Data Structures and Algorithms 120

Assignment Semester 1, 2016

Student ID: 18249833

Name: Xhien Yi Tan (Xavier)

This submission includes the following required classes:

- { FloatingPoint, Vertex, Edge, Network }.java
- { Qc, Qn }.java
- { HNode, OBDDH }.java

In addition to that, the following supporting classes are also included:

- Component.java
- FileIO.java
- { DSAQueue, DSALinkedList }.java (Including DSAListNode and DSAListIterator)
- Main.java

The following Testing classes are also included:

- UnitTest { FloatingPoint, Component, Vertex, Edge, Network }.java
- UnitTest{ Qn, Qc }.java
- UnitTest{ DSAQueueGenerics, DSALinkedListGenerics }.java

Also included the non-class files:

- DSADocumentation.pdf
- Cover Sheet.pdf

Overview

My program will all starts from reading in a file and determine whether the file is sorted or unsorted and then store the contents into appropriate ADT's for sorting if it is unsorted. After that the program will start writing the sorted contents into a file that has the same file name but with different suffix - .srt. User can load the sorted data whenever he needed from the file that has the suffix of .srt. After reading from a file and writing to a file, it will start the computation of the sorted Network.

Class Description (Other than those covered in the class specification section for class description)

- Component Class
  - Purpose: A parent of class of both Vertex and Edge classes
  - Reason: Both Vertex and Edge has the same class fields such as name, reliability and type. So in this case, I extract some of the same fields into a class, which is Component Class. With the use of inheritance, the information or details is made manageable and the code from the parent class can be reused. Without the use of inheritance, there will be many duplicate methods like getRel(), getName() and also getType().
- FileIO Class
  - Purpose: It is used to read from a file and write to a file. Its main purpose is to read from a file and store the contents of the file into a suitable data structure and can be used later to compute the reliability. After reading in an unsorted file which the file usually has a suffix of .nt, the Network class will sort the contents and FileIO is responsible for writing sorted contents into the same file name but with different suffix which is .srt. The reason for that is that if the user wants to use the sorted data, the user can load from the sorted file instead of reading in the unsorted file and do the sorting again.
  - Reason: Without the use of this class, nothing can read the contents of the file. I have created this class to read from a file and also write to a file to make user has the ability to load from a sorted file

Justification of decision made (Questions that the marker may have)

- Network Class
  - **Edge (QUEUE)**: There are many reason for me to use queue for Edge in the Network class. First of all, nextEdge() method. This method returns the next Edge in the Network. By looking at this, I have decided to use queue because every time when we deal with Edge, we deal with 1 Edge at a time, which is the property of queue. I would not use Stack or LinkedList or even Array for Edge because it is unnecessary to have LIFO and also access the last Edge in the list and also array cant shrink or grow. We are only dealing with the first edge, one at a time.
  - **Vertex (QUEUE, LINKEDLIST)**: I have chosen LinkedList for storing Vertex in the Network class and Queue for sorting Vertex in the Network class. In my **sorting Vertex** part, I use the idea of BFS. As you can see that I have a queue for storing

Vertex. I need to enqueue and dequeue every time I need to find other vertex that has a relationship with. With the idea of BFS, I need to use a queue for that part. After sorting all my Vertices using queue, I will insert sorted vertices into a linked list, this is because with a list user can access the SOURCE and TARGET when needed. I would not use Stack or Queue or Array because it is unnecessary to have LIFO for Vertex and also dealing only with the first Vertex and also an array cant grow or shrink.

- **Network(String filename):** I may need to explain on this method more. This method imports a filename and returns a Network. I will pass this filename into the reading() method in FileIO class. It returns a network, and it will be set to the class. After that, I will check that the file suffix is whether .srt (which stands for sorted file) or .nt (which stands for unsorted file). If the suffix is .nt, I will sort the Network and write the sorted Network into a file where the file name is the same as the imported one except that the suffix of the file will be changed to .srt. If the user wants to use the sorted data in the program later, it will load the data from the file name that has a suffix of .srt.

- **Sorting Vertex:** As I have mentioned from above, I use the idea of BFS to sort my Vertex using queue. I have two methods for sorting vertices – calcSource() and sortVertexRecurse(). First of all in the calcSource() method, I will loop through the Edges to look for a SOURCE Vertex and enqueue that SOURCE Vertex into the queue and also set the SOURCE Vertex to VISITED. After enqueue, I will again loop through the Edges to find whichever Edge has a SOURCE Vertex and I will change the states of the SOURCE. This method ends and return the queue that has SOURCE in it and pass it into sortVertexRecurse(). In this method, I recursively dequeue to get the first Vertex and insert that to a list (where sorted Vertices will be stored) and use that Vertex to look for another Vertex that is connected from the first Vertex. After looking up the Vertex that is connected from the first Vertex, I will again change the states to VISITED and enqueue into a queue and do the same thing again. For sorting vertices, I have a method called iterateEdge(), this method just iterate(loop) the Edges and change the states of the Vertices that have been enqueued into the queue.

- **Sorting Edge:** For sorting Edges, I have a method called sortEdge(). This method imports the sorted list of Vertices which is sorted in calcSource() and sortVertexRecurse(). By using the sorted list, I use the Vertices' names to loop through the Edge queue to get the Edge that has Vertex A and Vertex B. For example, using Vertex A and Vertex A to find that any Edge that has those two vertices. After that, use Vertex A and Vertex B, so on and so on. Once the Edge is found, enqueue it into a queue. Finally, set the sorted Edge queue to this class.

- FileIO Class

  - **Reading:** In the reading part of my FileIO class, I will first read line by line, and store line that has 'V' as the second letter to a Queue, and also store line that has 'E' as the second letter to a Queue as well, this is to prevent unsorted file. These queues

that store lines are later used to process each Vertex and Edge. The reason I choose to use Queue over array in this case is that before using array, it must be initialized. Due to my implementation, I would not know the numbers of line needed for Vertex at that point, which means I can't initialize an array to store lines without a proper size.

- **Process Vertex or Edge:** In order to process each of these, I dequeue the line that I stored in a queue. Using that particular line, I break each line into tokens with the use of String Tokenizer. After that, I can determine the fields by using the tokens. For example, if the first letter of the token is R, which mean the field must be reliability.

- **Qc and Qn Class:**

  - **Queue:** For both Qc and Qn class, Queue is used. In these both classes, we are dealing with node one at a time. The reason that I choose Queue is because the property of Queue fits in to this. Both Qc and Qn have similar methods. In addition to that, Qn has methods called add() and swap(). In the add() method, I import a node and loop through the next level queue to find another node that has the same state as the imported node. Once found, true is return. In the swap() method, I will dequeuer every single node in the next level queue to the current level queue.