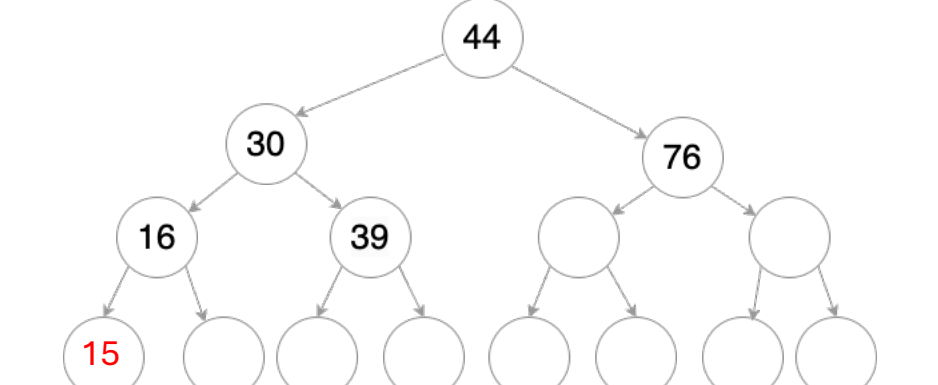
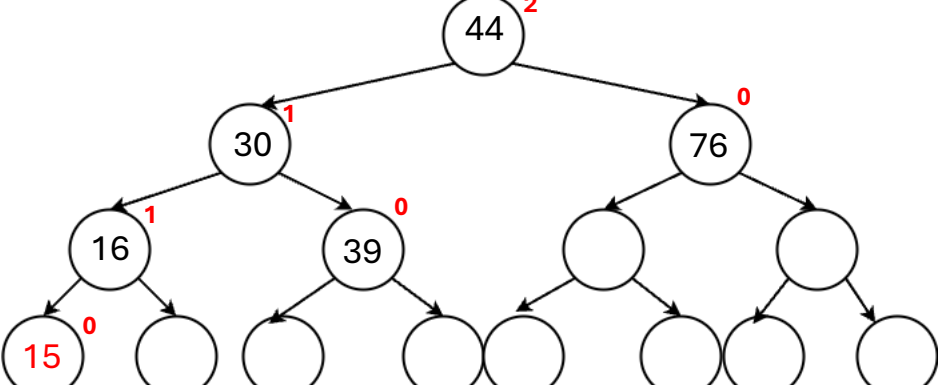
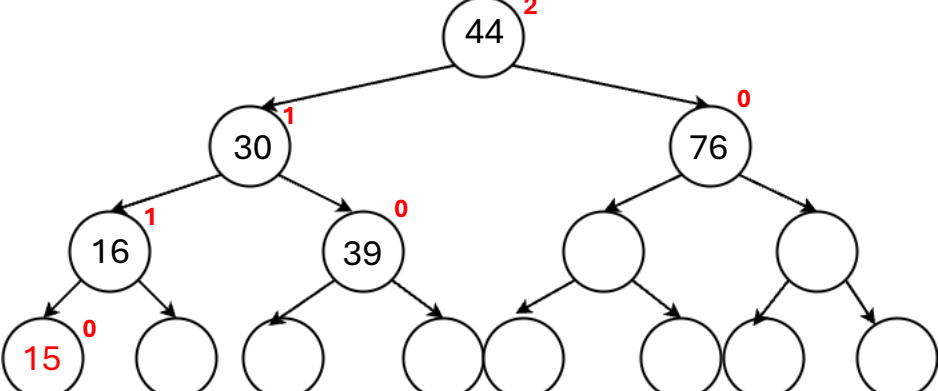
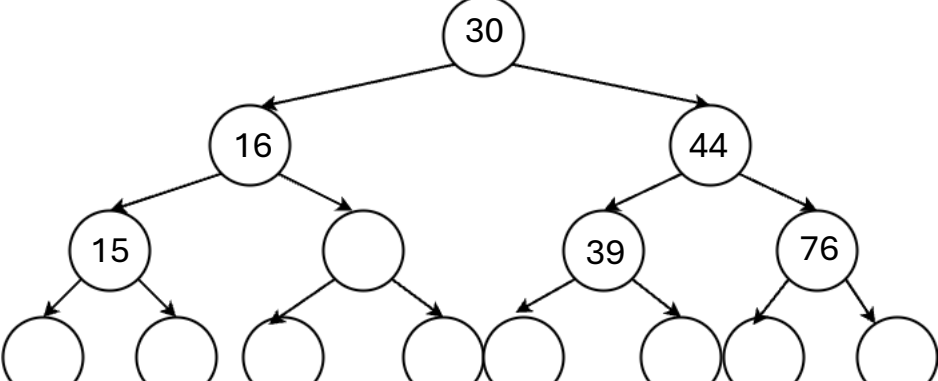


1) AVL Insert 15

Insert the value 15 into this AVL tree and apply any alterations necessary to maintain the rules of an AVL tree. Show your work.

<p>Step 1: Insert 15 just like a regular BST.</p>	
<p>Step 2: Calculate AVL (rotation) values.</p>	
<p>Step 3: Right Rotation needed at tree root.</p> <ul style="list-style-type: none">- 30 becomes SubRoot- 44 left takes over 30's right child (39)- 30 Takes 44 as new right child- 30 Becomes new root	
<p>Step 4: Done</p>	

1) AVL Insert 15

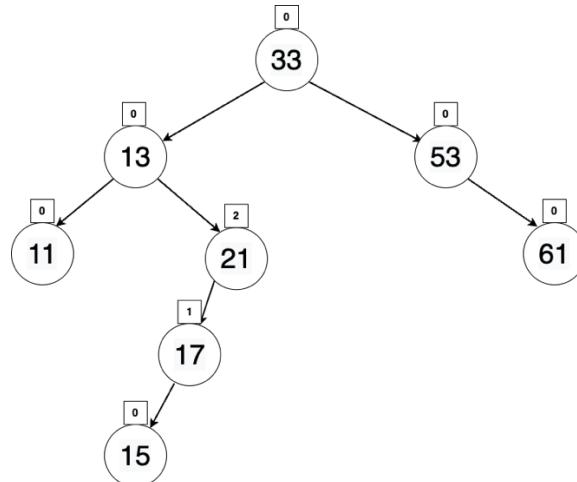
Step 1:

Insert 15 just like a regular BST.

Step 2:

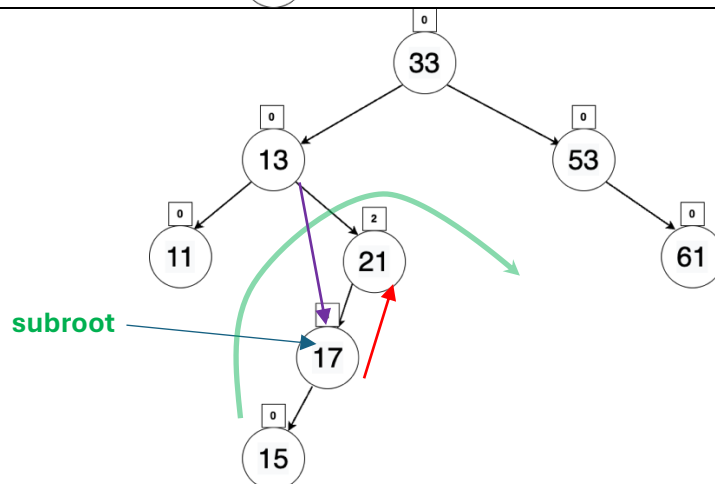
Calculate AVL values.

(I stopped calculating at the 21 node since a rotation was imminent and balance factors get recalculated after a rotation)



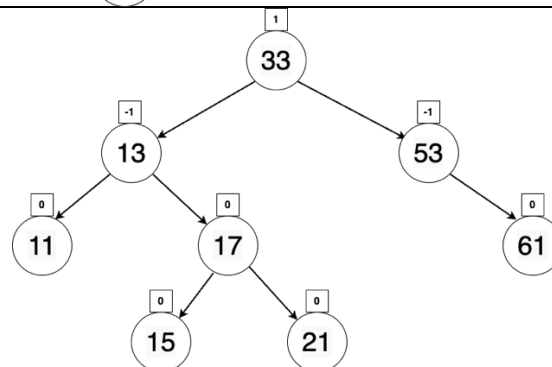
Step 3: Right Rotation

1. Identify subroot (17)
2. 17 takes 21 as right child
3. 13 Takes 17 as its new right child.
4. 21 would take over 17's right subtree, but it's null
5. Rotation complete



Step 4:

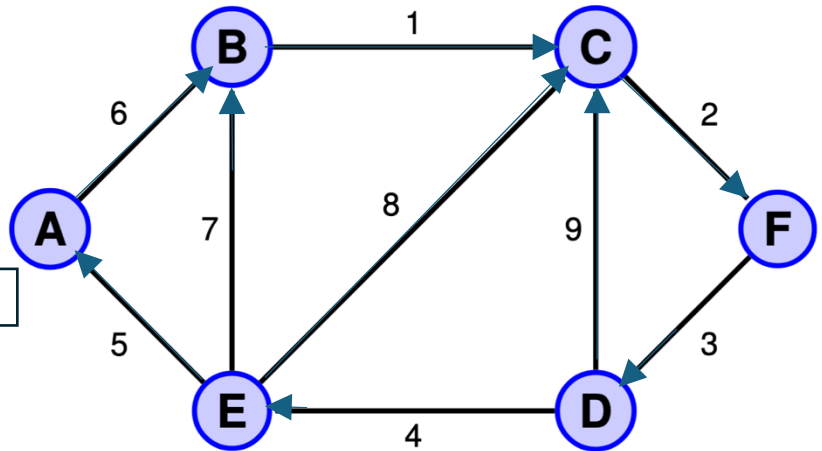
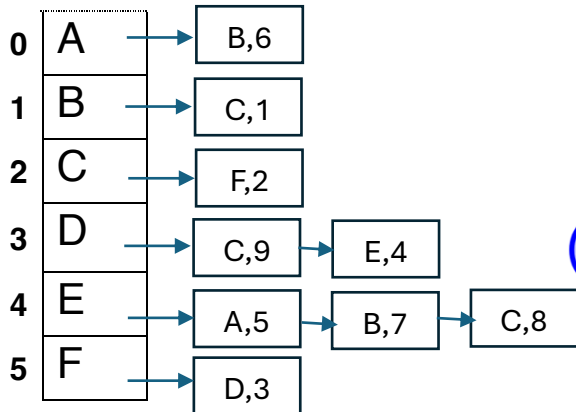
Recalculate balance factors and we can see all values within range $[-1, 0, 1]$



2) Graph Representations

Show a list based and matrix-based representation of the following graph.

List



Matrix

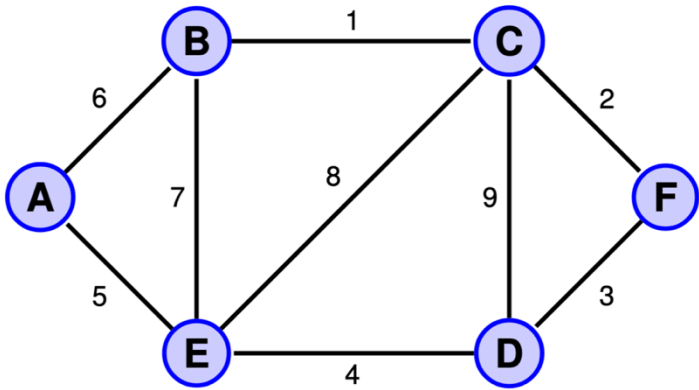
	0	1	2	3	4	5
0 A		6				
1 B			1			
2 C						2
3 D			9		4	
4 E	5	7	8			
5 F				3		

3) DFS and BFS

Perform a depth first search and a breadth first search using the graph on this page and placing the vertex values (A-F) in the spaces provided below as they become visited. Start with A using our alphabetical choices rule.

(only pushes nodes not already in stack)

Push A
Pop A
Push (A's neighbors) B, E
Pop E
Push (E's neighbors) C, D
Pop D
Push (D's neighbors) F
Pop F
Push (F's neighbors) Null



DFS

A	E	D	F	C	B
1	2	3	4	5	6

(only pushes nodes not already in queue)

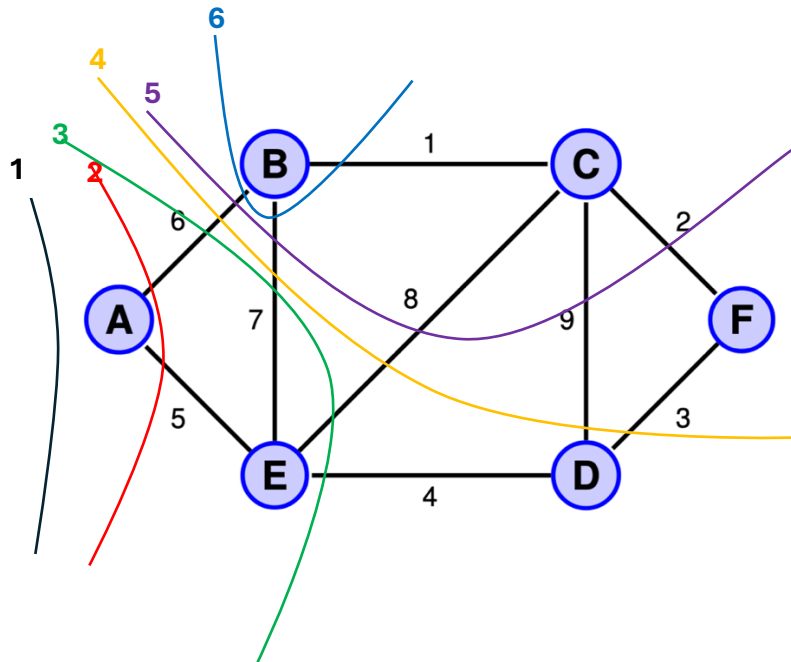
Push A
Pop A
Push (A's neighbors) B, E
Pop B
Push (B's neighbors) C
Pop E
Push (E's neighbors) D
Pop C
Push (C's neighbors) F
Pop D
Pop F

BFS

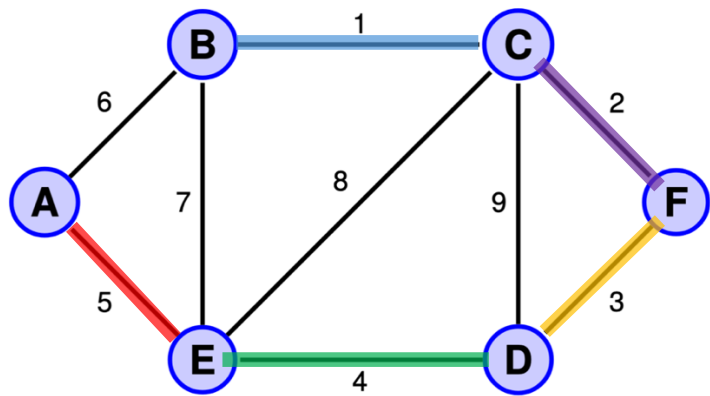
A	B	E	C	D	F
1	2	3	4	5	6

4) MST: Prims

Add edges to the list on the right as they are added to the MST. Graph edges are typically written in a similar fashion to this: **(Start Vertex, End Vertex, Edge Weight)**. Example: **D to F = (D, F,17)**. Remember when making choices (if it applies here) do them alphabetically.

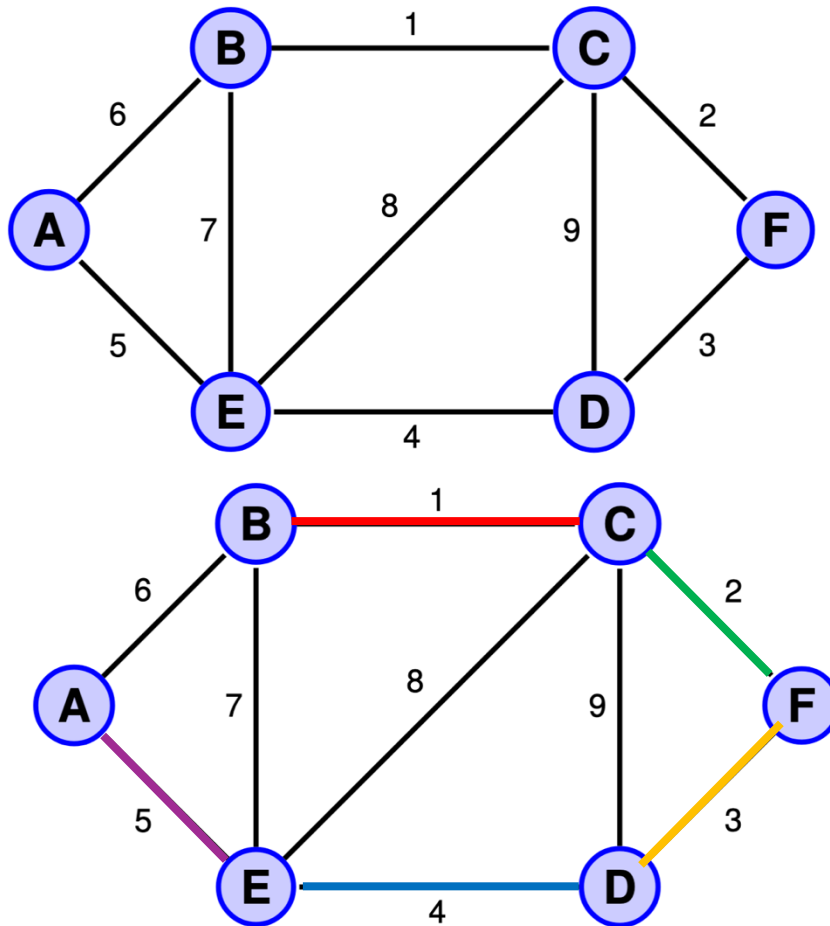


	CUT	Edges Crossed	MIN	MST
1	{ABCDEF}			A
2	{A} I {BCDEF}	(AB6) (AE5)	A,E,5	AE
3	{AE} I {BCDF}	(AB6) (BE7) (CE8) (DE4)	D,E,4	ADE
4	{ADE} I {BCF}	(AB6) (BE7) (CE8) (CD9) (DF3)	D,F,3	ADEF
5	{ADEF} I {BC}	(AB6) (BE7) (CE8) (CD9) (CF2)	C,F,2	ACDEF
6	{ACDEF} I {B}	(AB6) (BC1)	B,C,1	ABCDEF



5) MST: Kruskal's

Add edges to the list on the right as they are added to the MST. Refer to previous question for an example. Remember when making choices (if it applies here) do them alphabetically.

[illegible][illegible]

6) Shortest Path – Dijkstra's

Priority Queue
(AE-5)
(AB-6)
(BC-7)
(ED-9)
(EB-12)
(EC-13)

A	B	C	D	E	F
Visited Vertices					
A	E	B	C	D	

Previous	
A	
B	A
C	E B
D	E
E	A
F	C

Distance Table						
A	B	C	D	E	F	
∞	∞	∞	∞	∞	∞	
0	6	13	9	5	9	
		7				
						3

1. Start with A
2. Add AB, AE to queue
3. Update distance to B & E
4. Choose shortest distance (E)
5. Update distances from E
6. ...
7. ...
8. ...
9. ...
10. ...
11. ...
12. ...

