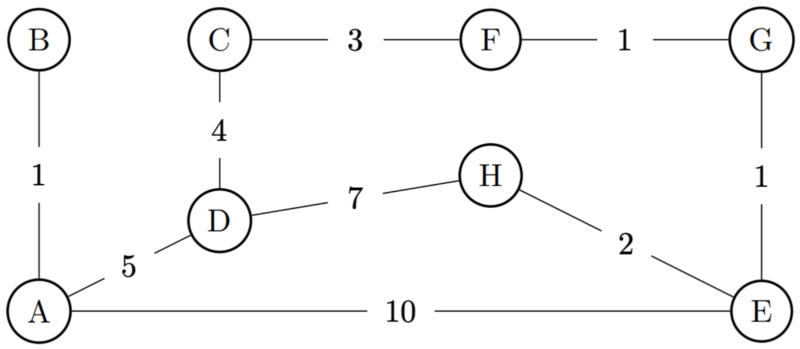
Question no. 4

Run Kruskal’s algorithm on the above graph to produce a minimum spanning tree (MST). The weights are the numbers drawn on top of each edge. Draw the minimum spanning tree Kruskal’s algorithm finds, indicate the order in which edges are added to it, and note the total weight of the MST. You may break ties arbitrarily. Provide the output, not code that produces the output.

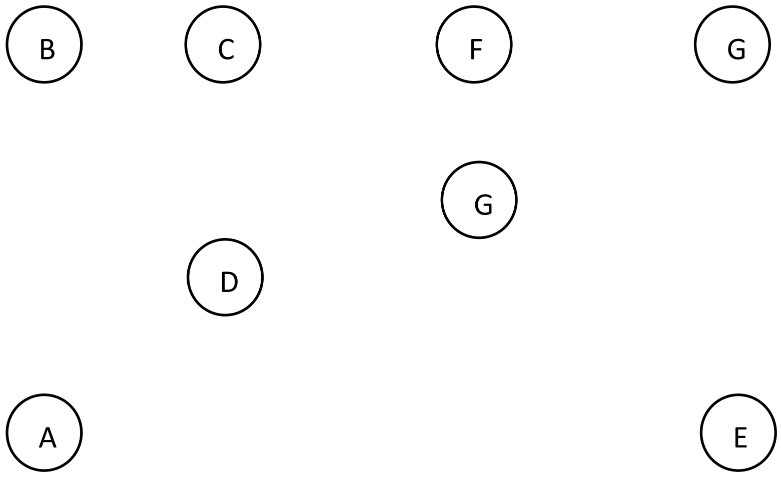


Solution:

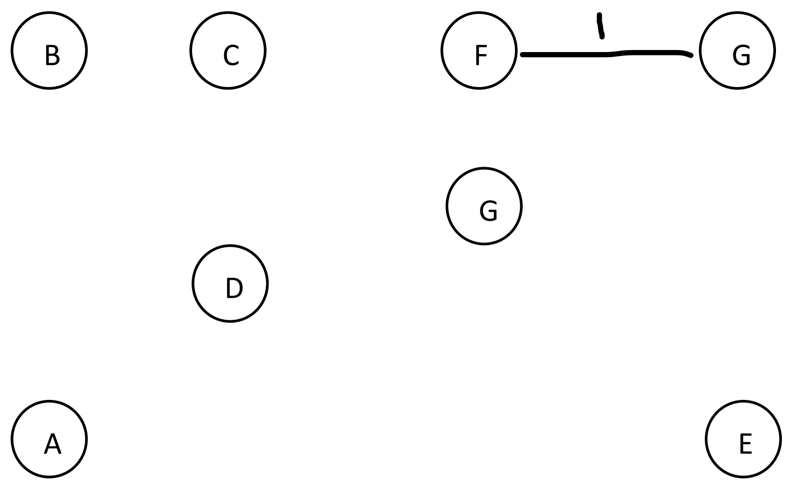
We have:
Kruskal’s Algorithm:
-Start with the empty tree T
-Repeat: Add the minimum weight edge to T unless it forms a cycle

Now, starting with the empty tree from the nodes given above:

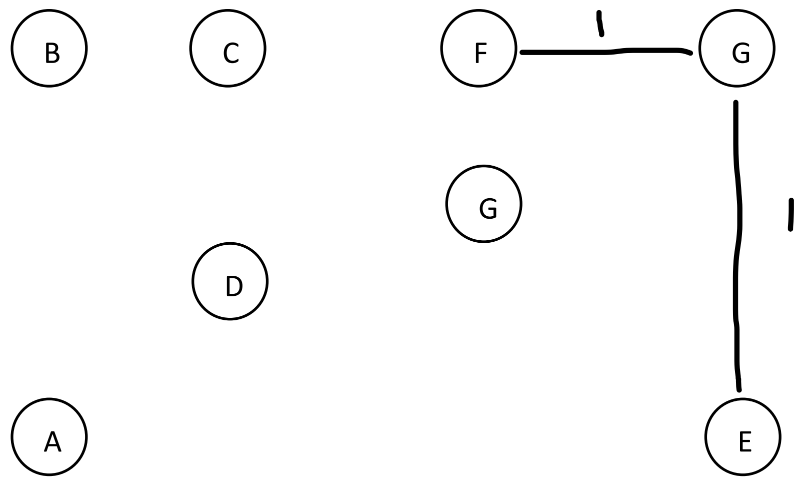
Step 1:



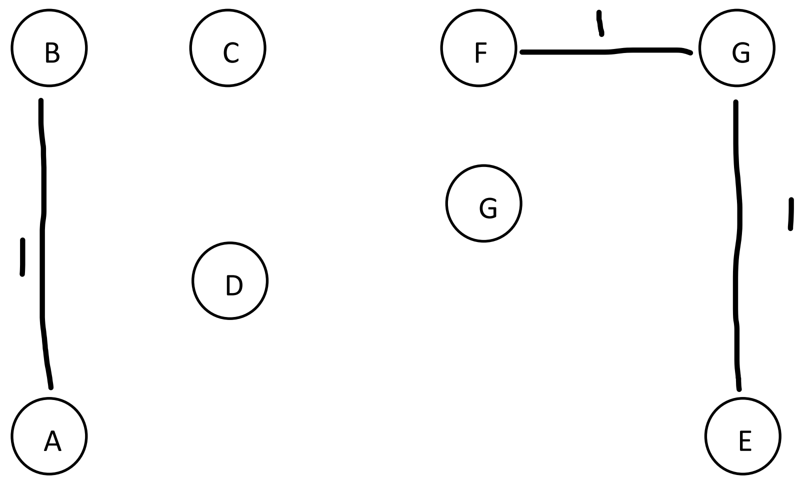
Step 2:



Step 2:

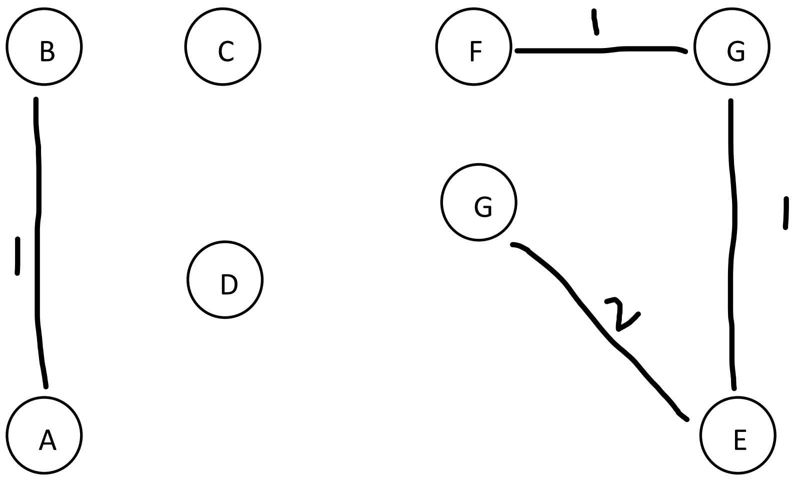


Step 3:

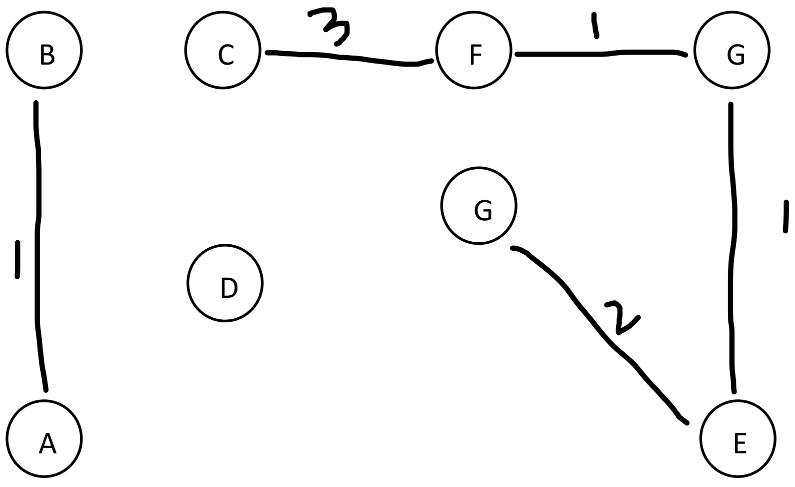


We are finished with weight 1; however, we will repeat unless the graph is cyclic.

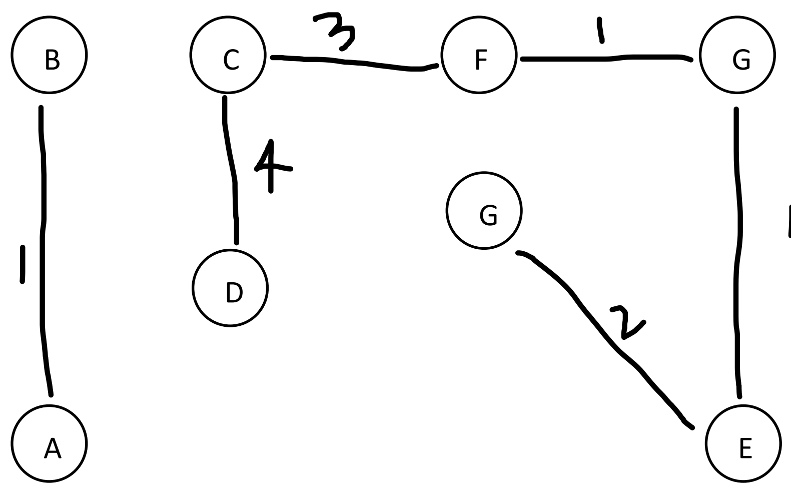
Step 4:



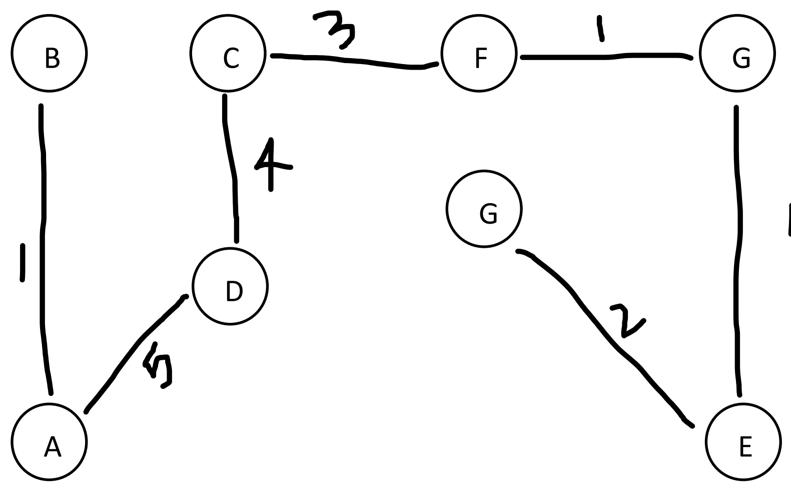
Step 5:



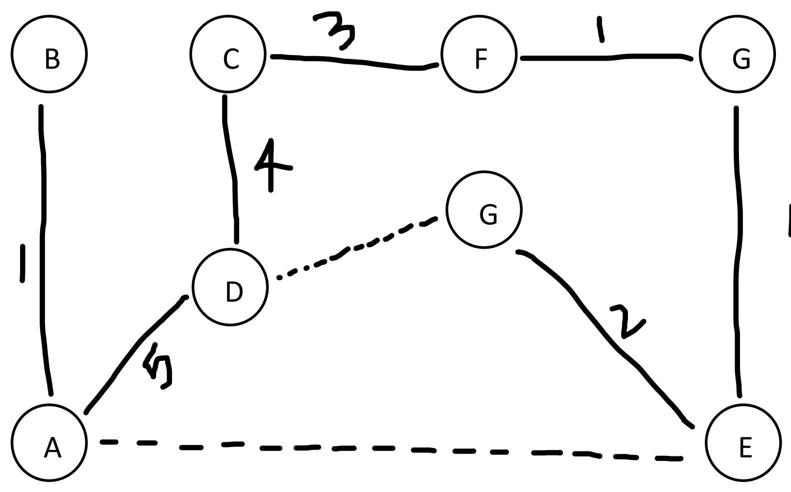
Step 6:



Step 7:



Our graph is now connected and is minimum spanning tree.



Hence, above graph is minimum spanning Tree having total weight of 1 + 5+4 + 3+1+1+2 = 17

Question 1:

Theory vs. Practice (9 points)
(a) List 3 reasons why asymptotic analysis may be misleading with respect to actual performance in practice. (3 points)

(b) Suppose finding a particular element in a binary search tree with 1,000 elements takes 5 seconds. Given what you know about the asymptotic complexity of search in a binary search tree, how long would you guess finding the same element in a search tree with 10,000 elements takes? Explain your reasoning. (3 points)

(c) You measure the time with 10,000 elements and it takes 100 seconds! List 3 reasons why this could be the case, given that reasoning with the asymptotic complexity suggests a different time. (3 points)

Solution:

(a)
The three reasons why asymptotic analysis may be misleading with respect to actual performance in practice are:

i. While measuring the asymptotic complexity of the algorithms, we generally ignore all the constants involved , for example, let say we have two different algorithms that takes 0(n) and 0(10000n) time respectively on a machine; however, we write both of the algorithms runs asymptotically in same time of 0(n) which can often be misleading as we won't be able to distinguish which algorithms will run better and one of the algorithms may take more time to run while working with same input size. This can impact our actual running time in some cases as algorithm having complexity 0(n) may slightly run faster than the algorithm with time complexity 0(10000n).

ii. While giving the definition of different asymptotic bounds, like f(x) = 0(g(x)), we always suppose our input size to be larger than n_0 (some constant); however, if we never looked at out input size and input size became less than some constant value in our algorithms, then it might not run effectively. Even asymptotically slower algorithms may run faster depending upon our input size.

iii. Asymptotic complexity may not always give you the running time of the algorithms. The running time of the algorithms may depend upon several things like how fast the computer is, programming language, compilers, hardware. We only consider the input size while calculating the asymptotic complexity of any programs which may not be the actual running time of the algorithms.

(b)

Suppose finding a particular element in a binary search tree with 1,000 elements takes 5 seconds. Given what you know about the asymptotic complexity of search in a binary search tree, how long would you guess finding the same element in a search tree with 10,000 elements takes? Explain your reasoning. (3 points)

We know that average time complexity to search in binary search tree is 0(logn), now we will use this formula to calculate the time required to search the element in a search tree with 10,000 elements.

We are given than to find the particular element in binary search tree with 1000 elements, it takes around 5 seconds.
So, we can expect that to find the same element with 10,000 elements it would take Log(10000)/log(1000) = 1.33 times more.

Hence, it would take around 5 * 1.33 = 6.67 s to find that particular element in the BST with 10,000 elements.

This is for the general case. If the BST is skewed, then we should use 0(n) as our time complexity which will give is us different time to find the particular element.

(c)

You measure the time with 10,000 elements and it takes 100 seconds! List 3 reasons why this could be the case, given that reasoning with the asymptotic complexity suggests a different time. (3 points)

The three reasons are:

i. Since, we are taking the general case for the time complexity, our actual elements in the BST may be left or right skewed, which will greatly affect out time measured by using time complexity. Or, it may happen that our, particular element is in very top of the BST, so we don't have to search any further or it might be the last element, so it would take very long to search it compared to the general case.

ii. Since, the size of the BST is very large, some low-end PC may suffer badly. With the increment of memory usage in RAM, specific PC may have to access even virtual memory to store the tree structure and elements themselves, which is slower than using physical memory.

iii. Also, different hardware and software used to run the algorithm may also give different time. Hence, the search is hugely affected by the specific hardware, software, operating system, memory usage as well as programming language.

2. Graph Properties (9 points)

(a) Prove that if two graphs A and B do not have the same number of nodes, they cannot be isomorphic. (3 points)
(b) Prove that if two graphs A and B have the same number of nodes and are completely connected, they must be isomorphic. (3 points)
(c) Prove that if two graphs A and B are isomorphic, they do not have to be completely connected. (3 points)
You need to give complete, formal proofs – state all your assumptions and make sure that you've explained every step of your reasoning.
 Hint: A good way to start is by writing down the definitions for everything related to what you want to prove.

(a) Prove that if two graphs A and B do not have the same number of nodes, they cannot be isomorphic. (3 points)

Formal Proof:

We have the formal definition:

$G_1 = (V_1, E_1)$ is isomorphic to $G_2 = (V_2, E_2)$ if there is a one-to-one and onto function (bijection) $f : V_1 \rightarrow V_2$ such that $(u, v) \in E_1$ iff $(f(u), f(v)) \in E_2$.

Now,
Let Graph A = {V, E} and Graph B = {V*, E*}

We need to prove that if graph A and B do not have the same number of nodes, they are not isomorphic.

Let assume,
Graph A has i nodes and Graph B has less than i nodes.
From the definition:
We have,
If the graph A and B are isomorphic then,
There is bijection function f: V -> V* such that (u,v) belongs to E iff (f(u), f(v)) belongs to E*.
Let say,
V = {u1,u2,u3,…..un} and
V* = {f(u1), f(u2), …….f(un-1)}
However, we have contradiction here,
Since, number of nodes in in A and B are not equal, we don't have the image of un to V*, hence, the function f:V-> V* is not bijection function which doesn't make the graph A and B isomorphic.

Hence, if two graph have different numbers of nodes, then they are not isomorphic.

(b)
Prove that if two graphs A and B have the same number of nodes and are completely connected, they must be isomorphic. (3 points)

$G_1 = (V_1, E_1)$ is isomorphic to $G_2 = (V_2, E_2)$ if there is a one-to-one and onto function (bijection) $f : V_1 \to V_2$ such that $(u, v) \in E_1$ iff $(f(u), f(v)) \in E_2$.

And two graphs A and B are isomorphic, if they are
i. Equal numbers of vertices
ii. Equal number of edges
iii. Same degree sequences
iv. Same number of circuits of particular length

(Geeksforgreeks)
Help from "Introduction to Graph Theory" by Douglas B. Wes and exercise answers

Now,
We have to prove that, if graph A and B have same numbers of nodes and are completely connected, they must be isomorphic.

Informal Proof: Since, two graphs A and B meet all the criteria above to be isomorphic, having equal number of vertices and edges, same degree sequence and same number of circuits of particular length, we can informally say that they are isomorphic.

Formal proof:
Given graph A and B, assume, they have n vertices, and each vertex in A is connected to another vertex as well as in B.
We want to show that there exists a one-to-one correspondence between vertices of A and B such that the adjacency relationship are preserved.

Now, we will prove this by:

Suppose u and v are adjacent vertices in A, then there exists edge (u,v) in A and since A is completely connected, every vertex in A is adjacent to every other vertex.
Hence, there is edge between u and v in A.

Now, for function f: V_1->V_2 , we have vertices f(u) and f(v) in B as well. Since, B is completely connected as well, there is also edge between f(u) and f(v) . Hence, f(u) and f(v) are adjacent in B.

Conversely, if f(u) and f(v) are adjacent vertices in B then there exist edge between f(u) and f(v) in B and hence, by definition of f, there is also edge between u and v in A.

Therefore, since we show that f preserves adjacency relationships between vertices, Hence, f is an isomorphic between A and B.

Hence, we can show that if two graph A and B have the same number of nodes and are completely connected, they must be isomorphic.

(c)

Prove that if two graphs A and B are isomorphic they do not have to be completely connected. (3 points)

We can prove this with just simple counter example.

Let have two isomorphic graph which are not completely connected.