2020

SOFTENG 364: Lab 6
Routing – Version 2

Craig Sutherland

Contents

Overview

By the end of this lab you should be able to:

1. Explain what NetJSON is and why it is used,
2. Visualise a network from a NetJSON definition,
3. Implement Dijkstra's algorithm in Java.

To receive credit for completing the worksheet, please complete the Lab 6 Quiz on Canvas. You'll receive feedback immediately afterwards, and may choose to redo the quiz if you wish. This worksheet and the associated quiz comprise Lab 6 and contribute 1% to the final mark. The due date (for the quiz) is 11:59 pm, Friday, 22nd May.

Several activities on the lab worksheet are framed as "questions", but responses needn't be submitted (i.e. the on-line quiz is the only submission required). Nonetheless, please don't hesitate to speak to a member of the 364 team during the lab if you are unsure of what a suitable response might be.

Preparation

The software we need this week is installed in the Engineering computer labs. If you're working on your own PC, please install the Java SE 8u251 (https://www.oracle.com/java/technologies/javase-downloads.html) and Eclipse[1] (https://www.eclipse.org/downloads/).

There is also a resource file on Canvas that contains template code for you to use in this lab. Download the code and extract it into a folder on your machine.

You may like to make a new git repository in which to track your SOFTENG 364 lab and assignment work.

Graph representation in JSON

NetJSON is a proposed interchange/file format for network entities.

Visit http://netjson.org/ and briefly (<3 minutes) consider the following questions.

Questions

Record the answers for these questions. You will use these results to answer the questions on the Canvas quiz.

What is JSON?

What types of network-related entities does NetJSON support? Which one is used to encode network graphs?

What identifiers are available for nodes?

How are attributes associated with nodes and links?

Task 1

Encode the network below in NetJSON format, including its link costs. Save your file as `Lab-6.json` . You will use this file later in this lab.

[1] You can use an alternate IDE if desired (e.g. IntelliJ IDEA or Apache NetBeans). However this manual assumes you are using Eclipse, as this is installed on the lab computers.

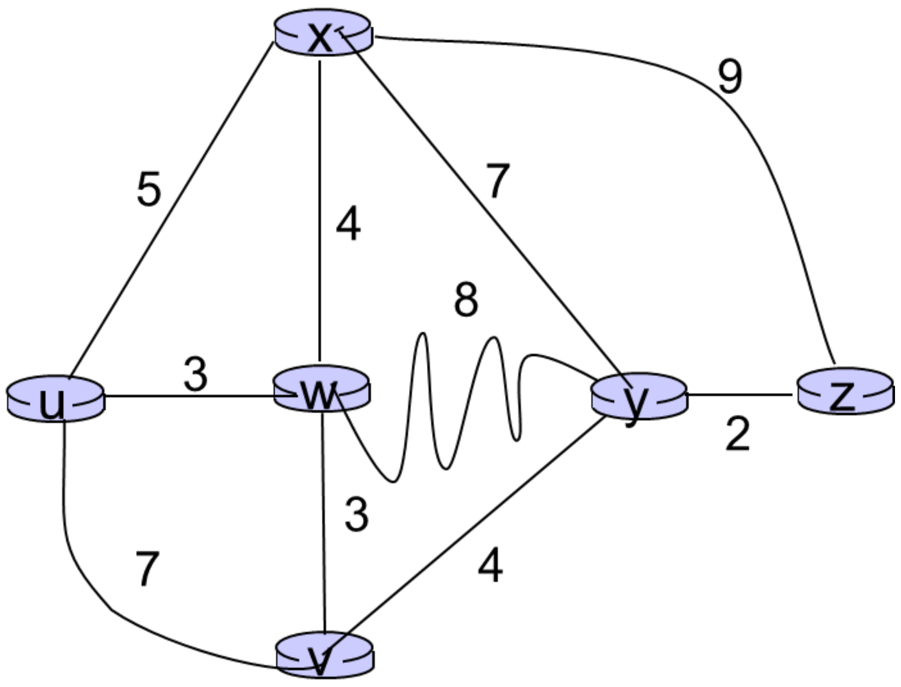


Figure 1: Network example

Loading NetJSON data

The next task is to read the file and import it into your code. As Java does not come with built-in support for either JSON or NetJSON, we will use a third-party jar to handle the parsing of JSON and build our own implementation. The specific library we will use is json-simple. Go to https://code.google.com/archive/p/json-simple/downloads and download json-simple-1.1.1.jar to a folder on your machine.

Next start Eclipse and create a new project:

1. Start Eclipse
2. Click on "Create a project…"
3. Select Java Project from the list and click on "Next >"
4. Type in SOFTENG364-Lab-1 as the project name and click on "Finish"
5. Click on "Don't Create"
6. Your project is now ready

The next step is to import the jar file and use it:

7. Add a new folder by right-clicking on the project name and selecting New → Folder
8. Type in lib as the folder name and click on "Finish"
9. Copy the jar file into the new folder (this can be done via drag-and-drop or copying via the file system)
10. Add the jar file to the build path by right-clicking on the jar file (in the project explorer), then selecting Build Path → Add to Build Path
11. The jar file will now be included in the Referenced Libraries node of the project tree
12. Add the following files to the project and compile it:
    - NetJSONReader.java: *main entry point for your project*
    - Network.java: *contains the details on a network*
    - NetworkNode.java: *contains the details on a node in the network*

This should load in the file you just wrote and display the number of nodes:

```
Loading Network graph from C:\Users\csut017\eclipse-workspace\SOFTENG364-Lab-1\Lab-
6.json
Network graph has been loaded
Loaded 6 nodes
```

If you get an error saying the file cannot be found, check that Lab-6.json is in the root folder of your project:

```
Loading Network graph from C:\Users\csut017\eclipse-workspace\SOFTENG364-Lab-1\Lab-
6.json
Exception in thread "main" java.io.FileNotFoundException: C:\Users\csut017\eclipse-
workspace\SOFTENG364-Lab-1\Lab-6.json (The system cannot find the file specified)
	at java.base/java.io.FileInputStream.open0(Native Method)
	at java.base/java.io.FileInputStream.open(FileInputStream.java:219)
	at java.base/java.io.FileInputStream.<init>(FileInputStream.java:157)
	at java.base/java.io.FileInputStream.<init>(FileInputStream.java:112)
	at java.base/java.io.FileReader.<init>(FileReader.java:60)
	at NetJSONReader.main(NetJSONReader.java:15)
```

Questions

Record the answers for these questions. You will use these results to answer the questions on the Canvas quiz.

What are the minimum attributes needed to record an edge?

What are the names of the attributes in NetJSON?

Task 2: Handle link elements

Extend the supplied code to links. You will need to add a new class called NetworkLink to your project, then modify the Parse() method in Network to handle links.

To test your code is working, display the number of nodes loaded at the main() in NetJSONReader.

Visualising a network graph

Now that we can load a network graph, let's visualise it. Again, Java does not contain any packages in the standard library for visualising graphs, but we can download a library that will visualise it for ius. The library we will use is GraphStream. Go to http://graphstream-project.org/download/ and download gs-core (Release 1.3).

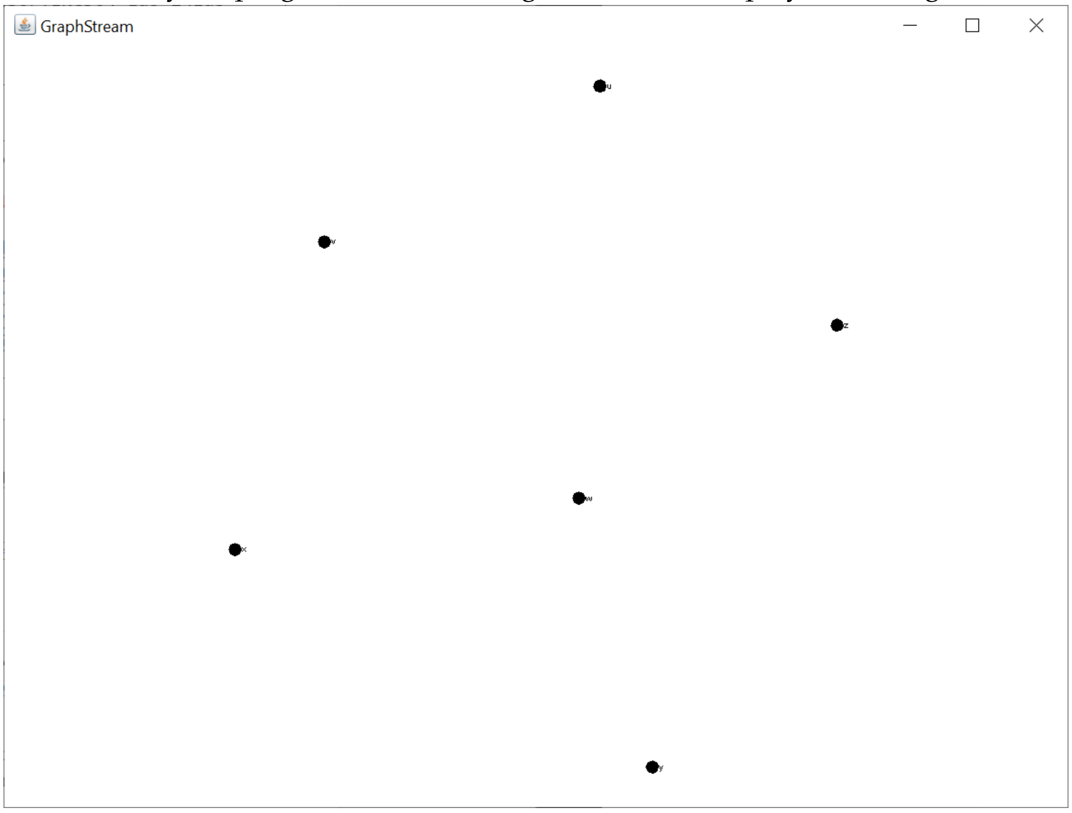
1. Copy gs-core-1.3.jar to the lib folder in your project and add it to the build path.
2. Open NetJSONReader.java and add the following code to the bottom of the main() method:

```
System.out.println("Generating graph");
// Start a new graph and add each node
Graph graph = new SingleGraph("Lab 1");
network.getNodes().forEach(node -> {
    // Retrieve the node ID and use this as the ID in the graph
    String nodeId = node.getId();
    Node n = graph.addNode(nodeId);       // Adds the node
    n.addAttribute("ui.label", nodeId);   // Sets the UI label so we can see the ID
                                          // in the visualisation
});
graph.display();
System.out.println("Graph generated");
```

3. Then add the following import lines to the top of the file:

```
import org.graphstream.graph.*;
import org.graphstream.graph.implementations.SingleGraph;
```

4. Compile and execute your program. You should get a window displayed looking like this:



GraphStream is a powerful library that allows you to visualise all types of graphs. In this lab we are only going to use the basic functionality in it. If you want to explore more, read the online documentation at http://graphstream-project.org/doc/.

Task 3: Visualise edges and their costs

Currently the code only displays the nodes in the network. You will need to expand the code to add the edges (links between nodes).

To add an edge use the following method:

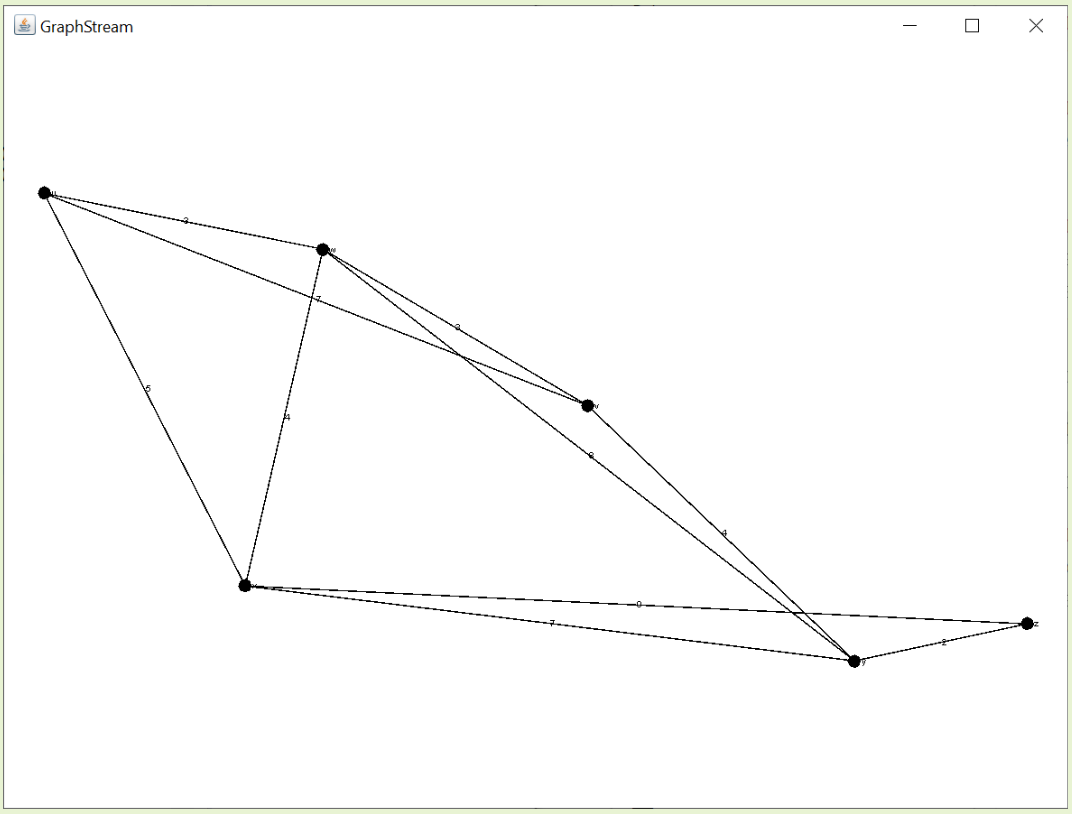
```
graph.addEdge("AB", "A", "B");
```

Where:

AB is the identifier of the edge
A is the first node
B is the second node

Use the `ui.label` attribute to display the edge cost. To change the length of the edges use the `layout.weight` attribute.

Your final graph should look like:



Implementing Dijkstra's algorithm

We can now visualise a graph (network) and see how all the nodes connect. The next challenge is to calculate the lowest cost route between a source node and all other nodes in the network.

1. Add the following files to the project and compile it:
    - DijkstrasAlgrorithm.java: *skeleton implementation of Dijkstra's algorithm*
    - NetworkNodeComparator.java: *a comparator for NetworkNode instances – used to sort nodes*
    - NodeSetDisplay.java: *an interface for displaying a set of nodes*
2. Open NetJSONReader.java and add the following code to the bottom of the main() method:

```
DijkstrasAlgrorithm al = new DijkstrasAlgrorithm();
al.initialise(network);
al.calculate("u", al);
```

3. Compile and execute your program. In the console output, you should get the following:

```
Exception in thread "main" java.lang.Exception: Source node not found
      at DijkstrasAlgrorithm.calculate(DijkstrasAlgrorithm.java:41)
      at NetJSONReader.main(NetJSONReader.java:46)
```

You are getting this exception because the code has not been completed (it is just a skeleton of the algorithm.) In particular, the `initialiseNodesForCalculate()` method has not been implemented.

4. Open DijkstrasAlgrorithm.java and find the `initialiseNodesForCalculate()` method and replace the current code with the following:

```
NetworkNode nodeToRemove = null;
for (NetworkNode node: _nodes) {
    if (node.getId().equals(source)) {
        node.setCost(0);
        node.setPredecessor(source);
        nodeToRemove = node;
    } else {
        node.setCost(Long.MAX_VALUE);
        node.setPredecessor("");
    }
}
return nodeToRemove;
```

5. This code requires some getters and setters on the NetworkNode class that do not exist. Open NetworkNode.java and implement the following getter and setter methods (you will need to write these yourself):
   - `long getCost()`
   - `void setCost(long cost)`
   - `String getPredecessor()`
   - `void setPredecessor(String predecessor)`
6. Compile and execute your program. In the console output, you should get the following:

```
Node:  u      v      w      x      y      z
#0:    Inf.   Inf.   Inf.   Inf.   Inf.   Inf.   u
#5:    Inf.   Inf.   Inf.   Inf.   Inf.   Inf.   u
```

As you guessed it, you are getting this output because there are more un-implemented methods in DijkstrasAlgrorithm.java.

Task 4: Implement the missing methods in DijkstrasAlgrorithm.java

In DijkstrasAlgrorithm.java, implement the following methods (there are already stubs in place for them):

`findLowestCostNode()`: find the node with the lowest current cost. This method retrieves the set of nodes that need to be checked (unvisited nodes.)

`updateNodeCosts()`: update all the costs of the nodes that have not been visited yet. You will need to iterate through all the edges (links between nodes) and find the edges that connect to the node. If the cost to the current node plus the edge cost is lower than the current cost, then update both the current cost and the predecessor.

You will also need to modify the `display()` method to retrieve both the current cost and the predecessor for a node.

Once you have modified the code you should get the following output:

```
Node: u     v     w     x     y     z
#0:   0,u   7,u   3,u   5,u   Inf.  Inf.  u
#1:   0,u   6,w   3,u   5,u   11,w  Inf.  uw
#2:   0,u   6,w   3,u   5,u   11,w  14,x  uwx
#3:   0,u   6,w   3,u   5,u   10,v  14,x  uvwx
#4:   0,u   6,w   3,u   5,u   10,v  12,y  uvwxy
#5:   0,u   6,w   3,u   5,u   10,v  12,y  uvwxyz
```

Questions

Record the answers for these questions. You will use these results to answer the questions on the Canvas quiz.

How many iterations are needed to fully resolve the costs for a network with ten nodes?

If two nodes have the same current cost, how do you select the node to resolve next?

What are the initial values for all nodes in the network? What are the values after the source node has been moved to the checked set?

What will be the cost of a node if it cannot be reached from the source node?

You have now finished lab 6. Remember to complete the Canvas quiz for this lab (https://canvas.auckland.ac.nz/courses/47894/quizzes/50199).