Tutorial course 4: Algorithms for traversing a graph and shortest paths Part 2

Guillaume Lachaud
Arnaud Lobera Boucheron

Contents

List of Figures

List of Listings

Breadth Search First (BFS) for shortest paths in unweighted (di)graphs

As the implementation of the graph has been designed to be usable on directed or undirected graphs, weighted or unweighted graphs, there is no change needed to the code of the section **Breadth Search First algorithm (BFS)**. For completeness, the code is added in this tutorial.

In an unweighted graph, using BFS with a source node gives the shortest path from the source node to all the other nodes, because each time it marks a node, it records (in the variable **edgeTo**) the last node between the source node and the marked node.

Given below are the variables of the different classes we are using:

```java
public class Node<T> implements Comparable<Node<T>> {
  T nodeName;
  List<Edge> neighbors;
  // ...
}
```

Listing 1: Node class

```java
public class Edge {
  Node tail;
  Node head;
  double weight;
  // ...
}
```

Listing 2: Edge class

```java
public class GraphList<T> {
  private int nodeCount;
  private int edgeCount;
  private ArrayList<Node<T>> adj;
  // ...
}
```

Listing 3: GraphList class

When parsing a file, the function **fillAdjList** in the GraphList class looks for a third value in a line of the file used to create the graph. If there is one, it is used as the weight of the edge. If not, we use the value 1.

```
public class BreadthFirstPaths<T> {
  private Map<Node, Boolean> marked;
  private Map<Node, Node> edgeTo;
  private final Node s;
```

Listing 4: BreadthFirstPaths class

Manipulating Weighted digraphs

As was previously remarked, the graph adjacency list has been designed to be able to accept directed/undirected, weighted/unweighted graphs, so we use the same code as before.

To create a new GraphList object with a directed graph, we can use the following constructor of the GraphList class:

```
public GraphList(String file, String splitDelimiter, boolean isDirectedGraph)
 ↪  throws java.io.IOException {
  createGraph(file, splitDelimiter, isDirectedGraph);
}
```

Listing 5: GraphList constructor to call to use directed graphs

Dijkstra algorithm for weighted digraphs

The following implementation is, like the BFS and DFS implementation, heavily influenced by [1].

We create the DijkstraSP class like so:

```
public class DijkstraSP<T> {

  private Map<Node, Edge> edgeTo;
  private Map<Node, Double> distTo;
  PriorityQueue<Node<T>> pq;
  private final Node s;
  // ...
}
```

Listing 6: DijkstraSP class

The **verifyNonNegative** function is defined as follows:

```
  public boolean verifyNonNegative(GraphList G) {
    Iterator<Edge> edgeIterator = G.getAllEdges().iterator();
    while (edgeIterator.hasNext()) {
      if (edgeIterator.next().getWeight() < 0) {
        return false;
      }
    }
    return true;
  }
```

Listing 7: **verifyNonNegative** function in the DijkstraSP class

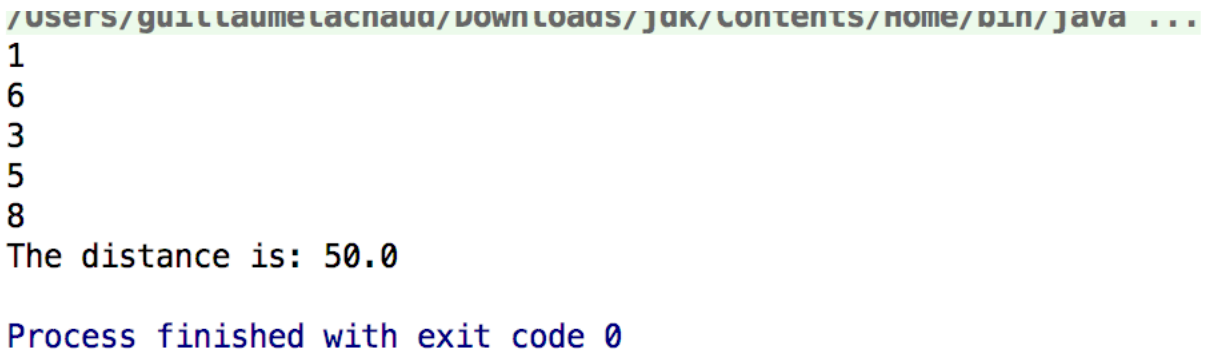
The rest of the implementation can be looked up in the code.

To find the shortest path between two nodes using Dijkstra algorithm, we can use the code implemented in the main of **DijkstraSP** (see Listing 8)

The result appears on the following figure:

```
public static void main(String[] args) throws java.io.IOException {
    GraphList graph = new GraphList("src/graph-WDG.txt", " ", true);
    DijkstraSP graphComponent = new DijkstraSP(graph, graph.getNode("1"));
    Stack<Node> nodeStack = graphComponent.shortestPathTo(graph.getNode("8"));
    while (!nodeStack.isEmpty()) {
      System.out.println(nodeStack.pop().getNodeName());
    }
    System.out.println("The distance is: " +
     ↪  graphComponent.distTo(graph.getNode("8")));
  }
```

Listing 8: Testing the Dijkstra shortest path algorithm



```
/Users/guillaumelachaud/Downloads/jdk/Contents/Home/bin/java ...
1
6
3
5
8
The distance is: 50.0

Process finished with exit code 0
```

Figure 1: Test of the Dijkstra algorithm with nodes 1 and 8.

References

[1] R. Sedgewick and K. Wayne, *Algorithms.* Addison-Wesley Professional, 4th ed., 2011.