DSA FINAL 2018 – 2019

1. Binary tree

Given a list of items: 45, 30, 80, 20, 35, 70, 90, 25, 21, 39, 37, 38, 75, 78, 76, 90

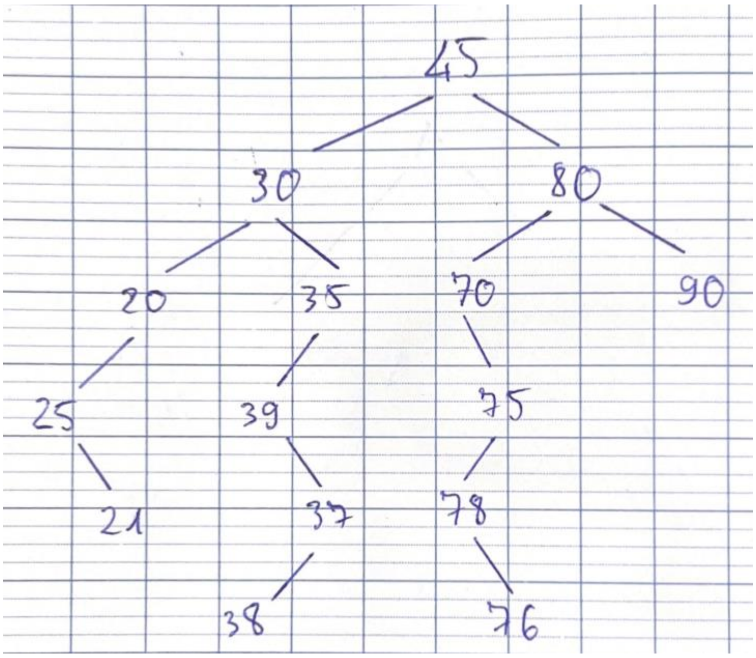
a. Insert all items one by one from left to right of the list into a binary search tree and draw the tree.

b. Delete item 20 from the tree and redraw the tree

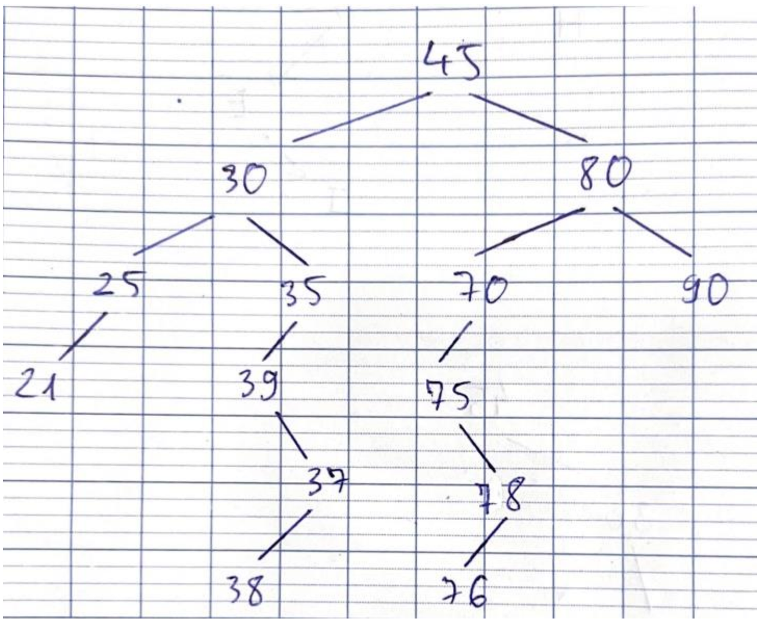
c. Delete item 45 from the tree and redraw the tree

Answer:

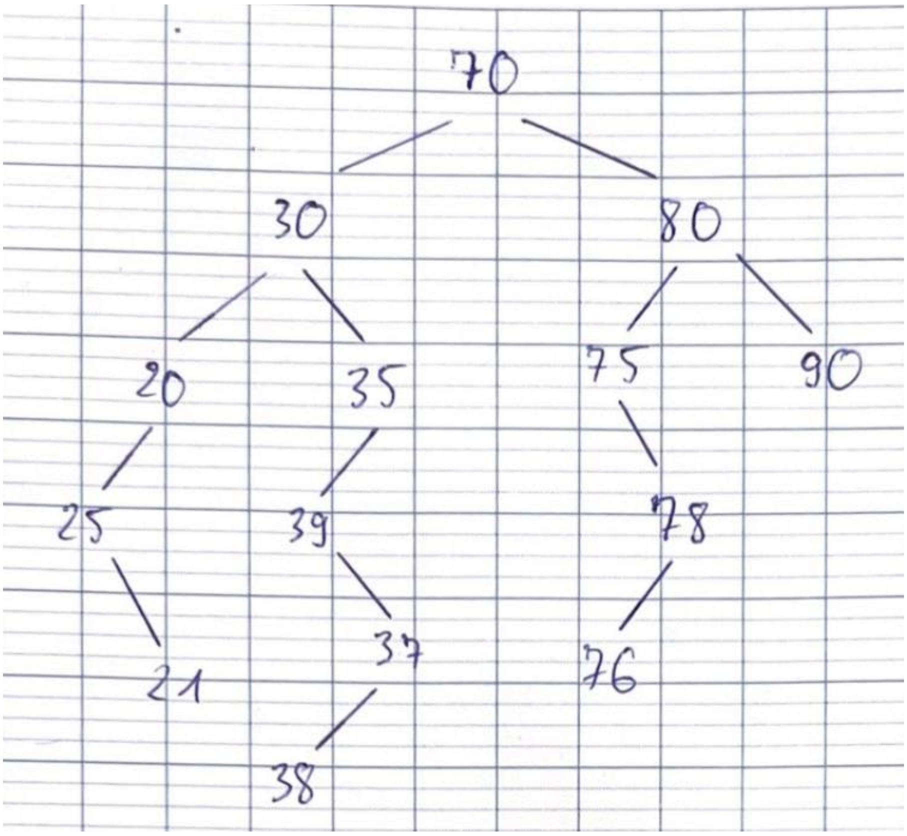
a.



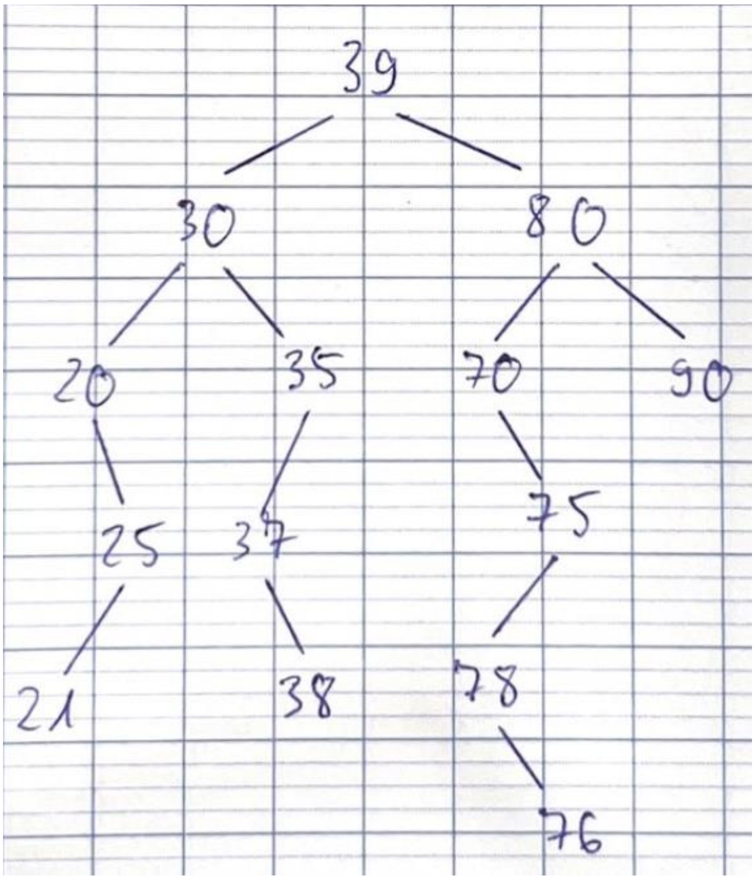
b.



c. Method 1: Succesor of right sub-tree → That is 70, the smallest value in the right sub-tree



Method 2: Succesor of left sub-tree → That is 39, the largest value in the left sub-tree.



2. Hash table

Given a hash table of size 11.

a. Assume that the linear probing algorithms is used to solve collision.

Insert into the hash table the following items: 3, 4, 22, 24, 6, 25, and draw the hash table.

b. Change the hash table's size to 15, redraw it.

<u>Answer:</u>

a. Hash function: Index = Value % 11

Key	0	1	2	3	4	5	6	7	8	9
Value	22		24	3	4	25	6			

*Note: 25 collides at 3 → Move to 4

25 collides at 5 → Move to 5

b. Hash function: Index = Value % 15

Key	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Value				3	4		6	22		24	25				

3. Graph – Elementary Algorithms

Given a graph G represented by the following list of succesors.

For each pair (x, y), x is the weight of the edge, y is the terminal extremity of the edge.

A: (5, D) → (2, E)
B: (4, E) → (2, D) → (2, F)
C: (3, F)
D: (2, G) → (1, F)
E: (1, G)
F: (3, H)
G: (3, H)
H: /

Fig.1: List of successors of G

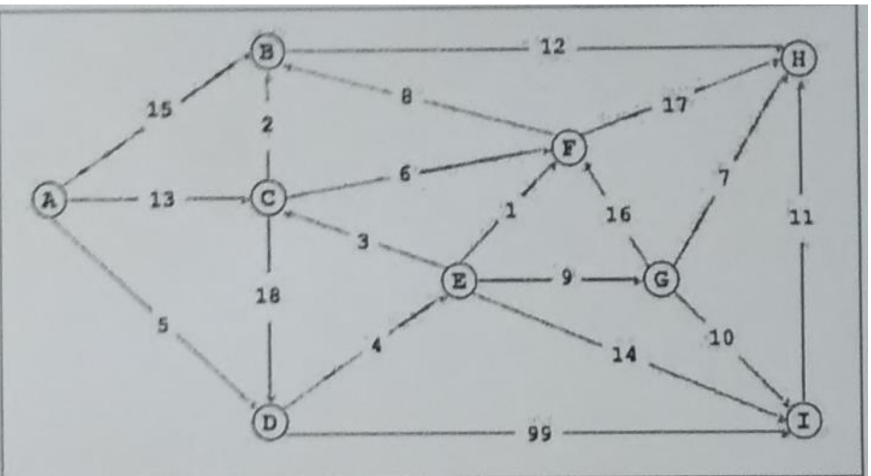


Fig. 2: Graph Gs

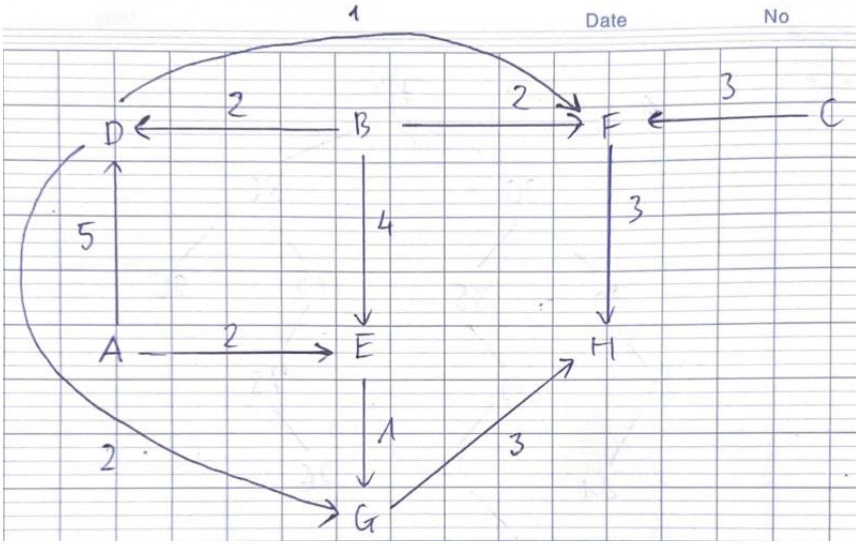
a. Draw the graph.

b. Show an adjacency matrix of the graph.

c. Topological sort the graph G from node A, and write a topological order of all nodes.

***<u>Answer</u>*:**

a.



b.

	A	B	C	D	E	F	G	H
A	0	0	0	5	2	0	0	0
B	0	0	0	2	4	2	0	0
C	0	0	0	0	0	3	0	0
D	0	0	0	0	0	1	2	0
E	0	0	0	0	0	0	1	0
F	0	0	0	0	0	0	0	3
G	0	0	0	0	0	0	0	3
H	0	0	0	0	0	0	0	0

c. We perform Depth-First Search or graph in question a.

→ Using pencil, we got: Discovered time of each node, and finishing time

→ Key point is that, each time you write down finishing time of each node, insert first that node to your answer list

* Remember to follow alphabetical order (A, B, C, D, E, …)

→ Answer list: C B A E D G F H (f = 16, 14, 12, 11, 9, 8, 6, 5)

* By observation, it is simply that topological sort actually is a list of nodes, which is the result of DFS algorithm, that has the finishing time biggest to smallest value!

* Note: The graph must be acyclic (no circle – no loop), to perform topological sort.

4. Graph – Shortest path algorithm

Run the Dijkstra's algorithm on the graph G_S in Fig. 2 <u>starting from node C</u>, and fill the following table with corresponding values after each step of the algorithm

Selected nodes	A	B	C	D	E	F	G	H	I
-	∞	∞	<u>0</u>	∞	∞	∞	∞	∞	∞
C	∞	<u>(2, C)</u>		(18, C)	∞	(6, C)	∞	∞	∞
B	∞			(18, C)	∞	<u>(6, C)</u>	∞	(14, B)	∞
F	∞			(18, C)	∞		∞	<u>(14, B)</u>	∞
H	∞			<u>(18, C)</u>	∞		∞		∞
D	∞				<u>(22, D)</u>		∞		(117, D)
E	∞						<u>(31, E)</u>		(36, E)
G	∞								<u>(36, E)</u>

5. Rooted trees with unbounded branching

Given a tree whose node may have arbitrary numbers of children. There is a schema to represent that kind of tree name left-child, right-sibling. Each node x contains a parent pointer p, and two other pointers:

- left-child[x] points to the leftmost child of node x, and

- right-sibling[x] points to the sibling of x immediately to the right.

If node x has no children, the left-child[x] = NULL, and if node x is the rightmost child of its parent, then right-sibling[x] = NULL.

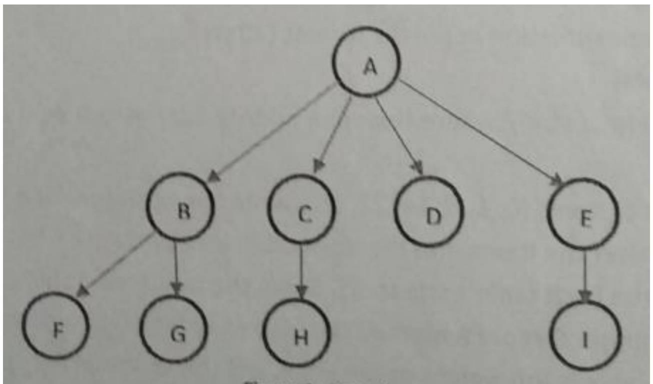


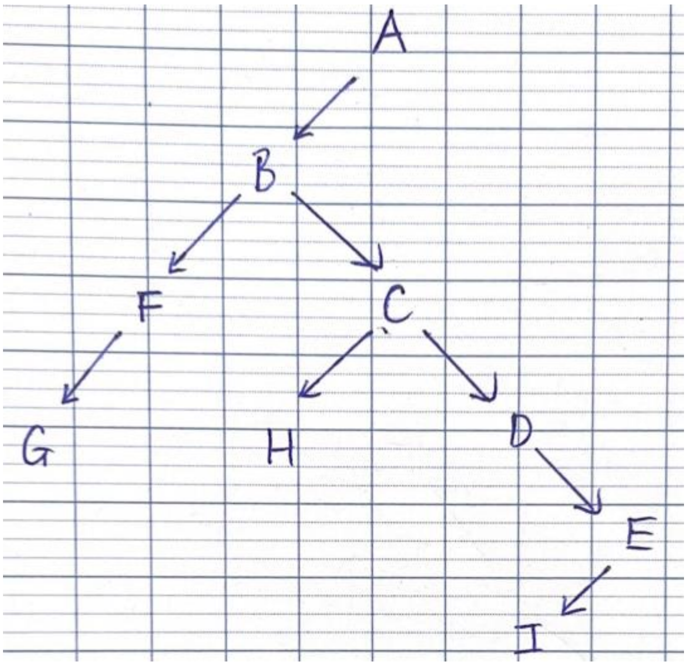
Figure 1 - Tree T

i. Draw a left-child, right-sibling representation of the tree T in figure 1.

ii. Write an O(n) non-recursive procedure that prints all the keys of an arbitrary rooted tree with n nodes, where the tree is stored using the left-child, right-sibling representation.

<u>Answer:</u>

i.



ii. Pseudocode:

```
Traverse(node):
      Stack container = new Stack();
      // or Queue container = new Queue();

        while (!container.isEmpty()) {
            // Push left, move right
            container.push(node.left);
            print(node.value);
            node = node.right;
            // Or push right, move left
            container.push(node.right);
            print(node.value);
            node = node.left;

            // The node is leaf
            if (node.left == node.right == null) {
                  node = container.pop();
                  // Or node = container.dequeue();
            }
        } // End while loop
// End algorithm
```

Sample code:

```
class Node {
    int key;
    Node parent;
    Node leftChild;
    Node rightSibling;

    Node(int key) {
        this.key = key;
        this.parent = null;
        this.leftChild = null;
        this.rightSibling = null;
    }
}

public class TreeTraversal {
```

```
    public static void printTreeKeys(Node root) {
        if (root == null) {
            return;
        }

        Node current = root;

        while (current != null) {
            System.out.println(current.key); // Print the key of the current node

            if (current.leftChild != null) {
                current = current.leftChild;
            } else if (current.rightSibling != null) {
                current = current.rightSibling;
            } else {
                // Move up the tree until finding a node with a right sibling or
reaching the root
                while (current.parent != null && current.rightSibling == null) {
                    current = current.parent;
                }
                if (current.parent == null) {
                    break; // Reached the root and finished traversal
                }
                current = current.rightSibling;
            }
        }
    }

    public static void main(String[] args) {
        // Create the tree structure
        Node root = new Node(1);
        Node node2 = new Node(2);
        Node node3 = new Node(3);
        Node node4 = new Node(4);
        Node node5 = new Node(5);
        Node node6 = new Node(6);
        Node node7 = new Node(7);

        root.leftChild = node2;
        node2.parent = root;
        node2.rightSibling = node3;
        node3.parent = root;
        node3.rightSibling = node4;
        node4.parent = root;
        node4.leftChild = node5;
        node5.parent = node4;
        node5.rightSibling = node6;
        node6.parent = node4;
        node6.rightSibling = node7;
        node7.parent = node4;

        // Call the printTreeKeys method
        printTreeKeys(root);
    }
}
```