AG2411 – GIS Architecture and Algorithms
Takeshi Shirabe
Geoinformatics, KTH
2021-11-26

Exercise 4
Implementing routines to read and write network data files

In the next two exercises, you will develop a small network analysis toolbox. You will begin with the implementation of two routines: one for reading an input data file exported from ArcGIS and the other for writing an output data file that can be imported to ArcGIS. These routines serve as a bridge between your toolbox and the geographic information system.

Normally, you would need to answer this important design question: which data structure(s) should be used to represent the topological structure of a network. In this exercise, however, you are suggested to use adjacency lists.

PREPARATION

Create a working directory in which you will save all data and code related to this exercise. Download from the course website a shapefile called "tgr24031lkA_Project," which represents the road network of the City of Philadelphia, USA. Then, export only the subset of the data corresponding to "TLID," "FNODE," "TNODE," and "LENGTH" attributes as an ASCII file in the working directory. **Note that it is assumed that all roads are two-way roads in this exercise.**

The first couple of lines of the exported ASCII file should look like:

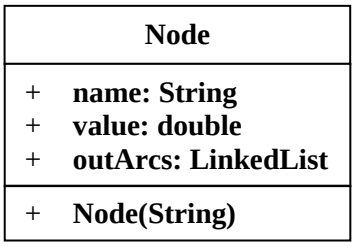
```
TLID,FNODE,TNODE,LENGTH
62043177,924,864,0.552210
62043179,864,862,0.061130
```

Alternatively, you may want to use network data of your choice. For example, if you are interested in network data of US counties, visit a US Census Bureau website "TIGER/Line® Shapefiles" (https://www.census.gov/geographies/mapping-files/time-series/geo/tiger-line-file.html). If the data do not have all the four attributes shown above, it will be your responsibility to create them (perhaps using ArcGIS).

TASKS

Note that there is no self-study guide, step-by-step instruction, or sample code for this exercise. Come to the lab and use all kinds of learning resources including your teaching staff, reference books, and past handouts to get ideas on how to approach to this assignment.

1. Create a class called **Node** from which all nodes are instantiated. It consists of at least the following attributes and method.



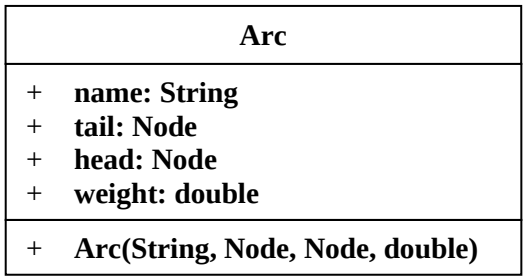
<Attributes>

- The **name** attribute is a *String* representing the name of **`this`** *Node*.
- The **value** attribute is a *double* representing the weight of **`this`** *Node*.
- The **outArcs** attribute is a *LinkedList* representing a set of all *Arcs* that emanate from **`this`** *Node*.

<Methods>

- The **Node** method (constructor) takes a *String* representing the name of a node as input and constructs a new *Node* such that its **name** is set to the input name *String*, its **value** to the positive infinity *double*, and its **outArcs** to an empty *LinkedList*.

2. Create a class called **Arc** from which all *directed* arcs are instantiated. It consists of at least the following attributes and methods.



<Attributes>

- The **name** attribute is a *String* representing the name of **`this`** *Arc*.
- The **tail** attribute is a *Node* representing the tail of **`this`** *Arc*.
- The **head** attribute is a *Node* representing the head of **`this`** *Arc*.
- The **weight** attribute is a *double* representing the weight of **`this`** *Arc*.

<Methods>

- The **Arc** method (constructor) takes a *String*, two *Nodes*, and a *double* (representing the name, tail, head, and weight of an arc, respectively) as input, constructs a new *Arc* such that its **name** is set to the input name *String*, its tail to the input tail *Node*, its head to the input head *Node*, and its **weight** to the input *double*.

3. Create a class called **Network** from which all *directed* networks are instantiated. It consists of at least the following attributes and methods.

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

<Attributes>

- The **name** attribute is a *String* representing the name of `this` *Network*.
- The **nodeMap** attribute is a *HashMap* object mapping a *String* representing a node name to the corresponding *Node* in `this` *Network*.

<Methods>

- The **Network** method (constructor) takes as input two *Strings* representing the name of a network and the name of an ASCII file storing an *undirected* network exported from ArcGIS, checks if the file exists and is correctly formatted, and constructs a new *Network* such that its **name** is set to the input network name *String* and its **nodeMap** is populated with mappings of node name *Strings* to their corresponding *Node*s.

  Note: the most critical step is
  1. to read each line of the input file,
  2. to break it into four tokens corresponding to "TLID," "FNODE," "TNODE," and "LENGTH,"
  3. to create two *Arc*s of opposite directions from them, and
  4. to put each *Arc* into the **outArcs** of its **tail** *Node*.

  Note 2: while the **name** of each *Node* may be taken from the "FNODE" or "TNODE" column of the input file, there is no obvious choice for the **name** of each *Arc*. Our suggestion would be to set the **name** of each *Arc* to its corresponding "TLID" value plus "a" or "b" depending on its direction.

- The **printNodes** method prints the **name** and **value** of each *Node* of `this` *Network* on the screen.
- The **printArcs** method prints the **name**, name of the **tail**, name of the **head**, and **weight** of each *Arc* of `this` *Network*. A lecture slide shows an example.
- The **save** method takes a *String* representing an output file name, creates a file with that name, and writes to it the "TLID" value as well as **name** and **weight** of each *Arc* in `this` *Network*. A lecture slide shows an example.

Your code may look like…

```
public class Network {

	// Attributes
	public String name;
	public HashMap<String, Node> nodeMap;

	// Constructor
	public Network(String name, String inputFileName) {

		// Initialize the attributes
		this.name = name;
		this.nodeMap = new HashMap<String, Node>();

		// You MAY need these local variables to store values or objects
		// temporarily while constructing a new Network object
		String line, arcID, tailName, headName;
		Node tail, head;
		double weight;
		Arc forwardArc, backwardArc;

		try {
			// Get access to the contents of an ASCII file
			File file = new File(inputFileName);
			FileReader fReader = new FileReader(file);
			BufferedReader bReader = new BufferedReader(fReader);

			// Read the first line, and do nothing to skip it
			line = bReader.readLine();

			// Read the second line, which represents the first
			// (undirected) arc stored in the file
			line = bReader.readLine();

			// Store each element of the network in forward star.
			while (line != null) {
				// Split each line into an array of 4 Strings
				// using , as separator.
				String[] tokens = line.split(",");
				arcID = tokens[0];
				tailName = tokens[1];
				headName = tokens[2];
				weight = Double.parseDouble(tokens[3]);

				// Do the following.
				// Check if nodeMap contains a Node whose name is
				// tailName or headName.
				// If not, create it, assign it to tail or head, and add
				// it to nodeMap.
				// Otherwise, retrieve it from nodeMap.
				// Then, create two Arcs:
				// one from tail to head, to be added to outArc of tail
				// one from head to tail, to be added to outArc of head.

				// Read the next line
				line = bReader.readLine();		}
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
```

```
	// Other methods
	public void save(String outputFileName) {

		// Do something
	}

	public void printNodes(){

		System.out.println("\tNODE NAME\tWEIGHT");
		Node node;
		for (String nodeName: nodeMap.keySet()) { // loop thru nodeMap
			node = nodeMap.get(nodeName);
			System.out.print("\t" + node.name); // \t represents tab space
			System.out.print("\t\t" + node.weight);
			System.out.println();
		}
	}

	public void printArcs(){

		// Do something
	}
}
```

4. Compile the **Node**, **Arc**, **and Network** classes.

5. Create a class called **Ex04** that consists only of the **main** method that takes two parameters indicating the name of an input file and the name of an output file, constructs a *Network* from the input file, prints attributes of its *Nodes* and *Arcs*, and saves the *Network* in the output file.

   The first couple of lines of the output file should look like:

```
TLID,NAME,WEIGHT
62043177,62043177a,890.966810
62043177,62043177b,890.966810
62043179,62043179a, 98.678440
```

6. Compile and test the **Ex04** class.

HELPS

1. Using a HashMap (revised)

Your *Network* class uses a *HashMap* to map node name *Strings* to their corresponding *Nodes*. The instruction of how to use a *HashMap* presented last week has been revised so that you can 1) test if a HashMap contains a certain key and 2) iterate through a *HashMap* and get access to the value of each element using its key.

```
// Create a HashMap
HashMap<Integer, Layer> hm = new HashMap<Integer, Layer>();

// Suppose these are IDs of layers
int i1 = 1, i2 = 2, i3 = 3;

// Create Integer objects from int values
Integer i1Obj = new Integer(i1);
Integer i2Obj = new Integer(i2);
Integer i3Obj = new Integer(i3);

// Create Layer objects
Layer l1 = new Layer("development", "./data/development.txt");
Layer l2 = new Layer("hydrology", "./data/development.txt");
Layer l3 = new Layer("vegetation", "./data/vegetation.txt");

// Map Integer objects (as keys) to Layer objects (as values)
hm.put(i1Obj, l1);
hm.put(i2Obj, l2);
hm.put(i3Obj, l3);

// This looks busy, but is an efficient style to map an Integer to a Layer.
hm.put(new Integer(4), new Layer("elevation", "./data/elevation.txt"));

// Retrieve a Layer
int i = 1;
Integer iObj = new Integer(i);
Layer l = hm.get(iObj);
System.out.println("Layer " + l.name + " has been retrieved.");

// This works, too
System.out.println("Layer " + hm.get(new Integer(2)).name + " has been
retrieved.");

// Check if the HashMap contains certain keys
for (int j=1; j<10; j++) {
      Integer jObj = new Integer(j);
      if (hm.containsKey(jObj)) { // returns true or false
            System.out.println(jObj.intValue() + " is contained by this
            HashMap");
      }
      else {
            System.out.println(jObj.intValue() + " is not contained by this
            HashMap");
      }
}
// This may be an easier way to iterate through the HashMap
for (Integer layerId: hm.keySet()) {
      l = hm.get(layerId);
      System.out.println("Layer " + l.name + " has been retrieved.");
}
```

Note: In the last block of code, the **keySet** method returns the set of keys of the hm *HashMap*, which, in the present example, are *Integer* objects. You can get access to all elements of a set sequentially using the following statement.

for (**Class of the objects stored in the set variable name: set**) {

}

2. Using a LinkedList

Your *Node* class should be designed in a way that all arcs emanating from a node are stored in that node's **outArcs**, which takes the form of a *LinkedList*. The following code illustrates how to use a *LinkedList*.

```
// Create a LinkedList (of Arcs).
LinkedList<Arc> arcList = new LinkedList<Arc>();

// Create Nodes
Node tail = new Node("864");
Node head1 = new Node("924");
Node head2 = new Node("862");
Node head3 = new Node("868");
Node head4 = new Node("931");

// Create Arcs
Arc a1 = new Arc("my_arc", tail, head1, 1);
Arc a2 = new Arc("your_arc", tail, head2, 1.5);
Arc a3 = new Arc("his_arc", tail, head3, 2.5);
Arc a4 = new Arc("her_arc", tail, head4, 2);

// Add the arcs to the LinkedList
arcList.add(a1);
arcList.add(a2);
arcList.add(a3);
arcList.add(a4);

// Get access to each element of the LinkedList
System.out.println(ll.get(0).name + ": " + arcList.get(0).weight);
System.out.println(ll.get(1).name + ": " + arcList.get(1).weight);
System.out.println(ll.get(2).name + ": " + arcList.get(2).weight);
System.out.println(ll.get(3).name + ": " + arcList.get(3).weight);

// Do the above more efficiently by iterating through the LinkedList
for (Arc a: arcList) {
      System.out.println(a.name + ": " + a.weight);
}
```

Note: As seen in the last block of code, you can get access to all elements of a *LinkedList* sequentially using the following statement:

for (**Class of the objects stored in the linkedList variable name: linkedList**) {

}

SUBMISSION

Submit the following.

- Source codes of **Node**, **Arc**, **Network**, **and Ex04** classes.

DUE DATE

To avoid penalty points, your submission must be made through Canvas no later than 11:59 PM, December 2, 2021.