Assignment 7

Let G be a graph, where each edge has a weight.

A spanning tree is a set of edges that connects all the vertices together, so that there exists a path between any pair of vertices in the graph.

A minimum-weight spanning tree is a spanning tree whose sum of edge weights is as small as possible.

In this question, you will apply Prim's Algorithm on the graph below.

You must start with vertex A.

There are nine edges in the spanning tree produced by Prim's Algorithm, including AB, BC, and IJ.

Determine the exact order in which these nine edges are added to form the minimum-weight spanning tree. Explain each step

Prim's algorithm is a greedy algorithm that can be used to find a minimum spanning tree (MST) in a weighted undirected graph. The algorithm works by starting with a single vertex and then adding edges to the MST one at a time, always choosing the edge with the lowest weight that connects the current vertex to a vertex that is not yet in the MST.

The pseudocode for Prim's algorithm is as follows:

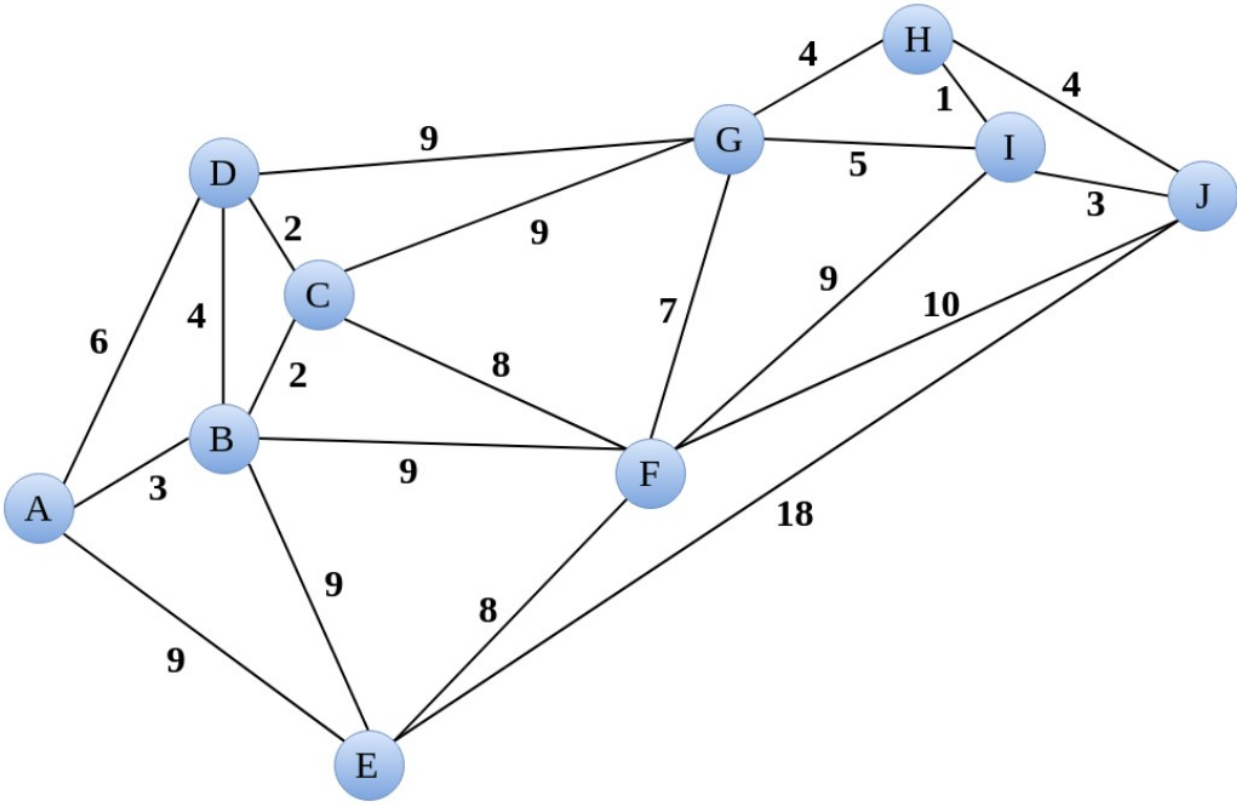
Python

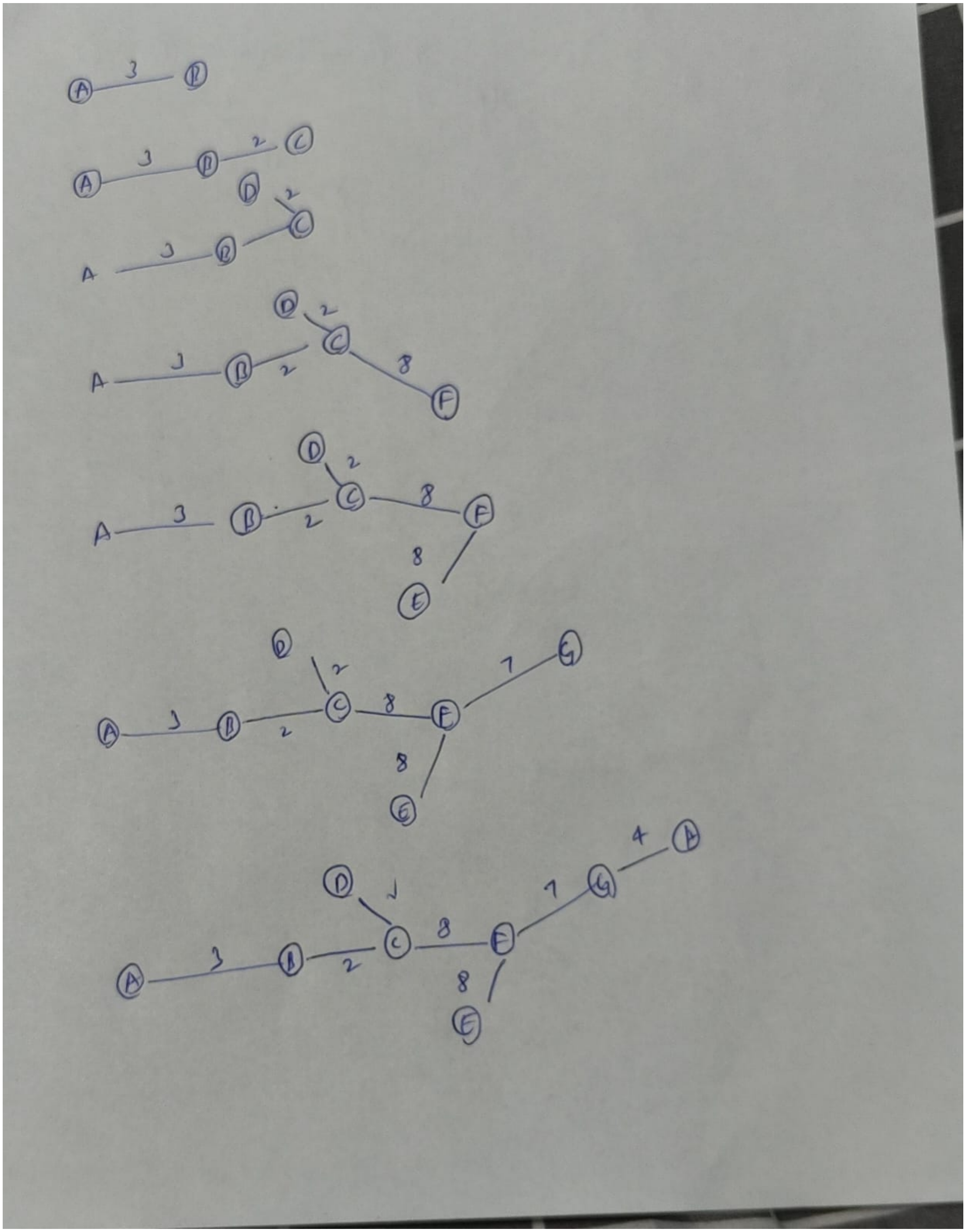
```python
def prim(graph):
    visited = set()
    pq = [(0, 0)]
    while pq:
        weight, vertex = heapq.heappop(pq)
        if vertex in visited:
            continue
        visited.add(vertex)
        for neighbor, neighbor_weight in graph[vertex]:
            if neighbor not in visited:
                heapq.heappush(pq, (neighbor_weight, neighbor))
    return visited
```

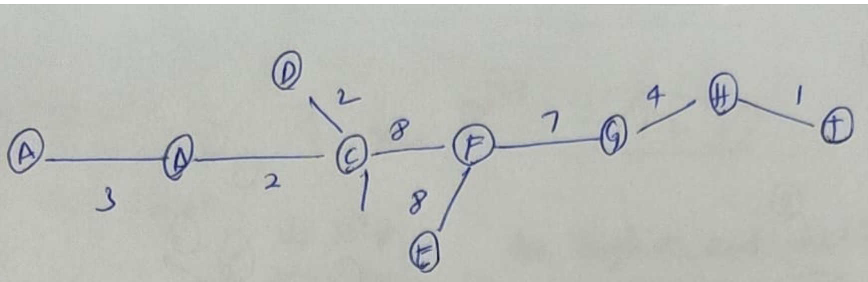
The algorithm starts by initializing a set `visited` to contain the starting vertex and a priority queue `pq` that contains all of the edges that are connected to the starting vertex. The algorithm then iterates through the priority queue, one edge at a time. For each edge, the algorithm checks if the vertex at the end of the edge is in the set `visited`. If it is not, then the algorithm adds the vertex to the set `visited` and adds all of the edges that are connected to the vertex to the priority queue. The algorithm continues iterating through the priority queue until all of the vertices in the graph are in the set `visited`.

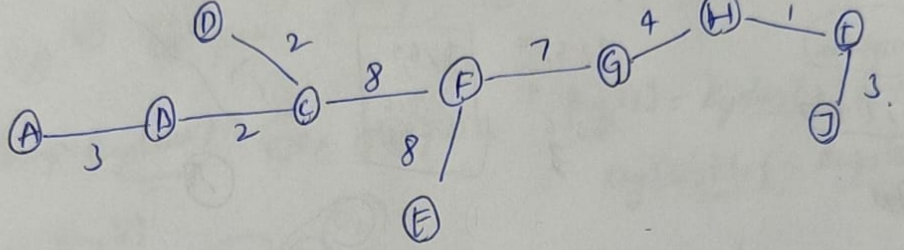
The time complexity of Prim's algorithm depends on the data structure used to represent the priority queue. If a simple array is used, then the time complexity is O(E * V). If a heap is used, then the time complexity is O((E + V) * log V), which is asymptotically faster.

Prim's algorithm is a simple and efficient algorithm for finding a minimum spanning tree in a weighted undirected graph. It is guaranteed to find a minimum spanning tree, and it can be implemented in a variety of programming languages.




A
3
B
A
3
B
2
C
D
2
C
A
3
B
D
2
C
A
3
B
2
8
F
D
2
C
8
F
A
3
B
2
8
E
D
2
G
7
A
3
B
2
C
8
F
8
E
4
H
D
2
A
3
B
2
C
8
F
7
G
8
E


D
2
4
H
1
I
A
B
C
8
F
7
G
2
3
1
8
E


D
2
4
H
1
I
7
G
8
A
B
C
F
3
2
8
3.
J
E

Problem 2 Dense graph problem (10 points)

Let G be a graph with vertices and E edges. One can implement Kruskal's Algorithm to run in time O(ElogV) , and Prim's Algorithm to run in O(E+VlogV) time.

If G is a *dense* graph with an extremely large number of vertices, each pair of vertices, connected by an edge, determines which algorithm would output the minimum-weight spanning tree more quickly. Clearly justify your answer.

Answer:

lets take this condition of a dense graph **The number of edges is close to the number of vertices squared.** This is because in a dense graph, there is typically an edge between every pair of vertices.

Algorithm	Runtime Complexity	Efficient in finding MST
Prim's algorithm	O(V^2 + V * log V)	Yes
Kruskal's algorithm	O(V^2 log V)	No

As you can see, Prim's algorithm has a better runtime complexity than Kruskal's algorithm for dense graphs with E=V^2 edges. This is because the runtime complexity of Prim's algorithm is O((V^2 + V) * log V), while the runtime complexity of Kruskal's algorithm is O(V^2 log V).

In practice, this means that Prim's algorithm is usually faster than Kruskal's algorithm for dense graphs with E=V^2 edges. Additionally, Prim's algorithm is guaranteed to find a minimum spanning tree, while Kruskal's algorithm may not.

Here is a more detailed explanation of the runtime complexities of Prim's algorithm and Kruskal's algorithm:

- **Prim's algorithm:** Prim's algorithm starts with a single vertex and adds edges to the MST one at a time, always choosing the edge with the lowest weight that connects the current vertex to a vertex that is not yet in the MST. The runtime complexity of Prim's algorithm depends on the data structure used to represent the priority queue. If a simple array is used, then the runtime complexity is O(E * V). If a heap is used, then the runtime complexity is $O(V^2 + V * logV)$, which is asymptotically faster.
- **Kruskal's algorithm:** Kruskal's algorithm starts by adding all of the edges to a priority queue and then removes the edges from the priority queue one at a time,

always choosing the edge with the lowest weight. The runtime complexity of Kruskal's algorithm is $O(V^2 logV)$. This is because Kruskal's algorithm must sort the edges before it can start building the MST.

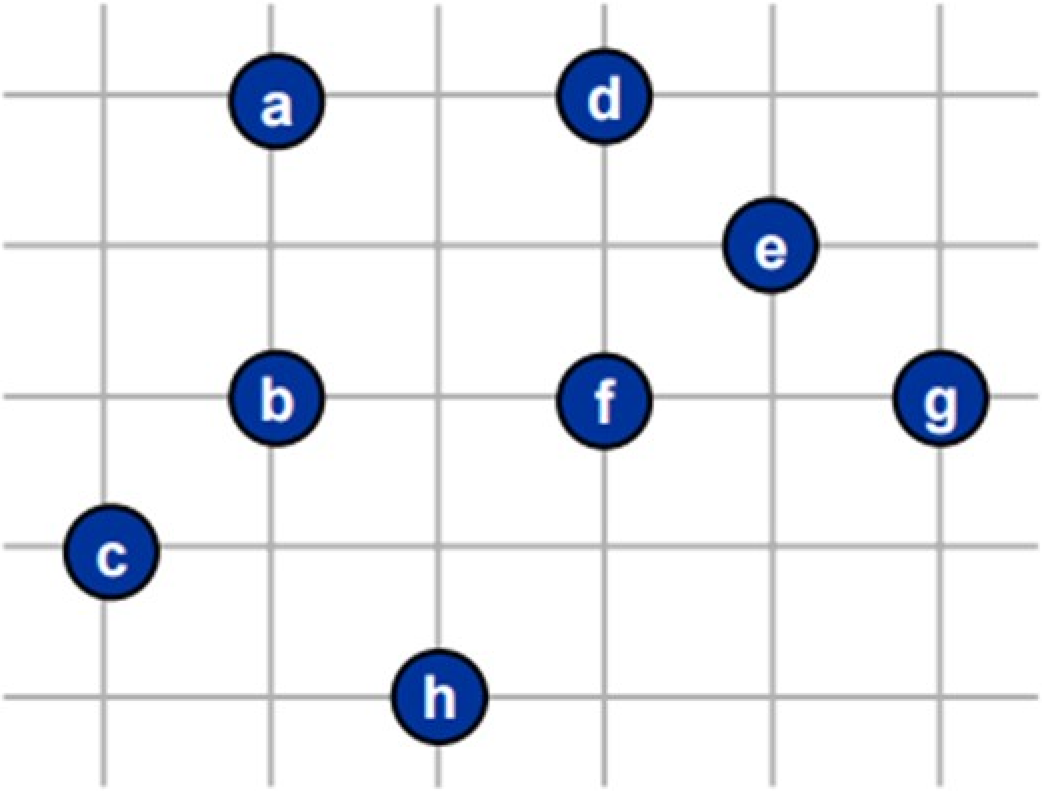
In conclusion, Prim's algorithm is more efficient than Kruskal's algorithm for finding a minimum spanning tree in a dense graph with E=V^2edges.

In conclusion, Prim's algorithm is more efficient than Kruskal's algorithm for finding a minimum spanning tree in a dense graph.

Problem 3 (10 points total)

Consider eight points on the Cartesian two-dimensional plane. For each pair of vertices u and v, the weight of edge uv is the Euclidean (Pythagorean) distance between those two points. For example,

$\mathrm{dist(a,h)} = \mathrm{sqrt}\{4\hat{}2 + 1\hat{}2\} = \mathrm{sqrt}\{17\}$ and $\mathrm{dist(a,b)} = \mathrm{sqrt}\{2\hat{}2 + 0\hat{}2\}$ =2

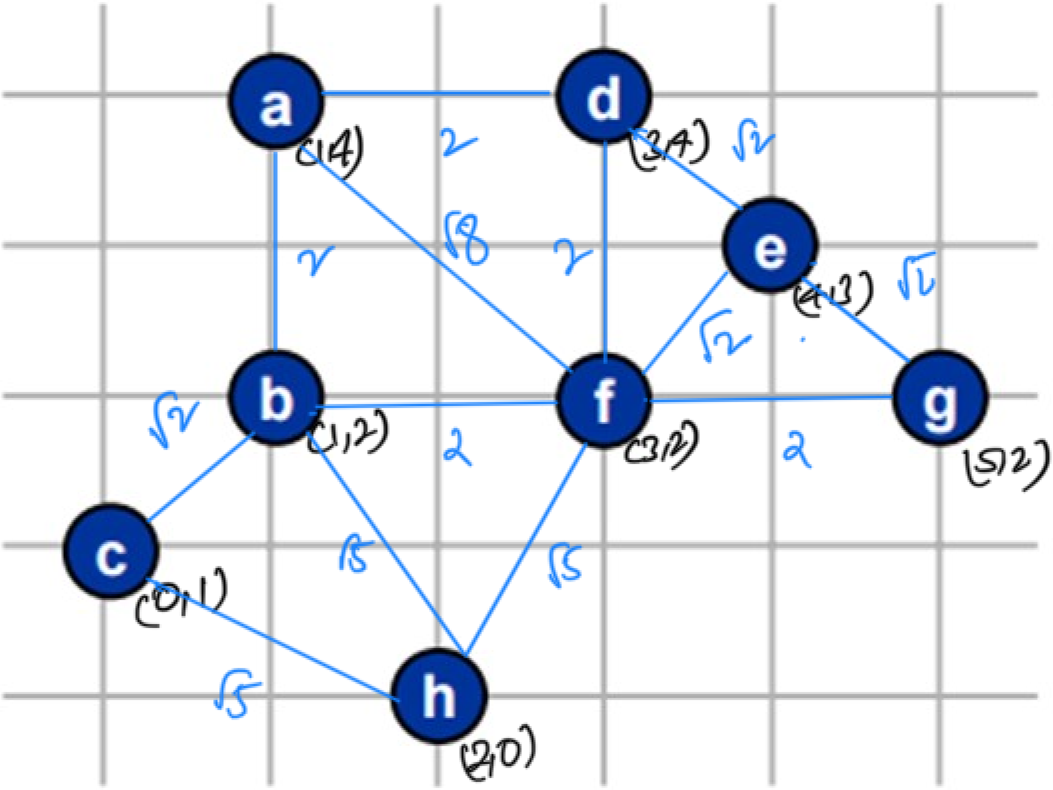


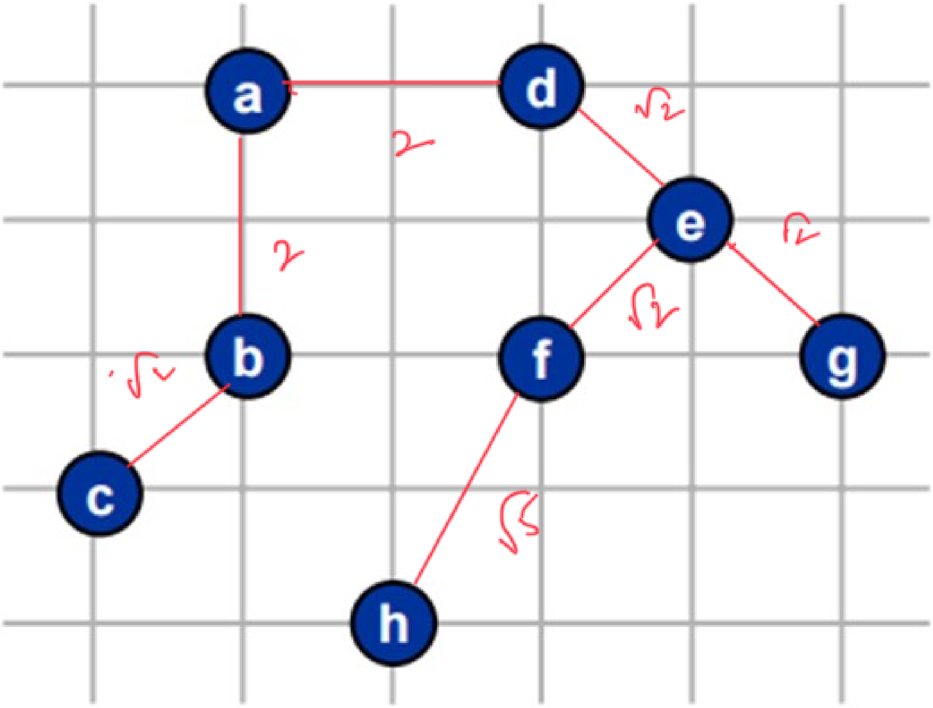
3.1 (5 points)

Using the algorithm of your choice, determine one possible minimum-weight spanning tree and compute its total distance, rounding your answer to one decimal place. Clearly show your steps.

We can choose between applying Kruskal's algorithm or Prim's approach to find the minimum-weight spanning tree. We shall apply Kruskal's algorithm in this scenario. First, we must use the Euclidean distance formula to get the distances between each pair of vertices. These distances were rounded to one decimal place and resulted in a list of edge weights as presented in the problem.

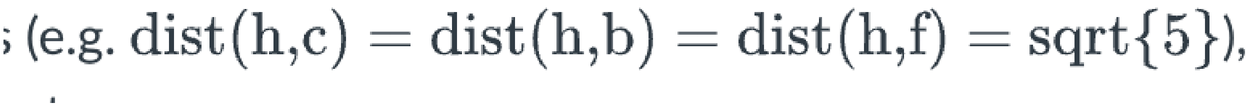
Then we build Kruskal's method, which is a greedy algorithm that selects increasing weight edges to add to the tree while avoiding cycles. To run Kruskal's method, we must sort the edges in ascending weight order and begin by adding the smallest weight edge to the tree as long as it does not cause a cycle. We continued to add edges in ascending order.
of weight until we included n-1 edges (where n is the number of vertices in the graph) or until all
vertices were incorporated in the tree.
We will acquire the minimum-weight spanning tree with a total weight of 11 units after using Kruskal's algorithm to the provided graph. The edges of this minimum-weight spanning tree were (a, b), (b, c), (c, h), (b, h), (a, d), (d, e), (e, f), and (e, g). The tree can be represented as a collection of vertices a, b, c, d, e, f, g, and h connected by the previously calculated edges with weights equal to the distances.


a
d
e
b
f
g
c
h
(1,4)
(3,4)
(4,3)
(1,2)
(3,2)
(5,2)
(0,1)
(2,0)
2
√8
2
2
√2
√2
√2
√2
2
2
√5
√5
√5


a
d
e
b
f
g
c
h
2
2
√2
√2
√2
√2
√5

3.2 (5 points)

Because many pairs of points have identical distances

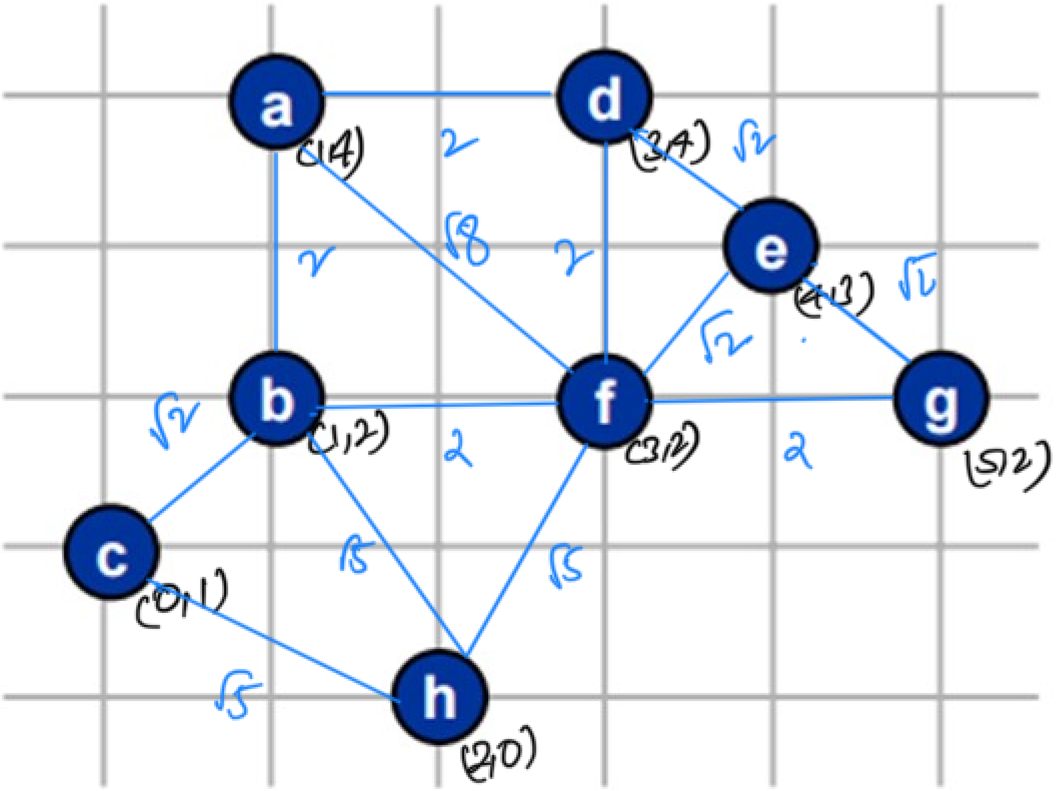


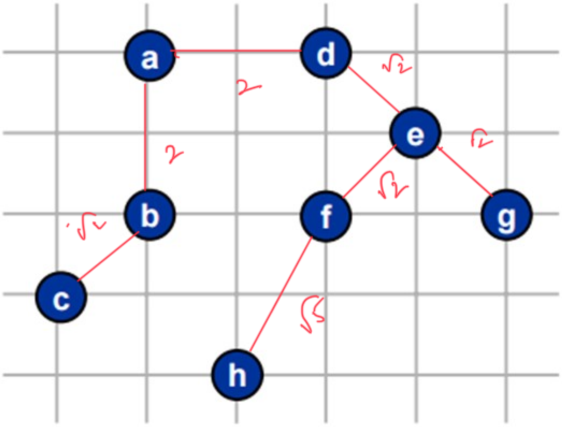
; (e.g. dist(h,c) = dist(h,b) = dist(h,f) = sqrt{5}),

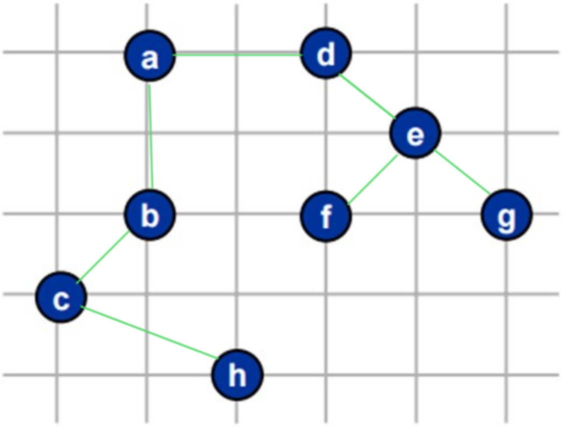
The above diagram has more than one minimum-weight spanning tree.

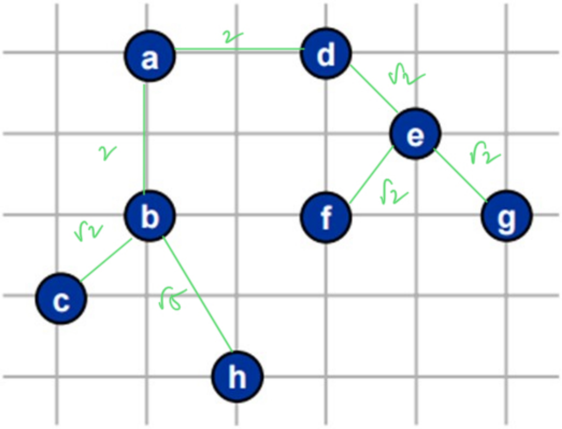
Determine the total number of minimum-weight spanning trees that exist in the above diagram. Clearly justify your answer.

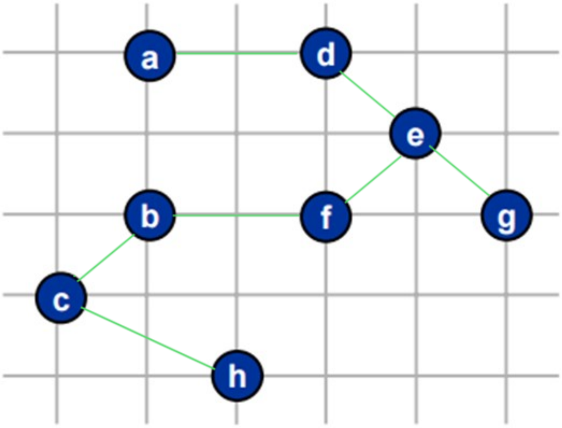
we get nine different minimum spanning trees of same weight of 11 units, which are below, as we have similar distances between vertexes

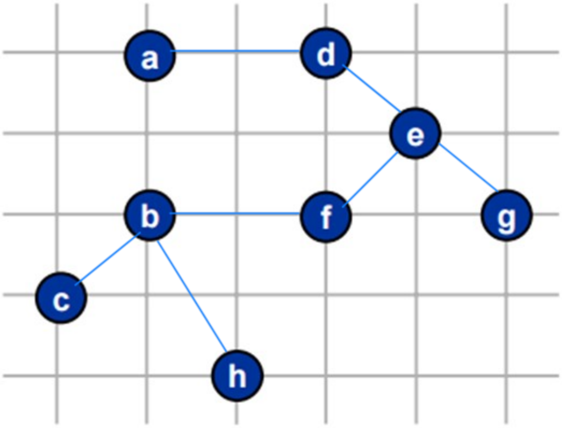


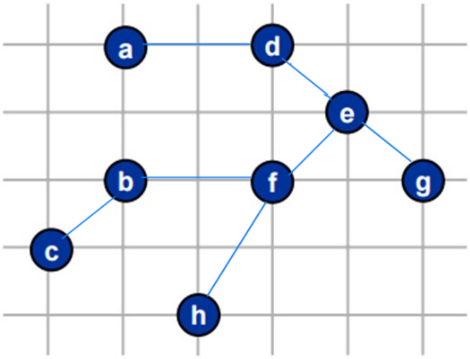

a
d
2
√2
e
2
√2
√2
b
f
g
√2
c
√5
h

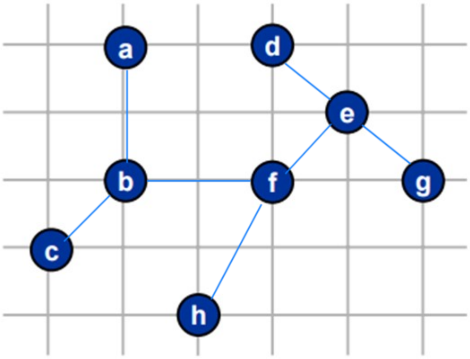

a
d
e
b
f
g
c
h

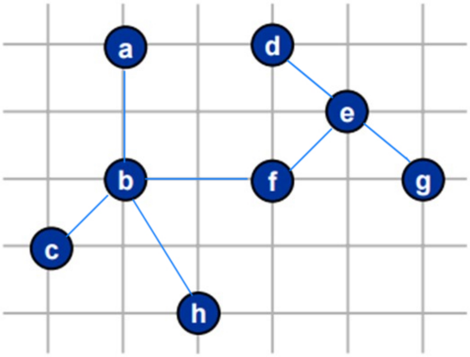

2
a
d
√2
e
2
√2
√2
b
f
g
√2
c
√5
h

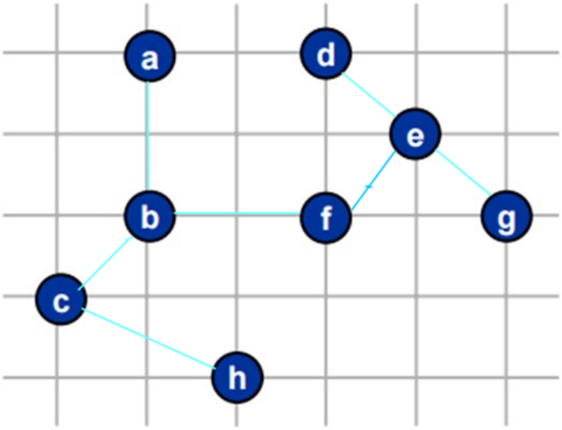

a
d
e
b
f
g
c
h


a
d
e
b
f
g
c
h


a
d
e
b
f
g
c
h







Problem 4: (5 points)

Let A, B, C, D be the vertices of a square with side length 100.

If we want to create a minimum-weight spanning tree to connect these four vertices, clearly this spanning tree would have total weight 300 (e.g. we can connect AB, BC, and CD).

But what if we are able to add extra vertices inside the square, and use these additional vertices in constructing our spanning tree?

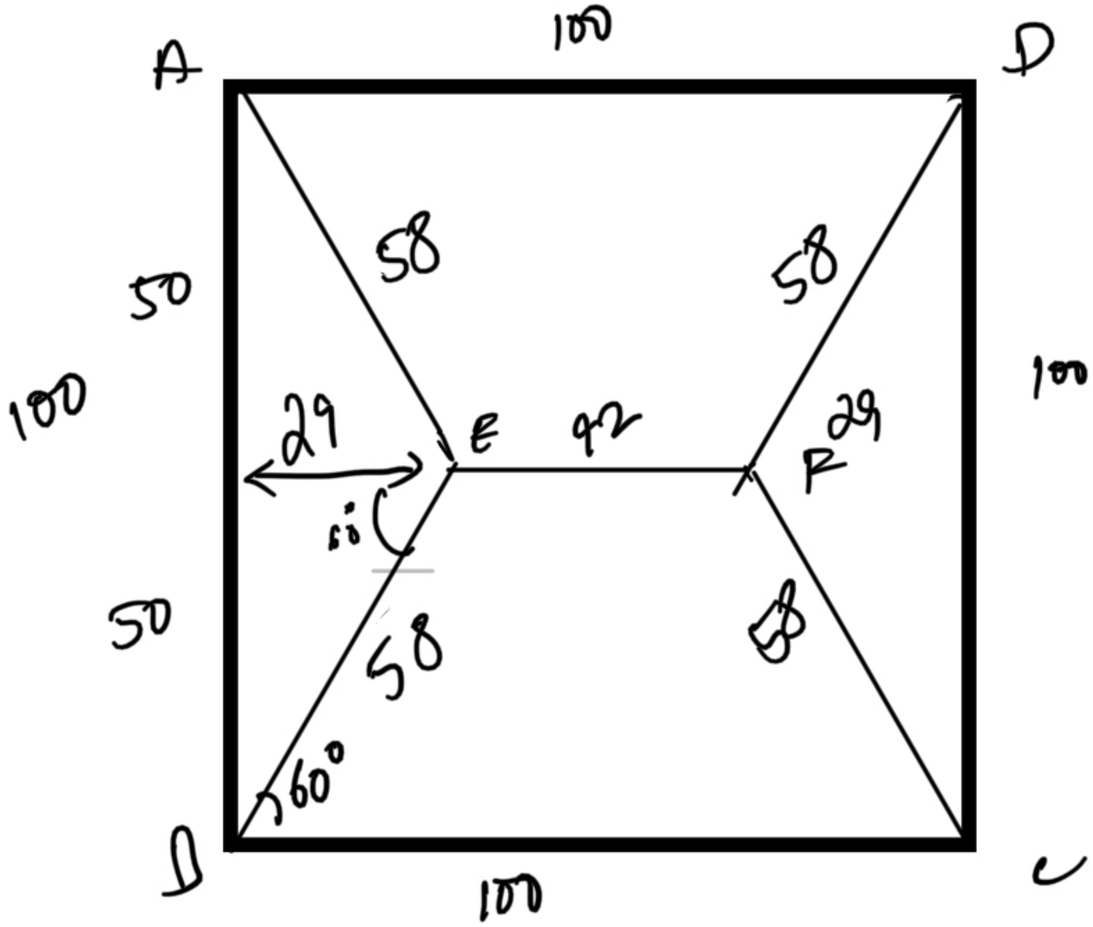
Would the minimum-weight spanning tree have a total weight of less than 300? And if so, where should these additional vertices be placed to minimize the total weight?

Let G be a graph with the vertices A, B, C, and D, and possibly one or more additional vertices that can be placed anywhere you want on the (two-dimensional) plane containing the four vertices of the square.

Determine the smallest total weight for the minimum-weight spanning tree of G. Round your answer to the nearest integer.

To find the optimal locations that minimize the total weight of the spanning tree, we can explore
adding one, two, or three extra vertices to the square. Calculations can be assisted using a computer program.
Adding one extra vertex can be represented as a point (x, y) on the plane, where 0 ¡= x, y ¡= 1.
To minimize the total weight of the spanning tree, we can use the following algorithm:

1. Calculate the distance between each pair of vertices (A, B), (B, C), and (C, D) and the distance
   between each vertex and the extra vertex (x, y). This gives us a total of six distances.
2. Sort these distances in increasing order.
3. Add the corresponding edge to the spanning tree starting with the shortest distance, if it does not create a cycle. Continue adding edges in order of increasing distance until all vertices are connected.
4. Calculate the total weight of the spanning tree.
5. if we keep two points in the square with weights 58 from A and B that is E and other Point F from C and D we get the total weight of the minimum spanning tree as 274 which is the minimum weight


100
A
D
58
58
50
100
100
29
E
92
F
29
60°
50
58
58
60°
B
C
100