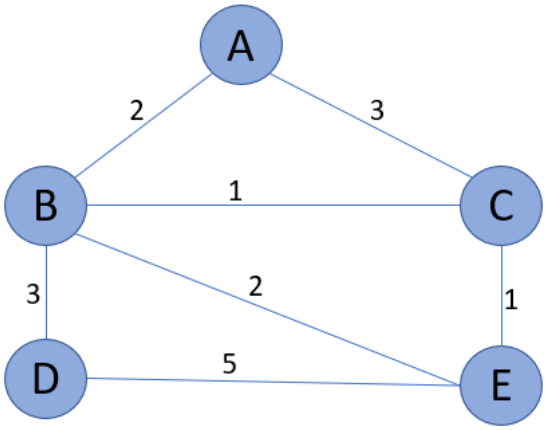
Assignment: Graph Algorithms – II

1. **Draw Minimum Spanning Tree**
   a. Draw minimum spanning tree for the below graph.
   b. Draw spanning Tree that is not minimum



2. **MST implementation:**
   a. Implement Prims' algorithm Name your function **Prims(G).** Include function in the file **MST.PY.** You can either use brute force approach or priority queue. Try to see if you can come up with a solution using priority queue.

      Input: a graph represented as an adjacency matrix
      For example, the graph in the Exploration would be represented as the below (where index 0 is A, index 1 is B, etc.).

```
input = [
  [0, 8, 5, 0, 0, 0, 0],
  [8, 0, 10, 2, 18, 0, 0],
  [5, 10, 0, 3, 0, 16, 0],
  [0, 2, 3, 0, 12, 30, 14],
  [0, 18, 0, 12, 0, 0, 4],
  [0, 0, 16, 30, 0, 0, 26],
  [0, 0, 0, 14, 4, 26, 0]
]
```

      Output: a list of tuples, wherein each tuple represents an edge of the MST as (v1, v2, weight)
      For example, the MST of the graph in the Exploration would be represented as the below.
      output = [(0, 2, 5), (2, 3, 3), (3, 1, 2), (3, 4, 12), (2, 5, 16), (4, 6, 4)]

      Note: the order of edge tuples within the output does not matter; additionally, the order of vertices within each edge does not matter. For example, another valid

output would be below (v1 and v2 in the first edge are flip-flopped; the last two edges in the list are flip-flopped).
output = [(2, 0, 5), (2, 3, 3), (3, 1, 2), (3, 4, 12), (4, 6, 4), (2, 5, 16)]

b. What is the difference between the Kruskal's and the Prim's algorithm?

3. Apply Graph traversal to solve a problem (Portfolio Project Problem):

You are given a 2-D puzzle of size MxN, that has N rows and M column (M and N can be different). Each cell in the puzzle is either empty or has a barrier. An empty cell is marked by '-' (hyphen) and the one with a barrier is marked by '#'. You are given two coordinates from the puzzle (a,b) and (x,y). You are currently located at (a,b) and want to reach (x,y). You can move only in the following directions.
L: move to left cell from the current cell
R: move to right cell from the current cell
U: move to upper cell from the current cell
D: move to the lower cell from the current cell

You can move to only an empty cell and cannot move to a cell with a barrier in it. Your goal is to reach the destination cells covering the minimum number of cells as you travel from the starting cell.

Example Board:

-	-	-	-	-
-	-	#	-	-
-	-	-	-	-
#	-	#	#	-
-	#	-	-	-

Input: board, source, destination.

Puzzle: A list of lists, each list represents a row in the rectangular puzzle. Each element is either '-' for empty (passable) or '#' for obstacle (impassable). The same as in the example.
Example:

```
Puzzle = [
    ['-', '-', '-', '-', '-'],
    ['-', '-', '#', '-', '-'],
    ['-', '-', '-', '-', '-'],
    ['#', '-', '#', '#', '-'],
    ['-', '#', '-', '-', '-']
]
```

source: A tuple representing the indices of the starting position, e.g. for the upper right corner, source=(0, 4).
destination: A tuple representing the indices of the goal position, e.g. for the lower right corner, goal=(4, 4).

Output: A list of tuples representing the indices of each position in the path. The first tuple should be the starting position, or source, and the last tuple should be the destination. If there is no valid path, None should be returned. Not an empty list, but the None object. If source and destination are same return the same cell.

Note: The order of these tuples matters, as they encode a path. Each position in the path must be empty (correspond to a '-' on the board) and adjacent to the previous position.

Example 1 (consider above puzzle)
Input: puzzle, (0,2), (2,2)
Output: [(0, 2), (0, 1), (1, 1), (2, 1), (2, 2)]

Example 2 (consider above puzzle)
Input: puzzle, (0,0), (4,4)
Output: [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (2, 4), (3, 4), (4, 4)]

Example 3: (consider above puzzle)
Input: puzzle, (0,0), (4,0)
Output: None

Example 4: (consider above puzzle)
Input: puzzle, (0,0), (0,0)
Output: [(0,0)]

a. Describe an algorithm to solve the above problem.
b. Implement your solution in a function **solve_puzzle(Board, Source, Destination).** Name your file **Puzzle.py**
c. What is the time complexity of your solution?
d. **(Extra Credit):** For the above puzzle in addition to the output return the directions as well in the form of a string.

For above example 1

Output: ([(0, 2), (0, 1), (1, 1), (2, 1), (2, 2)], 'LDDR')

For above example 2

Output: ([(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (2, 4), (3, 4), (4, 4)], 'RRRRDDDD')

For above example 4

Output: ([(0, 0)], '')