Complexity analysis LAB 3 DAT037

Ali Mohamed & Ali Mohamud

Complexity analysis

Our implementation of Dijkstra's algorithm utilizes priority queue in Java Collection Framework, in which the underlying algorithm is a heap. Another important implementation of the Java Collections framework is the intensive use of HashMap. The source code further down the page, only include operations where the execution time differs from constant or is critical to the complexity analysis. Let the notation for edge and node be, E and V.

The while-loop on row 2 will run as many times as there are edge in the queue. Therefore a complexity of O(E). The size of the queue is limited by the number of add operations in row 6.

The poll process on row 3 will have a complexity of O(log E), since it's a priority queue. The queue will contain edges that's added in row 6. But since the operation will execute every time the while-loop runs, the complexity will be O(E log E). But since we can disregard multi-graph structure the number of elements in the queue can be simplified to V^2. Thus a final complexity to O(E log V^2) which is the same as O(E log V)

The for-loop on row 5 will only execute one time for every node, but the amount of iterations for every node is limited by the edges connected to that given node. The total number of iterations will be limited by the amount Edges in total. Therefore a complexity of O(E).

The execution time for adding element to the priority queue is O(log V) since the upper bound is limited by V^2. But since the add method will be invoked for every iteration in the loop, the complexity will be O(E log V).

In order to run computePath(), the method initiate() must be called to set the appropriate values must be set to null, false and infinity. This method has a complexity of O(V).

The total complexity is equal to O(V) + O(E(log V)+E(log V)) which is the same as O(V+E(log V))

```
1. public void computePath(){
2.         while(!prio.isEmpty()){
3.                 E currentNode = prio.poll();
4.                 if(!nodeMapVisited.get(currentNode)){
                           ...
5.                         for(Edge<E> e: graph.getConnections(currentNode)){
                                   ...
6.                                 prio.add(neighbour);
                           }
7.                         ...
                   }
8.         }
}
```