In this Project we will improve the project that we built in project 0, which there, we implemented and made tests on unidirectional unweighted graph.

In this project our graph will be unidirectional but this time it will be weighted.

We add some new methods that doesn’t written in the interface. The method equals- that will be used to examine equality between different graphs. And the method compare- which implement the priority queue that we built, in order to find the shortest path distance and some more constructors.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Class WGraph\_DS\*\*\*\*\*\*\*\*\*\*\*\*

I defined unidirectional weighted graph and then I defined the different methods on it.

I added a node, connected the nodes (added edge), I removed edge, removed node, returning the edge length, and method that return the number of nodes in the graph, and the number of edges in the graph. In addition I defined a parameter that count all the changes that I made in the graph.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*WGraph\_Algo\*\*\*\*\*\*\*\*\*\*\*\*

In this class we made different methods on the graph that we built, and I will elaborate on every action.

\*\*Init\*\*

Making a pointer for the graph that we built (shallow copy).

\*\*\*copy\*\*\*

We will go through all the nodes in the graph with the iterator and will duplicate the values from the old graph: tag, info, key, weight. In addition we will copy all the neighbors of every node and we will put them in the new graph according the node's key

\*\*isConnected\*\*

In order to check if the graph is connected or not, we will use the data base: ArrayDeque<node\_data>.

I "painted" all the graph in white, I defined a certain node which from it I will start my test, and I will go through all the nod's neighbors and "paint" them in black, and every time I "paint" a node with black I will add 1 to the counter I defined in advance.

And then, by using the queue, I will go through the neighbors of the neighbors and etc. and in every node I go to, I will "paint" it in black so I wont count it again. In the end I will get the number of nodes and then I will check if the number of nodes is equal to the counter I defined in advance.

If they equal, then it means the graph is connected and if they not then the graph isn’t connected.

\*\*\*shortestPathDist\*\*\*

For the implementation of this method we will use the data structure: PriorityQueue which we will implement in the Class: WGraph\_DS.

In order to find the shortest path between two nodes in a weighted graph, we will define the tag of each node as MAX\_VALUE except of the src which we will defined it’s tag as 0. We will add the src to the PriorityQueue that we built and then we will go through his neighbors with for each. Afterward we will check if the tag of the neighbor is higher than the tag of the vertex that we hold now plus the length of the edge between them, if it is then we will replace the tag of the neighbor for the tag of the vertex we hold plus the length of the edge between them, then we will add the neighbor with the lowest tag to the PriorityQueue and so on (on all the graph). And eventually we will know for sure the lowest tag we can get for the dest by using this method which will be the shortest path distance between the dest and the src.

\*\*\*shortestPath\*\*\*

In this metho we will use the data structure: LinkedList <node\_info>, to keep the keys of the vertexes in the shortest path distance.

We will receive from the method shortestPathDist the length of the shortest path, and because we updated the neighbor with the tag of the vertex we been hold plus the edge between them only if the neighbor's current tag is higher , then we can start the return of the path from the dest by going to the neighbor which his tag plus the edge between them is equal to the tag of the vertex we hold and so on, until we get to the src.

In the end we will return the path as LinkedList.

\*\*\*save\*\*\*

In this method we will save our graph in text file on our pc by using java's functions.

By using FileOutputStream we will save the graph in text file on the pc, but this method save the file in language which isn’t text language.

So we will use the function ObjectOutputStream in order to translate what we get from FileOutputStream to a text.

\*\*\*load\*\*\*

In this method we will do the opposite action from that we did in "save". Firstly we will translate the text file to a language which it possible to convert to programming language and then we will translate this to java.