Assignment 3

Here is the report belonging to pa3 for 1DT907 course from Samuel Berg

How to run the code

Instead of being `java#` as it is on linux and mac, for windows you would need to add the `.java` after the file you want to run.

```
javac *.java

java Main1
java Main2
java Main3
java Main4
java Main5
```

Problem 4

Djikstra's Algorithm

```
Dijkstra's Shortest Distances sample:
From 0 to 8729: 5.329343291714084
From 0 to 4916: 5.580331178322289
From 0 to 8863: 5.869060149692333
From 0 to 6545: 5.752871187232946
From 0 to 4: 1.549113175557646
From 0 to 1838: 5.529777702868398
From 0 to 8775: 5.3976037503422525
From 0 to 1581: 5.79784568415894
From 0 to 7099: 6.0953901166444515
From 0 to 690: 6.189950574655911

Dijkstra's Time: 13.9578 ms
```

Bellman-Ford Algorithm

```
Bellman-Ford's Shortest Distances sample:
From 0 to 5608: 5.020887020182483
From 0 to 4103: 6.236187629593231
From 0 to 470: 7.487941643279421
From 0 to 8411: 6.062208995135986
From 0 to 4706: 5.124226916858716
From 0 to 2379: 7.228916440418932
From 0 to 1044: 6.454888444866436
```

```
From 0 to 5588: 7.511343191375932
From 0 to 1112: 5.748846376855659
From 0 to 5097: 7.293874740141204

Bellman-Ford's Time: 4.6775 s
```

Conclusion

Note: `V = # vertices` and `E = # edges` in the following text.

The above shown samples are from a run with `10 000` vertices and `20 000` edges. Which shows that Djikstra's algorithm is faster than the Bellman-Ford algorithm. This holds true in all of the test cases I have provided the algortihms. The reason to Dijkstra's algorithm being faster than the Bellman-Ford algorithm is because Dijkstra's algorithm greedily selects the shortest path from the current node to all other nodes in the graph and it never revisits nodes once they have been processed. This gives Dijkstra's algorithm a time complexity of `O((E + V) * log V)`. Meanwhile Bellman-Ford algorithm iterates over all edges multiple times to find the shortest path. This means it has to iterate over all edges `V - 1` times to gain the shortest path. This gives it the time complexity of `O(V * E)` in it's worst case.

Note: Djikstra's algorithm can't handle negative edge weights meanwhile Bellman-Ford can.

If we then do the calculation with the time complexity for each algorithm, we can then see that Djikstra's algorithm is faster than Bellman-Ford's algorithm. Djikstra's algorithm has a time complexity that results to `O((20 000 + 10 000) * log 10 000)` = `O(120 000)` meanwhile Bellman-Ford's algorithm has a time complexity that results to `O(20 000 * 10 000)` = `O(200 000)`, this for the above given example. if we compare these `Djikstra's vs Bellman-Ford` we get `O(120 000) < O(200 000)` which means Djikstra's algorithm is faster than the Bellman-Ford algorithm.

```
To explain this more theoreticaly this is due to Djikstra's being a so
called "greedy" algorithm and due to it's time complexity being `O((E + V)
* log V)`, which means it has an logarithmic increasing time complexity.
Meanwhile Bellman-Ford due to being able to handle negative edge weights
has a time complexity of `O(V * E)`. Which leads to the times mentioned in
the sample outputs are reasonable for the respective algorithms.
```