МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Построение и анализ алгоритмов»

Тема: Коммивояжер

Студентка гр. 1304 _____ Хорошкова А.С.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург

2023

Цель работы.

Построение гамильтонова цикла в *ориентированном* графе.

Задание.

Дана карта городов в виде ассиметричного, неполного графа $G = (V, E)$, где $V(|V|=n)$ – это вершины графа, соответствующие городам; $E(|E|=m)$ – это ребра между вершинами графа, соответствующие путям сообщения между этими городами.

Каждому ребру m_{ij} (переезд из города iв город j) можно сопоставить критерий выгодности маршрута (вес ребра) равный w_i(натуральное число [1, 1000]), $m_{ij}=inf$, если $i=j$.

Если маршрут включает в себя ребро m_{ij}, то $x_{ij}=1$, иначе $x_{ij}=0$.

Требуется найти минимальный маршрут (минимальный гамильтонов цикл): $\min W = \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}\, w_{ij}$

Выполнение работы.

Описание алгоритма.

Сначала считывается граф в структуру List< PriorityQueue<Edge>> так, чтобы каждая вершина являлась индексом для очереди с исходящими из неё рёбер. В свою очередь рёбра заполняют очередь с приоритетом так, чтобы наверху всегда было ребро с наименьшим весом. Засекается время старта. После чего с начальной вершины (по умолчанию 1) запускается «коммивояжер», который ищет гамильтонов цикл. С помощью алгоритма бэктрекинга ищется минимальный цикл. Фиксируется время конца алгоритма. Печатается результат.

В качестве оптимизаций алгоритма бэктрекинга реализованы следующие условия для продолжения алгоритма по текущему ребру:

1. Длина текущего пути меньше ранее найденного минимального.
2. Размер минимального остовного дерева не превышает разницы между ранее найденным минимальным и текущим размером пути.

Описание функций и структур данных.

Класс, реализующий алгоритм, назван Salesman. Класс имеет следующие поля. graph — ориентированный граф, заданный в формате, описанном при описании метода printResult(). START_BACKTRACKING_VERTEX — вершина, с которой начинается алгоритм. way — итоговый путь. currentWay —

текущий путь во время работы алгоритма. minDistance — длина минимального пути. currentDistance — текущая длина пути во время работы алгоритма. time — затраченное время на работу алгоритма. graphSize — размер графа. Хранится для оптимизации времени.

Внутри класса определён внутренний класс Edge, реализующий интерфейс Comparable<Edge>, — класс, хранящий одно исходящее ребро. Класс имеет поля to — конечная вершина, weight — вес ребра, конструктор, принимающий на вход оба поля, и переопределённый метод compareTo(), сравнивающий рёбра по их весу.

Класс имеет следующие методы. Saleman() — конструктор без параметров, считывает граф с помощью метода readGraph(). После запускает алгоритм для поиска цикла с помощью метода runSalesman(). В конце печатает результат с помощью метода printResult(). readGraph() — считывание графа в формате <начало графа> <граф в табличном виде>. runSalesman() — запуск алгоритма поиска минимального гамильтонова пути с помощью функции backtracking(int vertex). Фиксация времени работы алгоритма. printResult() — вывод результата работы программы в консоль. backtracking(int vertex) — алгоритм бэктрекинга для поиска минимального по весу пути, начинается с переданной в качестве параметра вершины. Оптимизация при помощи остовного дерева реализуется с помощью метода getSpanningTreeSize(HashSet<Integer> usedVertex, int startVertex). getSpanningTreeSize(HashSet<Integer> usedVertex, int startVertex) — поиск минимального остовного дерева для вершин, не входящих в usedVertex, начиная со startVertex. saveWay() — сохранение текущего пути как результата. Итоговый результат сохраняется как объект класса StringBuilder для оптимизации времени.

Также создан публичный метод Main, содержащий единственный метод main() – точку входа в программу. В методе main() создаётся новый экземпляр класса Saleman.

Тестирование.

Ниже представлена таблица с данными для тестирования (Таблица 1). В Левом столбце представлены входные данные (input), в правом столбце представлены выходные данные (output).

Таблица 1. Примеры тестовых случаев

input:	output:
6 *inf 1 1 1 1* *1 inf 1 1 1* *1 1 inf 1 1* *1 1 1 inf 1* *1 1 1 1 inf* *1 1 1 1 1 inf*	*Way: 1-2-6-5-4-3-1* *Distance: 6* *Time: 8mls*
20 *- 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1* *1 - 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1* *1 1 - 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1* *1 1 1 - 1 1 1 1 1 1 1 1 1 1 1 1 1 1* *1 1 1 1 - 1 1 1 1 1 1 1 1 1 1 1 1 1* *1 1 1 1 1 - 1 1 1 1 1 1 1 1 1 1 1 1* *1 1 1 1 1 1 - 1 1 1 1 1 1 1 1 1 1 1* *1 1 1 1 1 1 1 - 1 1 1 1 1 1 1 1 1 1* *1 1 1 1 1 1 1 1 - 1 1 1 1 1 1 1 1 1* *1 1 1 1 1 1 1 1 1 - 1 1 1 1 1 1 1 1* *1 1 1 1 1 1 1 1 1 1 - 1 1 1 1 1 1 1* *1 1 1 1 1 1 1 1 1 1 1 - 1 1 1 1 1 1* *1 1 1 1 1 1 1 1 1 1 1 1 - 1 1 1 1 1* *1 1 1 1 1 1 1 1 1 1 1 1 1 - 1 1 1 1* *1 1 1 1 1 1 1 1 1 1 1 1 1 1 - 1 1 1* *1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 - 1 1* *1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 - 1* *1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 -*	*Way: 1-2-20-19-18-17-16-15-14-13-12-11-10-9-8-7-6-5-4-3-1* *Distance: 20* *Time: 10mls*

Продолжение Таблица 1

input:	output:
5 *inf 1 - 1 1* *1 inf - - 1* *1 - inf 1 1* *1 - - inf 1* *1 - - 1 inf*	*There is no way*
3 *- 1000 1* *1 - 1* *1 1 -*	*Way: 1-3-2-1* *Distance: 3* *Time: 1mls*
20 *inf 12 65 2 2 16 15 67 38 17 1 74 68 21 13 26 85 7 21 20* *5 inf 8 63 14 10 46 11 5 22 3 27 57 55 62 25 45 14 50 63* *14 68 inf 24 16 28 19 17 19 6 11 93 15 63 42 23 58 73 91 67* *12 50 75 inf 87 62 89 21 60 41 45 89 68 35 32 9 16 88 23 75* *84 19 89 90 inf 93 69 52 71 3 62 62 23 71 77 93 68 24 20 38* *17 77 48 19 70 inf 22 43 5 63 83 78 10 25 91 8 39 79 35 50* *8 29 95 19 94 15 inf 20 60 79 43 32 5 7 61 60 49 30 25 15* *7 1 76 60 64 20 1 inf 12 4 42 15 75 10 34 71 9 35 69 79* *23 41 90 38 81 68 22 49 inf 91 87 30 58 1 6 47 48 6 100 78* *21 20 72 67 43 22 30 78 50 inf 22 47 26 66 72 59 11 47 30 41* *5 60 98 97 34 45 7 55 1 47 inf 8 47 38 35 97 15 53 61 95* *64 51 21 64 75 92 64 21 68 66 56 inf 70 25 77 84 55 87 82 48* *45 2 49 54 81 34 9 97 18 76 43 40 inf 54 46 22 77 1 84 42* *4 63 93 4 73 13 73 66 73 17 95 10 29 inf 1 27 71 11 85 5* *4 80 81 11 76 68 83 28 67 16 45 79 1 84 inf 74 81 19 15 26* *20 54 17 89 16 8 56 18 42 84 20 83 76 62 61 inf 84 30 34 20* *17 12 71 62 41 46 12 42 96 37 34 25 46 53 36 11 inf 13 81 49* *54 70 11 4 75 28 20 60 24 9 63 76 10 61 8 28 30 inf 25 4* *63 85 47 77 90 32 4 29 16 31 82 11 76 71 33 92 70 8 inf 6* *22 29 72 5 31 43 50 22 95 63 87 52 72 40 5 94 2 41 16 inf*	*Way: 1-11-9-14-15-19-12-8-2-3-5-10-17-7-13-18-20-4-16-6-1* *Distance: 151* *Time: 2822mls*

Выводы.

В ходе лабораторной работы был реализован поиск минимального гамильтонова цикла (задача коммивояжера) с помощью алгоритма бэктрекинга.

В ходе тестирования все графы обрабатывались менее, чем за 3 секунды благодаря введённым оптимизациями (сравнение текущей длины пути и минимального, поиск минимального остовного дерева для оставшихся вершин для оценки минимальной длины продолжения пути).

Алгоритм верно нашёл кратчайший гамильтонов цикл для всех примеров, использовавшихся при тестировании. При отсутствии каких-либо гамильтоновых циклов в графе программа определяла это и выводила соответствующее сообщение.