цветом.

Найденные города помечаются красным цветом.

Недостижимые города также помечаются красным цветом.

**Выходные данные**

Выходными данными являются построенный графически граф дорог, набор искомых городов, длины путей между городами.

**Способы обращения к программе**

Для запуска программы необходимо запустить файл **app.exe**. Далее необходимо указать имя файла для загрузки графа. После этого нужно ввести название центрального города и минимальное расстояние до остальных городов.

После каждого этапа ввода рекомендуется ознакомиться с графическим представлением графа, расположенным в файле **graph.png**.

**Возможные аварийные ситуации и ошибки пользователя**

При неверном вводе имени файла программа завершает свою работу. При неверном формате входного файла программа завершает работу.

При вводе неверного названия города программа предлагает ввести имя города повторно.

**Структуры данных**

Хранение графа осуществляется с помощью списков смежности. Выбранный формат для данной задачи является более эффективным решением по сравнению с матрицей смежности, так как количество ребер в графе будет оцениваться сверху как **a\*N** (**a** - некоторая константа), что много менее **N\*N**, где **N** - число вершин (городов).

Реализация структур для хранения графа:

**struct edge // ребро графа**

**{**

**struct edge \*next; // следующее ребро в связном списке**

**struct node \*node\_from; // инцедентные вершины**

**struct node \*node\_to; //**

**int weight; // вес ребра**

**bool picked; // флаг для обработки (взяли его или нет)**

**};**

**struct node // вершина графа**

**{**

**struct node \*next; // следующая вершина в связном списке**

**struct edge \*edges; // связный список инцедентных ребер**

**const char \*name; // название города в вершине**

**int distance; // расстояние от центральной вершины**

**bool picked; // флаг для обработки (взята вершина или нет)**

**};**

**struct road\_graph // сам граф**

**{**

**struct node \*nodes; // связный список вершин графа**

**struct node \*center; // центральная вершина**

**};**

**Набор функций**

Для обработки графов в программе используются следующие функции:

**// Ищет и возвращает указатель на узел в графе с заданным именем.**

**struct node \*get\_node(const road\_graph\_t graph, const char \*name);**

**// Вставляет новую вершину в граф, если её нет.**

**struct node \*insert\_node(road\_graph\_t graph, const char \*name);**

**// Соединяет вершины ребром с заданным весом.**

**void connect\_nodes(struct node \*node\_from, struct node \*node\_to, int weight);**

**// Рассчитывает кратчайшие расстояния от центральной вершины до всех остальных.**

**void rg\_calc\_distances(road\_graph\_t graph);**

**// Устанавливает флаг у всех вершин, удалённых от центральной более чем на Т.**

**void rg\_pick\_farther(road\_graph\_t graph, const char \*center, int distance);**

**// Выводит граф в виде изображения в выходной файл.**

**void rg\_output(road\_graph\_t graph, const char \*filename);**

**// Удаляет граф и освобождает все ресурсы.**

**void rg\_destroy(road\_graph\_t \*graph);**

**Описание алгоритмов обработки данных**

Для решения основной поставленной задачи был выбран **алгоритм Дейкстры** с использованием очереди из посещаемых вершин для поиска кратчайших расстояний от выбранной вершины до всех остальных.

Данный алгоритм имеет временную сложность **O(N\*logN + M)**, где **N** -число вершин в графе, а **M** - число ребер, и подходит для обработки графов с неотрицательными весами.

**Алгоритм поиска кратчайших расстояний**

**G: road\_graph - исходный граф**

**V: queue - очередь посещенных вершин (пустая)**

**curr: node - текущая обрабатываемая вершина**

**начало**

**для всех вершин node в G:**

**node.distance := INT\_MAX;**

**конец для;**

**G.center.distance := 0; { стартуем из центральной вершины }**

**curr := G.center;**

**пока curr <> NULL:**

**для всех ребер edge из curr.edges:**

**если edge.node\_to.distance > curr.distance + edge.weight:**

**edge.node\_to.distance := curr.distance + edge.weight;**

**добавить в очередь V вершину edge.node\_to;**

**конец если;**

**конец для;**

**curr := извлечь вершину из очереди V;**

**конец пока;**

**конец.**

**Набор функциональных тестов**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Описание теста** | **Входные данные** | **Выходные данные** |
| 1 | Неверное имя файла | a.txxt | Сообщение о неверном имени файла |
| 2 | Пустой файл | empty.txt | Сообщение о том, что файл пуст |
| 3 | Неверный формат файла | wrong.txt | Вывод сообщения о неверном формате файла |
| 4 | Граф с двумя вершинами | double.txt | Вывод графа с одной дорогой |
| 5 | Граф с городами России | russia.txt | Нормальное выполнение программы |
| 6 | Ввод некоректного имени центрального города | russia.txt  Омскъ 1200 | Вывод сообщения о неверном названии города и приглашение к повторному вводу |
| 7 | Ввод некоректного расстояния | russia.txt  Омск -1200 | Вывод сообщения о некорректном расстоянии и приглашение к повторному вводу |
| 8 | Ввод нулевого расстояния | russia.txt  Омск 0 | Норальный вывод графа |

Рассмотрим работу программы на простых примерах. В качестве удобства, во всех примерах центральным выбирается город "А", а расстояние от него до остальных городов вводится 10.

|  |  |
| --- | --- |
| Начальный граф | Граф с вычисленными расстояниями и выбранными городами |
|  |  |
|  |  |
|  |  |
|  |  |

**Выводы по проделанной работе**

Хранение графа в виде списков смежности подходит для задач, в которых средняя степень вершины не будет зависеть от числа вершин. Или, другими словами, число ребер будет намного меньше, чем половина от квадрата числа вершин.

Выбирать матрицу смежности для хранения графа оправдано в тех случаях, когда число ребер будет близко к числу ребер в полном графе.

Выбор конкретного алгоритма обработки графа зависит от его структуры. При большом числе вершин, но малом числе ребер следует выбирать алгоритмы с временной сложностью O(N\*M) (алгоритм Беллмана - Форда), O(N\*logN + M) (алгоритм Дейкстры с использованием доп. структур). А если же обрабатываемый граф содержит ребра практически между всеми парами вершин, то будут хорошо работать алгоритмы со сложностью O(N\*N) (алгоритм Дейкстры со списком), O(N\*N\*N) (алгоритм Флойда-Уоршелла) и O(N\*M + N\*N\*logN) (алгоритм Джонсона).

**Контрольные вопросы**

1. *Что такое граф?*

Граф - это конечный набор вершин и соединяющих их рёбер. Если ребра имеют направление, то граф называется ориентированным.

2. *Как представляются графы в памяти?*

В зависимости от поставленной задачи графы могут представляться в виде списка вершин и матрицы смежности этих вершин, либо с помощью списков смежности. Выбор зависит от соотношения между количеством вершин и количеством дуг в графе.

3. *Какие операции возможны над графами?*

В графах можно производить поиск отдельных вершин или рёбер, находить кратчайшие расстояния между вершинами, определять компоненты связности графа, а также его ацикличность.

4. *Какие способы обхода графов существуют?*

Обход в глубину (DFS - Depth First Search). При таком обходе, начиная с произвольной вершины v0 ищется ближайшая смежная вершина v, для которой, в свою очередь, осуществляется поиск в глубину (т.е. снова ищется ближайшая, смежная с ней вершина) 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.

Обход в ширину (BFS - Breadth First Search). Обработка вершины v осуществляется путем просмотра сразу всех новых соседей этой вершины. При этом полученный путь является кратчайшим путем из одной вершины в другую.

5. *Где используются графовые структуры?*

Графовые структуры активно применяются для представления сетевых отношений в базах данных. Также распростанено их пременение в нейронных сетях.

6. *Какие пути в графе Вы знаете?*

Путь - последовательность вершин графа, соедененных ребрами.

Простой путь - путь без повторения вершин.

Цепь - путь без повторения ребер.

Простая цепь - цепь без повторения вершин.

Цикл - путь, начальная и конечная вершина которого совпадают.

Простой цикл - цикл, который не проходит дважды по одной вершине.

Гамильтонов путь (цикл) - простая цепь (простой цикл), содержащая(ий) все вершины графа без повторений.

7. *Что такое каркасы графа?*

Каркас графа (остовное дерево) - связный подграф, содержащий все вершины графа и не имеющий циклов. Количество ребер в каркасе связного графа всегда на единицу меньше количества вершин графа.