Assignment Four – LaTeX Sorts

Daniel Bilotto
danielbilotto1@marist.edu

December 11, 2021

1 Compute Class

He is the main class, and I should probably mention it here that it does not work. But nonethe less I will go through it. Unlike last project I go the read a file to work (yay!) but this time I can't seem to get the algorithm right. See I have read the file in and everything works right up until I try to add the edge to the list. but beyond that everything seems to be working. I have a linkedlist/graph that contains my nodes/vertices and each node has a array list of edges. After this though we have the spice stuff which is also not working. But the basic idea is that I'm splitting on spaces (could split on ; and then space) and then populate a array list for the spices. And then execute the greedy algorithm. I spelt too much time (basically all of it) on the SSSP algorithm that I really didn't have much to work out the spices which as I mention later in the document sucks because I am honestly interested in greedy algorithms. Point being the algorithm would take the scoops fractionally down the sorted list of unit prices until the capacity is met.

Some things to note is that while I'm only reading in one graph, the code is use able to add more graphs in I just did it that way to test the code when working.

```
1  import java.io.File;
2  import java.util.Scanner;
3  import java.util.ArrayList;
4  //import java.util.ArrayList;
5  import java.util.Arrays;
6  import java.util.Collections;
7
8  public class Compute {
9    // Create a new keyboard Scanner object.
10     static Scanner keyboard = new Scanner(System.in);
```

```
11
12
13  public static void main(String[] args)
14  {
15    String fileName = null;
16      String[] line = null;
17      String[] line2 = null;
18      int num;
19      int source;
20      int dest;
21      int weight;
22      QueueBilotto graph = new QueueBilotto();
23      int verCount = 0;
24      int edgeCount = 0;
25      NodeBilotto edgeNode = new NodeBilotto();
26      SSP path = new SSP();
27
28
29      try
30      {
31        fileName = "test.txt";
32
33        File theFile = new File(fileName);
34
35        Scanner input = new Scanner(theFile);
36
37        //this will read the file together with the code below
38
39        while(input.hasNext())
40        {
41          line = input.nextLine().replaceAll("  ", " ").split(" ");
42          //System.out.println(Arrays.toString(line));
43
44          if (line[0].equalsIgnoreCase("new"))
45          {
46            //System.out.println(Arrays.toString(line));
47            graph = new QueueBilotto();
48            verCount = 0;
49            edgeCount = 0;
50            num = 0;
51
52          }
53
54          else if (line[0].equalsIgnoreCase("add"))
55          {
56            if (line[1].substring(0,1).equalsIgnoreCase("v"))
57            {
58              //System.out.println(Arrays.toString(line));
59              NodeBilotto vert = new NodeBilotto();
60              num = Integer.parseInt(line[2]);
61              vert.ver = num;
62              graph.enqueue(vert);
63              verCount++;
64            }
65            else
66            {
67              //System.out.println(Arrays.toString(line));
```

```
68
69              source = Integer.parseInt(line[2]);
70              dest = Integer.parseInt(line[4]);
71              weight = Integer.parseInt(line[5]);
72
73
74              NodeBilotto srcNode = graph.search(source);
75              NodeBilotto destNode = graph.search(dest);
76
77
78              Edge edge = new Edge(destNode, weight);
79
80              //srcNode.edge.add(edge);
81              edgeCount++;
82
83            }
84          }
85
86
87
88        }//while
89
90
91
92        input.close();
93        keyboard.close();
94      }//try
95
96      catch(Exception ex)
97      {
98        System.out.println("Oops, something went wrong!");
99      }//catch
100
101     //this sends back a message if something goes wrong in
    importing the text into the array from magic items
102
103
104     /*
105     try
106     {
107       fileName = "spice.txt";
108
109       File theFile = new File(fileName);
110
111       Scanner input = new Scanner(theFile);
112
113       //this will read the file together with the code below
114
115       while(input.hasNext())
116       {
117         line = input.nextLine().replaceAll("\\s+", " ").split("
    ");
118         System.out.println(Arrays.toString(line));
119         int price;
120         int qty;
121
122         if (line[0].equalsIgnoreCase("spice"))
```

```
123           {
124             ArrayList<Spice> spices = new ArrayList<Spice>();
125             Spice spice = new Spice();
126             spice.name = line[3];
127             price = Integer.parseInt(line[6]);
128             qty = Integer.parseInt(line[9]);
129             spice.price = price;
130             spice.qty = qty;
131             spice.unitPrice.add(price/qty);
132             spices.add(spice);
133             selectSort(spice.unitPrice);
134 
135           }
136 
137           else if (line[0].equalsIgnoreCase("knapsack"))
138           {
139            int scoops = 0;
140            int capacity = Integer.parseInt(line[3]);
141 
142            if (capacity > 0)
143            {
144 
145            }
146 
147           }
148 
149 
150         }//while
151 
152 
153 
154         input.close();
155         keyboard.close();
156       }//try
157 
158       catch(Exception ex)
159       {
160         System.out.println("Oops, something went wrong!");
161       }//catch
162 
163 
164 
165       */
166 
167       //Prints out the graph!! (well at least the vertices!)
168 
169 
170       /*
171       for (int i = 0; i < edgeNode.edge.size(); i++)
172       {
173         System.out.println(edgeNode.edge.get(i).getSource().getData
      ());
174         System.out.println(edgeNode.edge.get(i).getDest().getData()
      );
175         System.out.println(edgeNode.edge.get(i).getWeight());
176       }
177       //Prints out the edges
```

```
178       */
179
180       /*
181       path.bellman(graph, edgeNode, verCount, edgeCount);
182       path.print(graph, graph.head, verCount);
183       System.out.println();
184       */
185
186
187     }//main
188
189
190   public static void selectSort(ArrayList<Integer> items)
191     {
192       for (int i = 0; i < items.size(); i++)
193       {
194         int smallPos = i;
195         for (int j = i; j < items.size(); j++)
196         {
197           if (items.get(j) < items.get(smallPos))
198           {
199             smallPos = j;
200           }
201         }
202         int swap = items.get(smallPos);
203         items.set(smallPos, items.get(i));
204         items.set(i, swap);
205       }
206
207     }//SS
208
209 }//Compute
```

2 Proof of multiple graphs!

He is proof that it can read in multiple graphs.

```
1  import java.io.File;
2  import java.util.Scanner;
3  import java.util.ArrayList;
4  //import java.util.ArrayList;
5  import java.util.Arrays;
6  import java.util.Collections;
7
8  public class Compute {
9    // Create a new keyboard Scanner object.
10     static Scanner keyboard = new Scanner(System.in);
11
12
13   public static void main(String[] args)
14   {
15     String fileName = null;
16       String[] line = null;
17       String[] line2 = null;
18       int num;
19       int source;
```

```
20        int dest;
21        int weight;
22        QueueBilotto graph = new QueueBilotto();
23        int verCount = 0;
24        int edgeCount = 0;
25        NodeBilotto edgeNode = new NodeBilotto();
26        SSP path = new SSP();
27
28
29        try
30        {
31          fileName = "graphs.txt";
32
33          File theFile = new File(fileName);
34
35          Scanner input = new Scanner(theFile);
36
37          //this will read the file together with the code below
38
39          while(input.hasNext())
40          {
41            line = input.nextLine().replaceAll("  ", " ").split(" ");
42            //System.out.println(Arrays.toString(line));
43
44            if (line[0].equalsIgnoreCase("new"))
45            {
46              System.out.println(Arrays.toString(line));
47              graph = new QueueBilotto();
48              verCount = 0;
49              edgeCount = 0;
50              num = 0;
51
52            }
53
54            else if (line[0].equalsIgnoreCase("add"))
55            {
56              if (line[1].substring(0,1).equalsIgnoreCase("v"))
57              {
58                System.out.println(Arrays.toString(line));
59                NodeBilotto vert = new NodeBilotto();
60                num = Integer.parseInt(line[2]);
61                vert.ver = num;
62                graph.enqueue(vert);
63                verCount++;
64              }
65              else
66              {
67                System.out.println(Arrays.toString(line));
68
69                source = Integer.parseInt(line[2]);
70                dest = Integer.parseInt(line[4]);
71                weight = Integer.parseInt(line[5]);
72
73
74                //NodeBilotto srcNode = graph.search(source);
75                //NodeBilotto destNode = graph.search(dest);
76
```

```
77
78                //Edge edge = new Edge(destNode, weight);
79
80                //srcNode.edge.add(edge);
81                edgeCount++;
82
83              }
84            }
85
86
87
88          }//while
89
90
91
92          input.close();
93          keyboard.close();
94        }//try
95
96        catch(Exception ex)
97        {
98          System.out.println("Oops, something went wrong!");
99        }//catch
100
101       //this sends back a message if something goes wrong in
      importing the text into the array from magic items
102
103     }//main
104
105
106 }//Compute
```

3 SSP class

This is the Single Source Shortest Path class and it should be correct but I haven't actually successfully gotten it to run yet. What this class does is calculates the path dynamically. So it first off takes the source and sets it to 0 and makes everything else infinity (or in this case the max integer because I don't think java has infinity) then by passing it the directed graph along with the source node and the number of vertexes. it will loop that many times minus one and find the shortest path, speaking of which that is the job of the relax method. To calculate the distance and the bellman method just does the work and drives it.

```
1 import java.util.ArrayList;
2
3 public class SSP {
4   public void initSS(QueueBilotto graph, NodeBilotto source, int
      verCount)
5   {
6     for (int i = 1; i < verCount; i++)
7     {
8       NodeBilotto vertex = graph.search(i); //come back to this
```

```
9       vertex.distance = Integer.MAX_VALUE;
10      vertex.prevVertex = null;
11      source.distance = 0;
12    }
13  }//init
14
15  public void bellman(QueueBilotto graph, NodeBilotto sour, int
      verCount, int edgeCount)
16  {
17    this.initSS(graph, sour, verCount);
18
19    NodeBilotto source = sour;
20    NodeBilotto destination = sour;
21
22
23    for (int i = 0; i < (verCount - 1); i++)
24    {
25      for (int k = 1; k < verCount; k++)
26      {
27        source = graph.search(k);//come back to this
28        System.out.println(source.getData());
29
30        for (int j = 0; j < source.edge.size(); j++)
31        {
32          destination = source.edge.get(j).getDest();
33          int weight = source.edge.get(j).getWeight();
34
35          this.relax(graph, source, destination, weight);
36        }
37      }
38    }
39
40
41  }
42
43  public boolean relax(QueueBilotto graph, NodeBilotto start,
      NodeBilotto end, int weight)
44  {
45
46    if (end.distance > start.distance + weight)
47    {
48      end.distance = start.distance + weight;
49      end.prevVertex = start;
50      return false;
51
52    }
53
54    return true;
55  }
56
57
58  public void print (QueueBilotto graph, NodeBilotto sourceVert,
      int verCount)
59  {
60    NodeBilotto dest = null;
61    int cost = 0;
62
```

```
63      for (int g = (sourceVert.ver + 1); g < verCount; g++)
64      {
65        dest = graph.search(g);
66        cost = dest.distance;
67        System.out.print(sourceVert.getData() + " -> " + (g) + " cost
          is " + cost + "; path is " + sourceVert.getData());
68
69        NodeBilotto tempDest = dest;
70        ArrayList<NodeBilotto> tempArray = new ArrayList<>();
71        while(tempDest.prevVertex != null) {
72          tempArray.add(tempDest);
73          tempDest = tempDest.prevVertex;
74        }//end while
75
76        for(int p = tempArray.size()-1; p > -1; p--) {
77          System.out.print(" -> " + tempArray.get(p).getData());
78        }//end for p
79        System.out.print(".");
80        System.out.println();
81      }
82    }
83  }
```

4 Edge class

This is the Edge Class. Pretty straight forward if I do say so myself. Basically an edge has 3 things. A node that tells it the source, another node that tells it the destination, and a int that tells it the weight.

```
1
2   public class Edge {
3
4     NodeBilotto mySource;
5     NodeBilotto myDest;
6     int myWeight;
7
8
9     public Edge(NodeBilotto dest, int weight)
10    {
11      myDest = dest;
12      myWeight = weight;
13    }
14
15
16
17    public void setSource(NodeBilotto newSource)
18    {
19      mySource = newSource;
20    }
21
22    public void setDest(NodeBilotto newDest)
23    {
24      myDest = newDest;
25    }
26
```

```
27   public void setweight(int newWeight)
28   {
29     myWeight = newWeight;
30   }
31
32
33   public NodeBilotto getSource()
34   {
35     return mySource;
36   }
37
38   public NodeBilotto getDest()
39   {
40     return myDest;
41   }
42
43   public int getWeight()
44   {
45     return myWeight;
46   }
47
48
49 }//edge
```

5 Node class

This is the Node class and quite a bit has changed. For starters it has some new variables like a array list of edges, a previous vertex, distance and an id called ver (line 12). Basically this is the same Node class we all know and love with a few more bells and whistles.

```
1
2  import java.util.ArrayList;
3
4  public class NodeBilotto
5  {
6    public NodeBilotto myData;
7    public NodeBilotto myNext;
8    public NodeBilotto prev;
9    int distance;
10   NodeBilotto prevVertex;
11   ArrayList<Edge> edge = new ArrayList<Edge>();
12   public int ver;
13
14   public NodeBilotto ()
15   {
16     myData = null;
17     myNext = null;
18     prev = null;
19   }//NodeBilotto
20
21   public NodeBilotto (NodeBilotto newData)
22   {
23     myData = newData;
24     myNext = null;
```

```
25    }//NodeBilotto
26
27    public NodeBilotto getData()
28    {
29      return myData;
30    }//getData
31
32    public void setData (NodeBilotto newData)
33    {
34      myData = newData;
35    }//setData
36
37    public NodeBilotto getNext()
38    {
39      return myNext;
40    }//getNext
41
42    public void setNext(NodeBilotto newNext)
43    {
44      myNext = newNext;
45    }//setNext
46
47
48
49
50
51  }//node
```

6 Queue class

This is the Queue Class or AKA the linked list class. Originally I had a class called "Graph" but i scrapped it because I have this and it's better. So things that were added are things like the serach method. I use this to pair the vertices read in by the file to the nodes in this list. I also changed some parameters to accept nodes and such.

```
1
2
3   public class QueueBilotto {
4
5     public NodeBilotto head;
6
7     private NodeBilotto tail;
8
9     public QueueBilotto()
10    {
11      head = null;
12      tail = null;
13    }//Queue
14
15    public boolean isEmpty()
16    {
17      return (head == null);
18    }//is empty
19
```

```
20
21    public void enqueue(NodeBilotto item)
22    {
23      NodeBilotto oldTail = tail;
24      tail = new NodeBilotto();
25      tail.setData(item);
26      tail.setNext(null);
27      if (isEmpty())
28      {
29        head = tail;
30      }//if
31      else
32      {
33        oldTail.setNext(tail);
34      }//else
35
36    }//enqueue
37
38
39    public NodeBilotto dequeue()
40    {
41      NodeBilotto item = null;
42      if (!isEmpty())
43      {
44        item = head.getData();
45        head = head.getNext();
46      }//if
47
48      return item;
49    }//dequeue
50
51
52    public NodeBilotto search(int num)
53    {
54      NodeBilotto curr = this.head;
55      while (curr != null && curr.ver != num)
56      {
57        curr = curr.getNext();
58      }
59
60      return curr;
61    }
62
63  }//queue
```

7 Spice class

This is the Spice Class. The Spices have a name, a price, a quantity, and a unit price. As of now I'm not sure if making the unit price a array list was a good idea but I did that so that I could use my selection sort to sort the spices by the unit price (see lines 240 in compute class). Overall I spent the majority of my time on SSSP and didn't get to think this one out as much which honestly sucks because greedy algorithms seem really interesting.

```
1
2  import java.util.ArrayList;
3
4  public class Spice {
5    String name;
6    int price;
7    int qty;
8    ArrayList<Integer> unitPrice = new ArrayList<Integer>();
9
10 }
```

8 Overall Thoughts

The complexity of SSSP is big O of (number of vertices and number of edges) or O(VE) Because of the initialization of O(V) and then loops O(E) times. And the greedy algorithm has a complexity of O(nlogn) if you use quick sort. The loop itself should take O(N) so then it's really the compleity of the sort you use. So since I was using selection sort it should be O(n to the 2).

Overall this project in my opinion was better than the last one. But I wasn't able to get everything to work. I feel like I'm very, very close to SSSP working. I personally think I'm one or two bugs off from it working. I will personally work on fixing it (both SSSP and the spices) and if allowed will re-push my solution to the github. If you are reading this on the 11th, there is probably a 90 percent chance I am debugging as you read this. Regardless I thought it would be best to submit what I have. But will repush if allowed to by you.