Lab 07

Graphs

Q1

- Does not require any coding.
- Submit a PDF file with your answers.
- Name your file **Lab7Question1.pdf**.
- Two parts:
    1. Part A - Prim's algorithm
    2. Part B - Kruskal’s Algorithm

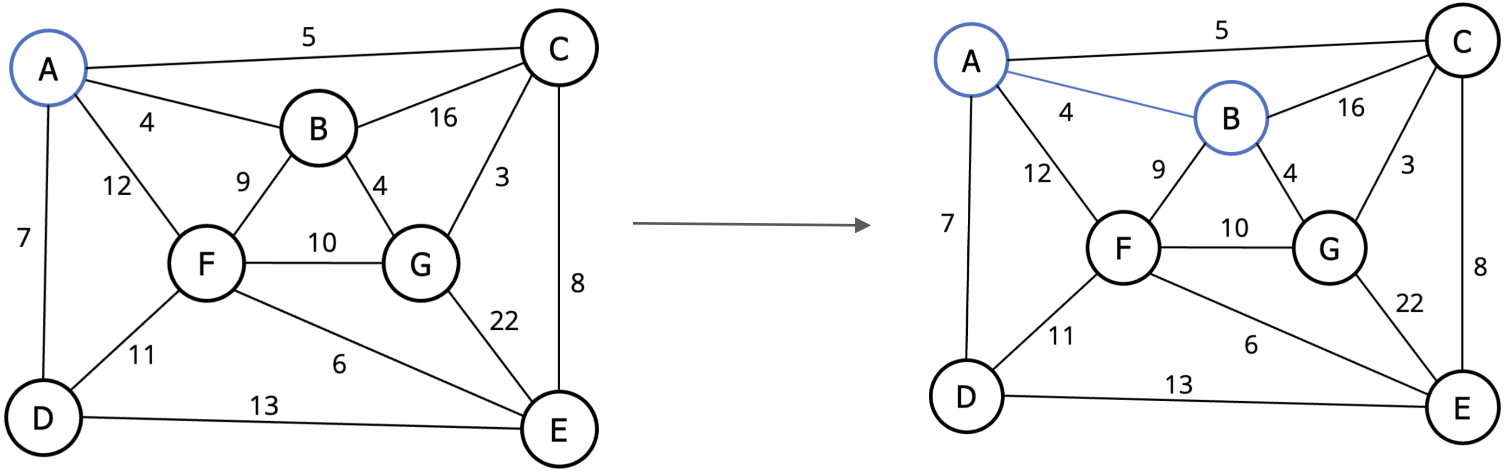
Q1 – Part A (Prim’s Algorithm)

Apply Prim's algorithm (covered in lecture 14) on the provided graph to construct a minimum spanning tree starting from vertex A.

Check lecture14_Graph_Part 3 - Page 8-16

- Greedy approach to find the minimum spanning tree.
- Hint:
- Grow a tree by adding an edge from the “known” vertices to the “unknown” vertices.
- Pick the edge with the smallest weight.

Q1 – Part A (Prim’s Algorithm)



Vertex = **{A,}**

Edge list = **{}**

Vertex = **{A,B}**

Edge list = **{{A,B}}**

Total Cost: ??

Q1 – Part B (Kruskal’s Algorithm)

Step through Kruskal’s algorithm (covered in lecture 14) to calculate a minimum spanning tree of the provided graph.

Check lecture14 - Page 20-32

Hint:

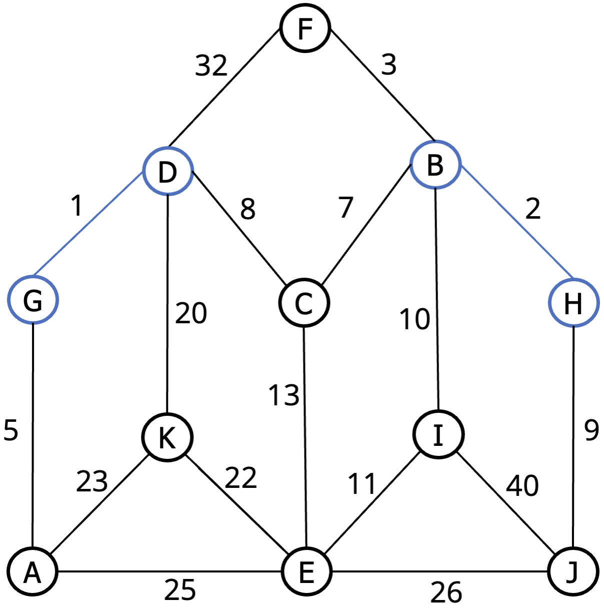
▪ Grow a forest out of edges that do not create a cycle.

▪ Pick an edge with the smallest weight

▪ Steps - Remove all loops and parallel edges (keep the minimum one) - Arrange all edges in their increasing order of weight - Add the edge which has the least weight

Q1 – Part B (Kruskal’s Algorithm)

Step through Kruskal’s algorithm (covered in lecture 14) to calculate a minimum spanning tree of a graph.



~~1~~	~~{D, G}~~
2	{B, H}

[redacted]

Edge List: ??

Total Cost: ??

Q2 - Adjacency Matrices and Lists

- Graphs are interconnected webs of nodes.
- Graphs in code are typically represented in 2 ways: adjacency matrices and adjacency lists

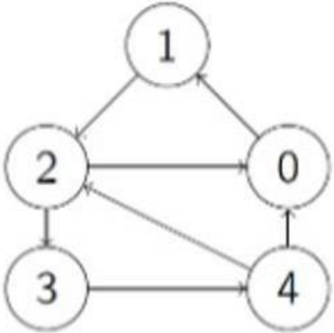
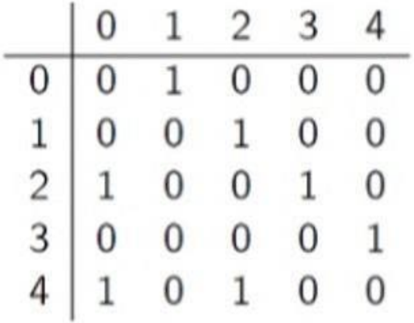


Figure 1: Graph



	0	1	2	3	4
0	0	1	0	0	0
1	0	0	1	0	0
2	1	0	0	1	0
3	0	0	0	0	1
4	1	0	1	0	0

Figure 2: Adjacency Matrix

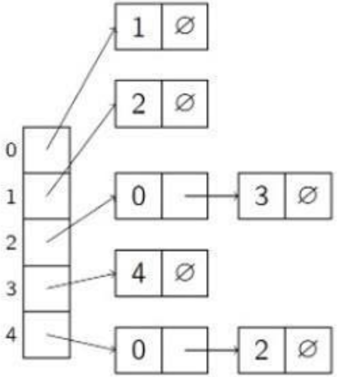


Figure 3: Adjacency List

Q2 - Adjacency Matrices and Lists

- Task is to write a program which will create a graph and then print it as both an adjacency matrix and as an adjacency list.
- You will need the files: **Graph.java** and **GraphTest.java**.
- **GraphTest.java**
  - Do not change anything in this file.
  - Notice randomMatrix() carefully.
    - Generates a random double-array of integers, with the size given, representing the adjacency matrix of a graph.
    - This random double array is then taken as the only argument for the **Graph** constructor.
- The task is to complete all of the necessary methods within the **Graph.java** file.

Q2 - Adjacency Matrices and Lists

Your task is to complete the followings in the Graph.java file :

- One constructor: **Graph()**
- Three methods :
    1. **generateAdjList()**
    2. **printMatrix()**
    3. **printList()**

Check lecture 12 - Page 27-35

Q2 - Adjacency Matrices and Lists

Part A – 5 Marks
Graph() and generateAdjList():

- Write the constructor of the graph that takes in the adjacency matrix (int[][]) as its only argument.
- It then initializes the Graph (i.e., initialize the values of numVertices and adjMatrix).
- Have to call generateAdjList() from the constructor. This method takes no arguments, but uses the data in the newly saved adjMatrix to populate the adjListArray.

- **Note:** *Please use Java's default LinkedList class. A sample code for adding four number into an array of LinkedList is given in the instruction file. If you prefer to use a list instead of an array, feel free to make corresponding changes in the code.*

Q2 - Adjacency Matrices and Lists

Part B - 2 marks

printMatrix(): Takes no arguments, and prints the adjacency matrix (adjMatrix) in the format shown below.

```
Adjacency matrix (4 nodes):
         0 1 1 1
         0 0 0 1
         0 0 0 1
         1 0 1 0
```

Part C - 3 marks

printList(): Takes no arguments, and prints the adjacency list (adjListArray) in the format shown below.

```
Adjacency list of vertex 0: 1, 2, 3
Adjacency list of vertex 1: 3
Adjacency list of vertex 2: 3
Adjacency list of vertex 3: 0, 2
```

Note that since the graphs are randomly generated, the values will not be identical to the output shown.

Q3 - DFS

Count Starting Nodes [2 + 3 (bonus) marks]

- Write a program which will take an undirected graph and visit all the vertices using a Depth-First Search (DFS).
- Note: With some graphs it is not possible to reach every vertex from one starting vertex (i.e. a disconnected graph). Therefore, you may need to select more than one starting vertex to complete the graph traversal.
- Provided files: **DFSGraph.java** and **DFSTest.java**.

Q3 - DFS

Your task is to complete **two** methods in the DFSGraph.java file: **printList()** and **countStartingNodes()**

Refresh your concept of Recursion.

Check lecture 13 - Page 3-26

Q3 - DFS

Part A - 2 marks

printList():

- This method prints out the graph in an adjacency-list format.
- You may get help from your printList() method from Question 2.

Q3 - DFS

Part B - 3 marks (BONUS)
countStartingNodes(): Finds the number of vertices that need to be selected as starting vertices to traverse the entire graph.

Check Lecture 12 slide 24.

- For the left graph, we can select one vertex and traverse the entire graph.
- For the right graph, you need to select minimum two vertices to traverse the entire graph.
- Note that you must select vertices in ascending order (i.e. you must try starting from vertex 1 before trying vertex 2).
- Hint: you may want to use a boolean flag to keep track of which vertices have already been visited.

Q3 - DFS

Part B - 3 marks (BONUS): countStartingNodes()

- Within this method, you should call DFS().
- This method traverses the graph from the given starting vertex, maintaining a record of which nodes have been visited.
- Recursion should be used with this method.
- *Note that you may wish to change the return type.*
- When the traversal has been completed, countStartingNodes() should print the number of starting vertices selected and a list of the selected vertices.