AG2411 – GIS Architecture and Algorithms
Takeshi Shirabe
Geoinformatics, KTH
2021-12-13

Exercise 5
Implementing a shortest path algorithm

In the previous exercise, you created the **Node, Arc, and Network** classes, which enabled you to read and write ASCII files of network data. In this exercise, you will extend these classes by adding the capabilities 1) to find a shortest path out-tree from an origin to all other nodes and 2) to find a shortest path from an origin to a destination.

PREPARATION

Stay in the same directory that you designated as your working directory in Exercise 4.

TASKS

1. Add to the **Node** class a new attribute called **prev**. The **Node** class is now structured as follows.

Node
+ **name: String** + **value: double** + **outArcs: LinkedList** + **prev: Node**
+ **Node(String)**

<Attributes>

- The **prev** attribute is the *Node* that immediately precedes **`this`** *Node* on a shortest path out-tree. When a node is instantiated, assign **`null`** to this attribute (also ***`POSITIVE_INFINITY`*** to the **weight** attribute), because a shortest path (tree) has yet to be found.

  Note: Alternatively, you may want to design the **prev** attribute to be ***Arc***, in case that the input network contains parallel arcs. In what follows, we assume such a case.

2. Add to the **Network** class two new methods, which are both called **dijkstra**. The **Network** class is now structured as follows.

Network
+ **name: String** + **nodeMap: HashMap**
+ **Network(String, String)** + **save(String): void** + **printNodes(): void** + **printArcs(): void** + **dijkstra(Node): void** + **dijkstra(Node, Node): void**

<Methods>

- The first **dijkstra** method takes a *Node* representing an origin as input, applies Dijkstra's algorithm to **this** *Network* to find a shortest path tree from the origin, sets the **value** of each *Node* to the shortest path distance, and sets the **weight** of each *Arc* to 1 if it is part of the shortest path tree and to 0 otherwise.
- The second **dijkstra** method takes two *Nodes* representing an origin and a destination as input, applies Dijkstra's algorithm to **this** *Network* to find a shortest path from the origin to the destination, sets the **value** of each *Node* to the shortest path distance, and sets the **weight** of each *Arc* to 1 if it is part of the shortest path and to 0 otherwise.

Your code for the first **dijkstra** method may look like…

```java
public void dijkstra(Node origin) {

      // Set the value (representing shortest path distance) of origin to 0
      origin.value = 0;

      // Create a set of nodes, called tempNodes, whose shortest path distances
      // are not permanently determined. Initially, this set contains all nodes.
      List<Node> tempNodes = new ArrayList<Node>();
      for (String nodeName: nodeMap.keySet()) {
            tempNodes.add(nodeMap.get(nodeName));
      }

      // Perform Dijkstra
      while (! tempNodes.isEmpty()) { // repeat until all nodes become permanent

            // Find a node with minimum value in tempNodes

            // Update the value of each node that is adjacent to the minimum-
            // weight node

            // Remove the minimum-value node from tempNodes

      }

      // Assign 1 to all arcs in the shortest path tree and 0 to all other arcs.
}
```

3. Compile the **Node, Arc, and Network** classes.
4. Create a class called **Ex05** that consists only of the **main** method that takes as input three or four *Strings* representing the name of an input file, the name of an output file, the name of an origin node, and optionally the name of a destination node, and does the following.
   - If the last parameter is not inputted, construct a *Network* from the input file, apply the first **dijkstra** method to it, save the resulting *Network* (which represents a shortest path tree from the origin) in the output file, print the value of each *Node* (which represents its shortest path distance from the origin), and print the weight of each *Arc* (which equals to 1 if that *Arc* belongs to the tree and to 0 otherwise).
   - If the last parameter is inputted, construct a *Network* from the input file, apply the second **dijkstra** method to it, save the resulting *Network* (which represents a shortest path from the origin to the destination) in the output file, print the value of each *Node* (which represents its shortest path distance from the origin), and print the weight of each *Arc* (which equals to 1 if that *Arc* belongs to the path and to 0 otherwise).

5. Compile and test the **Ex05** class.

HELPS

1. Visiting all arcs in a network

You may find the following method useful, which visits all arcs and do something.

```
// Assign 0 to all arcs
private void clearArcWeight() {
      Node n;
      for (String nodeName: nodeMap.keySet()) {
            n = nodeMap.get(nodeName);
            for(Arc a: n.outArcs){
                  a.weight = 0;
            }
      }
}
```

2. Tracing a path starting with a specified node

On the completion of Dijkstra's algorithm, a shortest path (tree) is represented in such a way that each node (except the origin) points to one and only one *Arc* that terminates at that node. With this information, the following method starts with the node specified by its parameter and traces a sequence of arcs *backwards* until the origin is reached.

```
// Assign 1 to all arcs that precede a specified node along a path
private void trace(Node n) {
      Arc a = n.prev;
      while (a != null) {
            a.weight = 1;
            a = a.tail.prev;
      }
}
```

3. Calling a method from another method

You may think that you have to write whole new code for the second **dijkstra** method. Notice, however, that it is almost identical with the first **dijkstra** method, because Dijkstra's algorithm is designed to find shortest paths from one node to all other nodes even if you are only interested in one path connecting two specific nodes. So just call the first dijkstra method from the second dijkstra method.

This is an example of calling a method from another method.

```
public void dijkstra(Node origin, Node destination) {
	dijkstra(origin); // or this.dijkstra(origin);
	clearArcWeight();
	trace(destination);
}
```

SUBMISSION

Submit the following.
- Source codes of **Node**, **Arc**, **Network**, **and Ex05** classes.

DUE DATE

To avoid penalty points, your submission must be made through **CANVAS** no later than 11:59 PM, **December 23, 2021**.