GTU Department of Computer Engineering

CSE 222/505 - Spring 2022

Homework #08 Report

Şiyar Tarık Özcaner

20010400421

1. System requirements

In this assignment there is 3 parts, first one asks for a list-type graph implementation which incorporates Vertex class. Second question asks for a comparison between BFS and DFS, difference of total weight achieved by mentioned traversal methods, and lastly an alternative version of djikstra’s algorithm is asked for us to implement.

1. Class diagrams

* Is in the directory

1. Problem solutions approach

Q1:

Well in MyGraph, I have two data structures for vertices named “vertices” and edges named “edges” with these names and their types are ArrayList<Vertex> and ArrayList<LinkedList<Vertex>> respectively. “vertices” structure is straight-forward and holds vertexes of the graph, “edges” holds interestingly Vertex objects to present edge, the reason for this is that I thought reason of “weight“data field in Vertex class was meant to represent relations between vertices (this created some extra problems that wasted a big chunk of my time, I wished I knew that “weight” was there for no specific reason ☹ before I started coding this project). And it works by following principle:

Index of the LinkedList is source and the id of the Vertex object in the Linkedkist is destination.

Implementation is done by the principle above. For example if an edge from 0 to 2 is going to be deleted that is found by checking 0th LinkedList of “edges” is iterated to find a Vertex with the id of 2.

When an edge is deleted “removedV” is incremented with this data field, uniqueness of vertices is achieved with the “numV” + “removedV” operation.

Theoretical Time Complexity:

1. newVertex: Θ(1)

2. addVertex: Θ(1)

3. addEdge: Θ(1)

4. removeEdge: Θ(1)

5. removeVertex (int vertexID): O(E) since “edge” is looked for any edge that goes to vertex to remove

6. removeVertex(int vertexLabel): O(E) same with the method above

7. filterVertices: O(V+E) vertices are removed filtered (V) then removed using removeVertex method (E)

8. exportMatrix: Θ(V+E) because for every call iterates through “edges” structure

9. printGraph: Θ(V+E) because for every call iterates through “edges” structure

Q2:

Here, very standardized graph search algorithms of Depth first search and breadth first search algorithms are implemented and with the help of “edges” structure weights of edges are summed for both algorithms and subtracted for the difference. Here the caveat is done by sorting edges from smallest to biggest in “getEdges” method.

Theoretical Time Complexity:

BFS and DFS both have the complexity of O(E) since nested loops in both of the algorithms look for every edge a vertex has for all of the vertices which results in processing of all of the edges.

Initializations of variables take Θ(V)

Bubble sort is used for sorting which is O(E^2) time complexity-wise this adds up to O(E^2)

Q3:

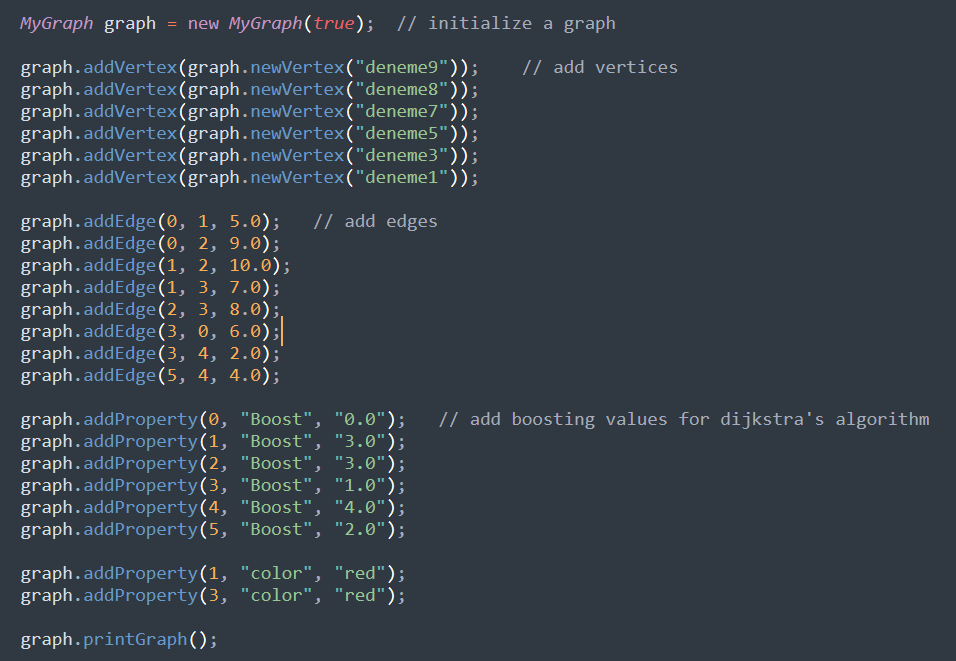
Again, a standard implementation of Dijkstra’s algorithm is implemented here with the caveat of “Boosting”, boosting values are added to weight calculations.

Theoretical Time Complexity:

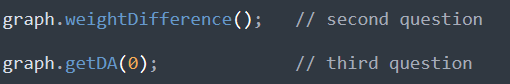
Nested loops of the algorithm results of a O(V^2) time complexity because for every vertex algorithm checks if other vertices are within the reach.

1. Test cases

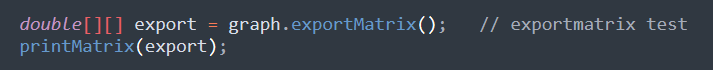
Directed graph is initialized with six vertices and printed



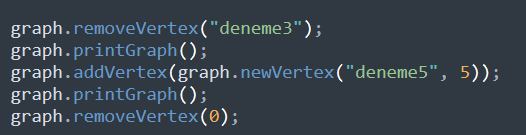
For second and third question



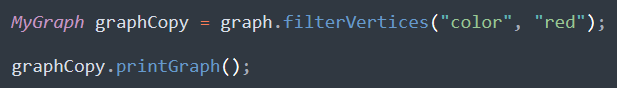
graph is exported as a matrix



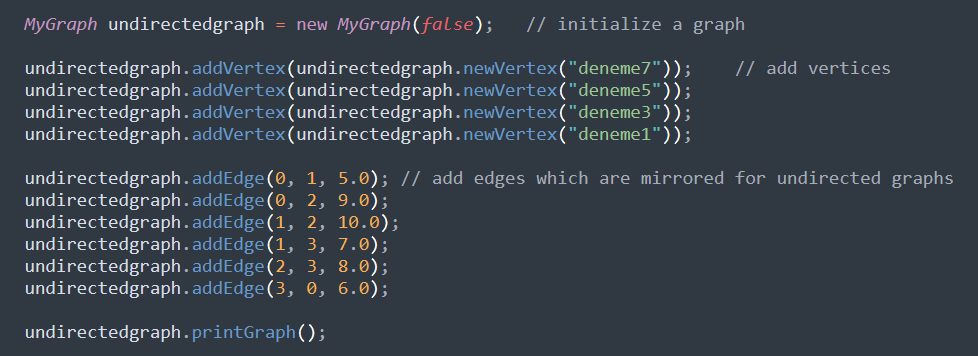
Remove method are tested



Graph is filtered



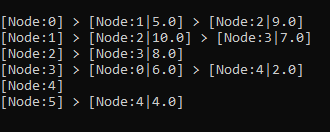
Undirected graph is created



1. Running command and results

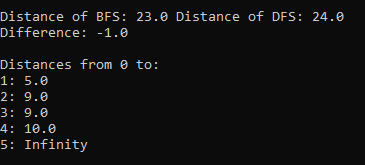
After a simple “make” command in the terminal these are the result:

Graph is printed after all of the add operations

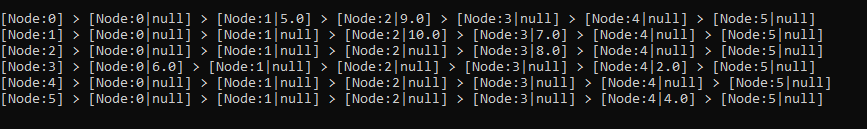


Here we can see difference between BFS and DFS:

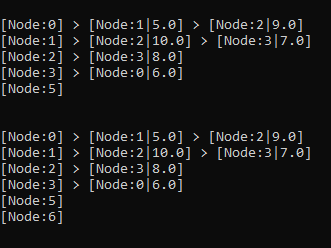
And Shortest distances from 0 to other vertices:



Matrix export is printed



Remove methods tested and add method is used here for prove of uniqueness



Graph is filteder with the key of “color” and value of “red”



Undirected graph example

