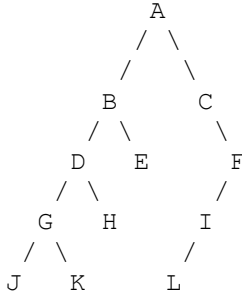


Be sure to answer each question carefully and completely.

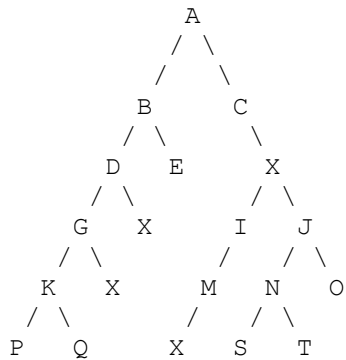


1. (12 pts.) For the tree sketched above, identify or answer each of the following:
 - a. type of tree
 - b. leave(s)
 - c. ancestor(s) of G
 - d. sibling(s) of D
 - e. level of H
 - f. subtree rooted at G

2. (8 pts.) For the tree sketched above, give the output if its nodes were visited and printed in:
 - a. a *preorder traversal*

 - b. a *in-order traversal*

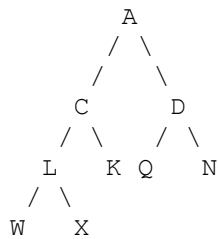
3. (8 pts.) Give the path from A to X for the following tree:



a) when doing a *depth-first* search

b) when doing a *breadth-first* search

4. (12 pts.) In the space below, answer the questions that follow regarding this *binary heap tree* organized using an alphabetical ordering:



a) Give the internal Python representation used in your text.

b) Draw the final tree after B is added.

c) Draw the final tree after A is deleted.

5. (4. pts.) Which data structure, a *stack*, *queue*, or *deque*, would be the most appropriate for each of the following?

a) Implementing a *breadth-first* search of a graph

b) Implementing a *depth-first* search of a graph

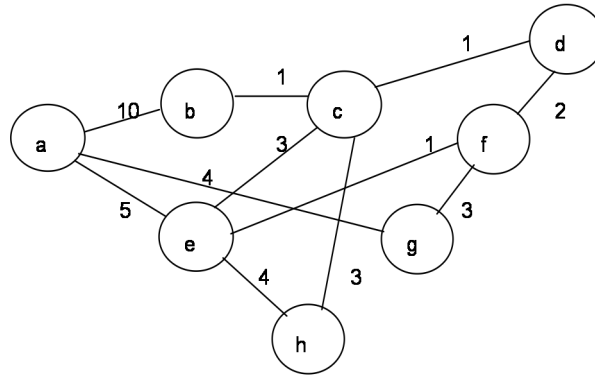
6. (12 pts.) Given the following *adjacency matrix* for a graph,

	v ₀	v ₁	v ₂	v ₃	v ₄	v ₅
v ₀		5		6		2
v ₁	10		4			12
v ₂				9		
v ₃					7	3
v ₄	1					
v ₅			11		8	

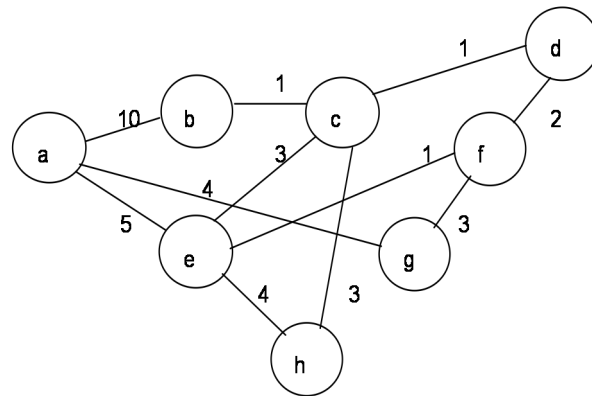
a) sketch the graph

b) give the corresponding *adjacency list* representation as box-pointer diagram (ignoring weights)

7. (8 pts.) Starting at Node *a*, use *Dijkstra's Algorithm* to calculate the minimal cost for visiting each node. Write that cost on the node. Below the graph, list the nodes in the order that you expanded them.



8. (8 pts.) Starting at Node *a*, use *Prim's Algorithm* to find a minimal spanning tree. Below the graph, list the edges you added to the spanning tree in the order that you added them. (List edges as ordered pairs of nodes. For example, (a, b) is the edges with weight 10 in the graph.)



9. (8 pts.) Give the parse tree for the expression $2 / 3 + 4 * 5 - 6$

10. (20 pts.) Using a list representation, implement in Python the following functions from an ADT for a *binary search tree*:
- a) `binaryTree()` creates a new binary tree.
 - b) `getLeftChild()` returns the binary tree that is the left sub-tree of the current node.
 - c) `setRootValue(val)` stores `val` in the root of the current node.
 - d) `insertLeft(val)` creates a new binary tree and inserts it as the left child of the current node.

Pledged: _____