Randy Truong
CS171

Project Report

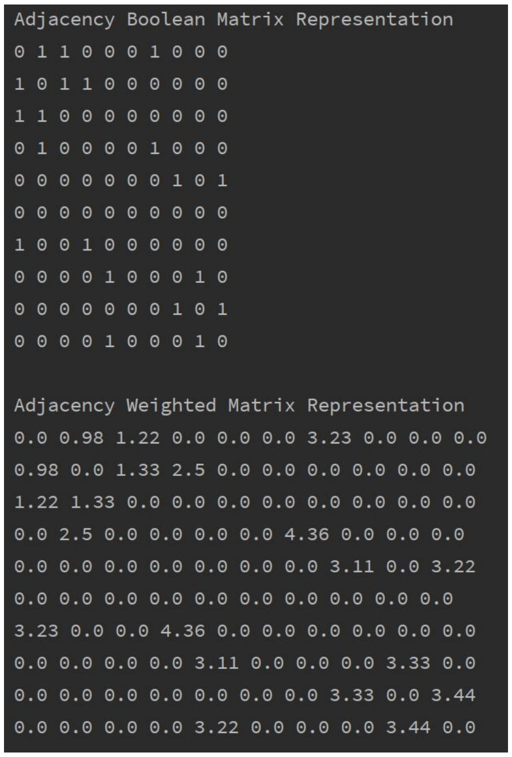
The attached file is called “WeightedUndirectedGraph.java.” This program constructs a weighted, undirected graph from a provided .txt file that the user either provides or implements. In this graph, we use the number of vertices, the number of edges, different pairs of vertices with associated weighted edges, so in the .txt file, the format of the data should be as follows:
<(int) denoting # of vertices>, <(int) denoting # of edges>
<(int) some vertex>, <(int) other vertex>, <(int/double edge weight>
<(int) some vertex>, <(int) other vertex>, <(int/double edge weight>
… (pair 1) ...
… (pair 2) ...
… (pair 3) ...
<(int) some vertex>, <(int) other vertex>, <(int/double edge weight>
(**) Note the commas that separate each value!
The program is designed with the following classes, constructors, and methods:

<u>WeightedUndirectedGraph</u>	<u>FileReader</u>
-V: int -E: int -adjMatrix: boolean[][] -adjWeightedMatrix: double[][] -adjListArray[]: LinkedList<Integer> -file: FileReader ---------------------- +WeightedUndirectedGraph(file: FileReader) +addEdge(): void +dfs(v: int, visited: boolean[]): void +checkConnected(): void +minEdge(key[]: double, mstSet[]: Boolean): int +primMST(graph[][]: double): void +primMST(): void +printMST(parent: int[], graph: double[][]): void +printAdjMatrix(): void +printAdjMatrixWeighted(): void	-scanner: Scanner ---------------------- +FileReader(name: String) +readInt(): int +readDouble(): double

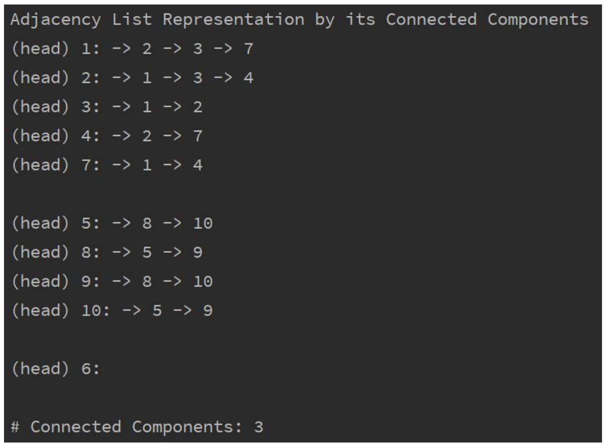
The program constructs a weighted, undirected graph from a text file, then finds adjacency list representations, adjacency matrix representations, connected components, dfs traversal, Prim’s traversal, MST, and various helper methods.
I used arrays, adjacency lists, which have space O(|V|+|E|), and matrices, which have space $O(V^2)$. My output for graph1.txt, graph2.txt, and graph3.txt are below. Although I know how to do Prim’s algorithm for MST on paper, I struggled to effectively implement it using code, but

my idea was to use a boolean array that marks the unvisited vertices, then starting Prim's traversal from that vertex. Further, my algorithm/program has an inefficient time complexity. When constructing the graph, I implement many for-loops, especially double for-loops for matrices, which have a time complexity of O(V^2), thus the program takes longer to run because I initialize matrices from at each i,j index, then I add values to each i.j index, then modify accordingly. Navigating a matrix is time- and space- consuming because the program must check every position; however, adding and removing vertices or edges is O(1) because it simply reads the values from the file and puts it in the matrix at [i][j] = [vertex][otherVertex] position. I implemented depth-first search (dfs), which has a time complexity of O(V + E).

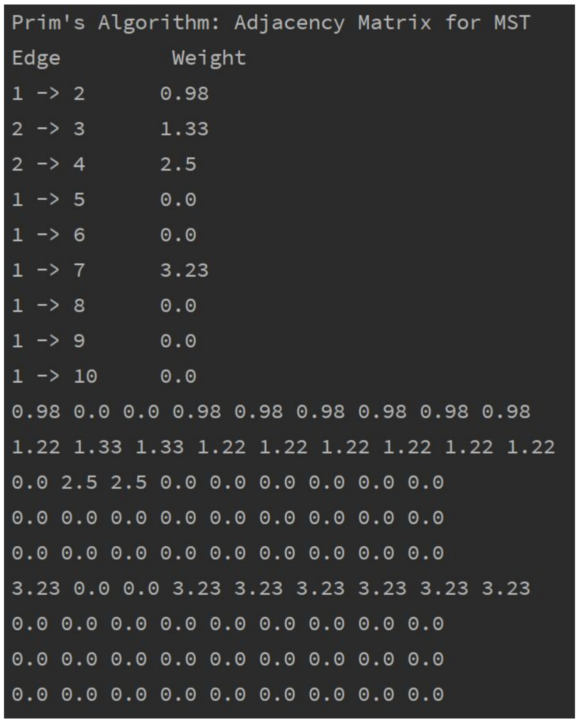
Output: graph1.txt



```
Adjacency Boolean Matrix Representation
0 1 1 0 0 0 1 0 0 0
1 0 1 1 0 0 0 0 0 0
1 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0 0 0
1 0 0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 1 0
0 0 0 0 0 0 0 1 0 1
0 0 0 0 1 0 0 0 1 0

Adjacency Weighted Matrix Representation
0.0 0.98 1.22 0.0 0.0 0.0 3.23 0.0 0.0 0.0
0.98 0.0 1.33 2.5 0.0 0.0 0.0 0.0 0.0 0.0
1.22 1.33 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 2.5 0.0 0.0 0.0 0.0 4.36 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 3.11 0.0 3.22
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
3.23 0.0 0.0 4.36 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 3.11 0.0 0.0 0.0 3.33 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 3.33 0.0 3.44
0.0 0.0 0.0 0.0 3.22 0.0 0.0 0.0 3.44 0.0
```

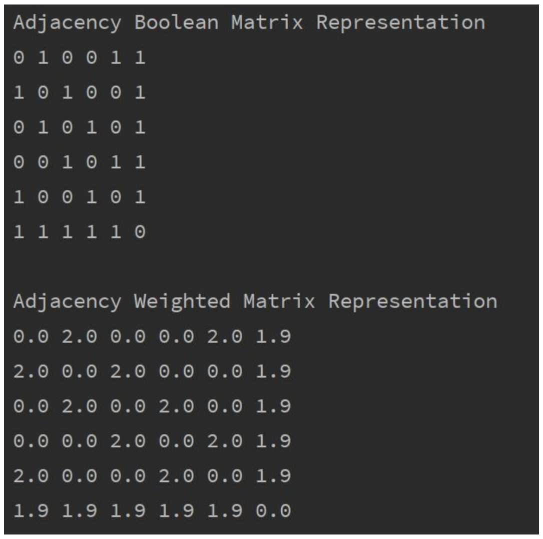


```
Adjacency List Representation by its Connected Components
(head) 1: -> 2 -> 3 -> 7
(head) 2: -> 1 -> 3 -> 4
(head) 3: -> 1 -> 2
(head) 4: -> 2 -> 7
(head) 7: -> 1 -> 4

(head) 5: -> 8 -> 10
(head) 8: -> 5 -> 9
(head) 9: -> 8 -> 10
(head) 10: -> 5 -> 9

(head) 6:

# Connected Components: 3
```

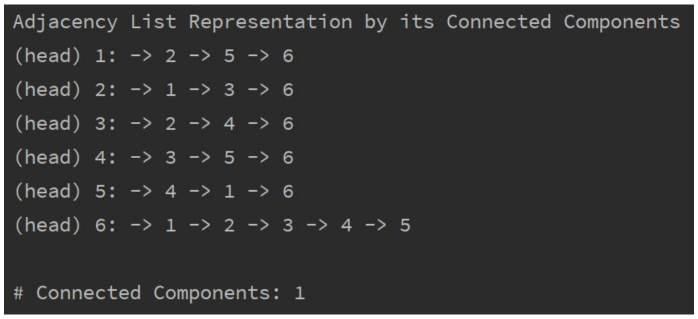


```
Prim's Algorithm: Adjacency Matrix for MST
Edge         Weight
1 -> 2      0.98
2 -> 3      1.33
2 -> 4      2.5
1 -> 5      0.0
1 -> 6      0.0
1 -> 7      3.23
1 -> 8      0.0
1 -> 9      0.0
1 -> 10     0.0
0.98 0.0 0.0 0.98 0.98 0.98 0.98 0.98 0.98
1.22 1.33 1.33 1.22 1.22 1.22 1.22 1.22 1.22
0.0 2.5 2.5 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
3.23 0.0 0.0 3.23 3.23 3.23 3.23 3.23 3.23
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

Output: graph2.txt

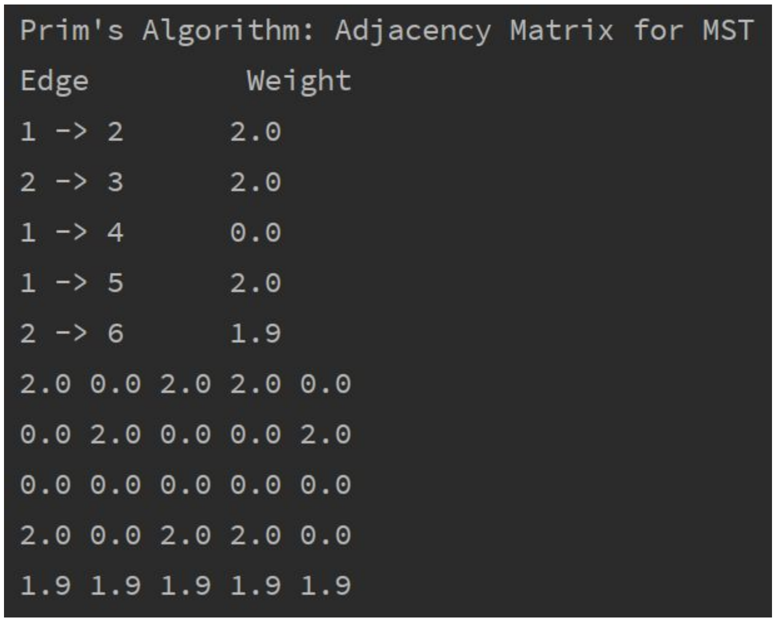


```
Adjacency Boolean Matrix Representation
0 1 0 0 1 1
1 0 1 0 0 1
0 1 0 1 0 1
0 0 1 0 1 1
1 0 0 1 0 1
1 1 1 1 1 0

Adjacency Weighted Matrix Representation
0.0 2.0 0.0 0.0 2.0 1.9
2.0 0.0 2.0 0.0 0.0 1.9
0.0 2.0 0.0 2.0 0.0 1.9
0.0 0.0 2.0 0.0 2.0 1.9
2.0 0.0 0.0 2.0 0.0 1.9
1.9 1.9 1.9 1.9 1.9 0.0
```

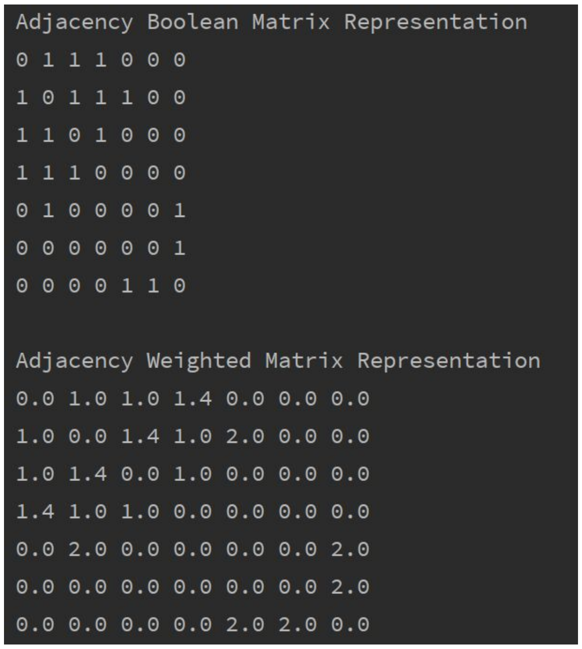


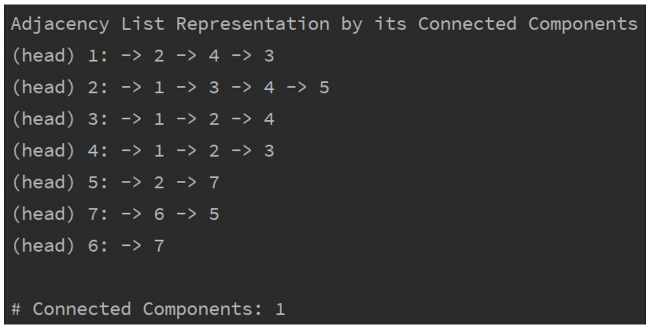
```
Adjacency List Representation by its Connected Components
(head) 1: -> 2 -> 5 -> 6
(head) 2: -> 1 -> 3 -> 6
(head) 3: -> 2 -> 4 -> 6
(head) 4: -> 3 -> 5 -> 6
(head) 5: -> 4 -> 1 -> 6
(head) 6: -> 1 -> 2 -> 3 -> 4 -> 5

# Connected Components: 1
```

Randy Truong
CS171

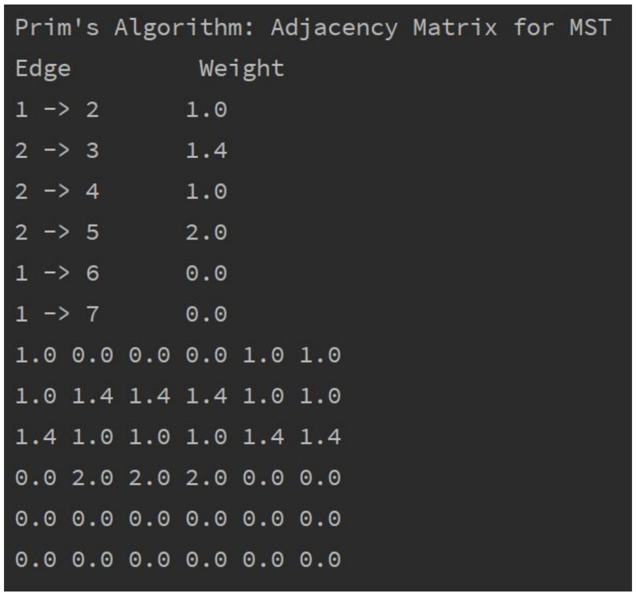

```
Prim's Algorithm: Adjacency Matrix for MST
Edge          Weight
1 -> 2       2.0
2 -> 3       2.0
1 -> 4       0.0
1 -> 5       2.0
2 -> 6       1.9
2.0 0.0 2.0 2.0 0.0
0.0 2.0 0.0 0.0 2.0
0.0 0.0 0.0 0.0 0.0
2.0 0.0 2.0 2.0 0.0
1.9 1.9 1.9 1.9 1.9
```

Output: graph3.txt


```
Adjacency Boolean Matrix Representation
0 1 1 1 0 0 0
1 0 1 1 1 0 0
1 1 0 1 0 0 0
1 1 1 0 0 0 0
0 1 0 0 0 0 1
0 0 0 0 0 0 1
0 0 0 0 1 1 0

Adjacency Weighted Matrix Representation
0.0 1.0 1.0 1.4 0.0 0.0 0.0
1.0 0.0 1.4 1.0 2.0 0.0 0.0
1.0 1.4 0.0 1.0 0.0 0.0 0.0
1.4 1.0 1.0 0.0 0.0 0.0 0.0
0.0 2.0 0.0 0.0 0.0 0.0 2.0
0.0 0.0 0.0 0.0 0.0 0.0 2.0
0.0 0.0 0.0 0.0 2.0 2.0 0.0
```


```
Adjacency List Representation by its Connected Components
(head) 1: -> 2 -> 4 -> 3
(head) 2: -> 1 -> 3 -> 4 -> 5
(head) 3: -> 1 -> 2 -> 4
(head) 4: -> 1 -> 2 -> 3
(head) 5: -> 2 -> 7
(head) 7: -> 6 -> 5
(head) 6: -> 7

# Connected Components: 1
```

Randy Truong
CS171



```
Prim's Algorithm: Adjacency Matrix for MST
Edge         Weight
1 -> 2      1.0
2 -> 3      1.4
2 -> 4      1.0
2 -> 5      2.0
1 -> 6      0.0
1 -> 7      0.0
1.0 0.0 0.0 0.0 1.0 1.0
1.0 1.4 1.4 1.4 1.0 1.0
1.4 1.0 1.0 1.0 1.4 1.4
0.0 2.0 2.0 2.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0
```