

BLACKBOXAI

Generated on: 18/07/2025 13:50:43

User

```
public List computeShortestPath (int sourceNode) {  
  
    int[] distance = new int[this.numberOfNodes]; // Keeps track of the distance of each node from source node. Arrays.fill(distance,  
    Integer.MAX_VALUE); // Assume the maximum distance is infinity. distance[sourceNode] = 0; // The distance from source node to source  
    node is always zero(0).  
  
    // Min Heap implementation of priority queue. Queue minHeap = new PriorityQueue<>((nwp1, nwp2) -> nwp1.edgeWeight() -  
    nwp2.edgeWeight()); minHeap.add(new NodeWeightPair(sourceNode, 0)); // The distance from source node to source node is always  
    zero(0).  
  
    while (!minHeap.isEmpty()) {  
  
        NodeWeightPair poppedPair = minHeap.remove(); int currentNode = poppedPair.node(); int distanceOfCurrentNodeFromSourceNode =  
        poppedPair.edgeWeight(); if (distanceOfCurrentNodeFromSourceNode > distance[currentNode]) continue; List neighbourPairList =  
        this.adjacentList.get(currentNode); for (NodeWeightPair neighbourPair : neighbourPairList) {  
  
            int neighbourNode = neighbourPair.node(); int distanceOfNeighbourNodeFromCurrentNode = neighbourPair.edgeWeight();  
  
            int distanceOfNeighbourNodeFromSourceNode = distance[currentNode] + distanceOfNeighbourNodeFromCurrentNode;  
  
            if (distance[neighbourNode] > distanceOfNeighbourNodeFromSourceNode) { distance[neighbourNode] =  
            distanceOfNeighbourNodeFromSourceNode; minHeap.add(new NodeWeightPair(neighbourNode, distance[neighbourNode])); } }  
        }  
  
        // Processing and returning result ... List resultList = new ArrayList<>(); for (int node = 0; node < this.numberOfNodes; node++)  
        resultList.add(new Distance(sourceNode, node, distance[node]));  
  
        return resultList;  
    }  
}
```

Generated by BlackBox AI

```
}
```

I believe the below mentioned line in the above code is unimportant, unnecessary and redundant. It's simply not required. Can you plz confirm?

if (distanceOfCurrentNodeFromSourceNode > distance[currentNode]) continue;

BLACKBOXAI

Yes, you are correct in identifying that the line

```
java
if (distanceOfCurrentNodeFromSourceNode > distance[currentNode]) continue;
```

is unnecessary in the context of Dijkstra's algorithm as implemented in your code.

Here's why:

- 1. Priority Queue Behavior: The priority queue (min-heap) is designed to always return the node with the smallest edge weight (distance) first. Therefore, when you pop a node from the priority queue, it is guaranteed that the distance of that node from the source is the shortest known distance at that point in the algorithm.
- 2. Distance Array: The `distance` array keeps track of the shortest known distance from the source node to each node. If you are processing a node (i.e., popping it from the priority queue), it means that the distance you have for that node is already the shortest distance. Thus, checking if `distanceOfCurrentNodeFromSourceNode` is greater than `distance[currentNode]` is redundant because it will always be false for the node being processed.

Removing this line will not affect the correctness of the algorithm, and it may slightly improve performance by avoiding unnecessary checks.

User

Can you piz revise the code snippet with removing that unnecessary line?

Generated by BlackBox AI

BLACKBOXAI

Here is the revised version of your `computeShortestPath` method with the unnecessary line removed:

```
java
public List computeShortestPath(int sourceNode) {
    int[] distance = new int[this.numberOfNodes];
    Arrays.fill(distance, Integer.MAX_VALUE);
    distance[sourceNode] = 0;

    Queue minHeap = new PriorityQueue<>((nwp1, nwp2) -> nwp1.edgeWeight() - nwp2.edgeWeight());
    minHeap.add(new NodeWeightPair(sourceNode, 0));

    while (!minHeap.isEmpty()) {
        NodeWeightPair poppedPair = minHeap.remove();
        int currentNode = poppedPair.node();

        List neighbourPairList = this.adjacentList.get(currentNode);

        for (NodeWeightPair neighbourPair : neighbourPairList) {
            int neighbourNode = neighbourPair.node();
            int distanceOfNeighbourNodeFromCurrentNode = neighbourPair.edgeWeight();
            int distanceOfNeighbourNodeFromSourceNode = distance[currentNode] +
                distanceOfNeighbourNodeFromCurrentNode;

            if (distance[neighbourNode] > distanceOfNeighbourNodeFromSourceNode) {
                distance[neighbourNode] = distanceOfNeighbourNodeFromSourceNode;
                minHeap.add(new NodeWeightPair(neighbourNode, distance[neighbourNode]));
            }
        }
    }

    List resultList = new ArrayList<>();
    for (int node = 0; node < this.numberOfNodes; node++) {
        resultList.add(new Distance(sourceNode, node, distance[node]));
    }
}
```

```
    return resultList;  
}
```

Key Changes:

- 1. Removed the redundant check: Eliminated

```
java  
    if (distanceOfCurrentNodeFromSourceNode > distance[currentNode]) continue;
```

as it serves no purpose.

- 2. Rest of the logic remains the same: The algorithm's functionality is unchanged and still correctly implements Dijkstra's shortest path algorithm.

What Were You Trying To Achieve with the Check?

- If you were attempting to improve efficiency by eliminating unnecessary neighbor processing, a better approach would be:

```
java  
    if (poppedPair.edgeWeight() > distance[currentNode]) continue;
```

However, even this is redundant due to the min-heap property, as explained earlier.

This version is clearer and slightly more efficient without the unnecessary check. Let me know if you'd like any further refinements!