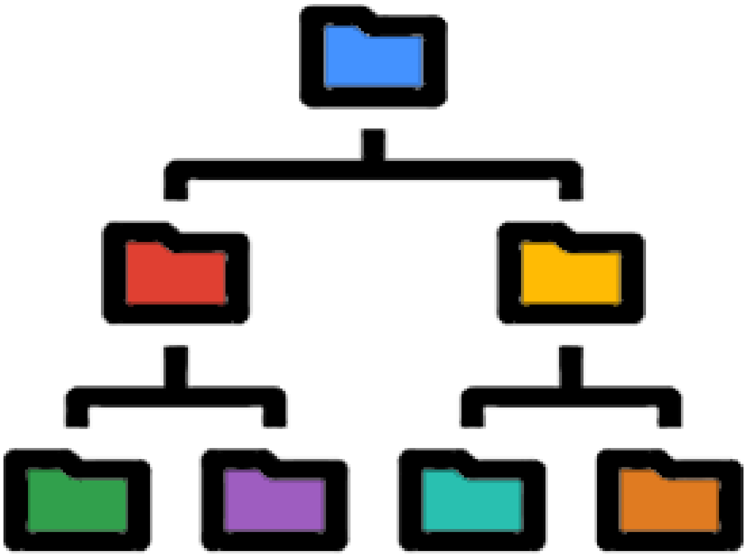
COMP9123

Data structures And Algorithms

Assignment 2



Student id:- 510390888

Name:- Nehal Patel

Each node represents a table at Restaurant Ruckus,

We can considered the root of the tree as the door of the restaurant

1. Check if the table is full

```
def is_table_full(node):
    if node.value ==7:
        return True
```

2. Check if the table is currently elves-only

```
def is_elves_only(node):
    for each i in node:
        if i is not elve:
            return False
    return True
```

3. Check if the table is currently dwarves-, hobbit-, or elves-only

```
def is_dhe_only(node):
    array = [dwarves, hobbit, elves]
    for each i in node:
        if i not in array[]:
            return False
    return True
```

4. Check the number of people currently at the table

```
def get_total_diners(node):
    return node.data
```

5. Check the number of elves currently at the table

```
def get_elves():
    int count_elve=0
    for i in node:
        if i ==elve:
            count <= count +1
    return count
```

6. Add an Elf, a Human, a Dwarf, or a Hobbit to a table (if the table is full, should return null):

```
def add_elf():
    if node.data == 7:
        return null
    int dist = -1;
    if ((root->data == x) || (dist = getDistance(root->left, x)) >= 0 || (dist =
    getDistance(root->right, x)) >= 0):
        return dist + 1;
    else if node.dist < dist:
        if is_elves_only(node):
            node.data = node.data +1
            return “Elve Added”
        else:
            return “Cannot be added”
    else:
        return “Cannot be added”
//////////////////////////////////////////////////////////////  NEXT  /////////////////////////////////////////////////////////////////
def is_hobbit_only(node):
    for each i in node:
        if i is not hobbit:
            return False
    return True

def is_dwarf_only(node):
    for each i in node:
        if i is not dwarf:
            return False
    return True

def add_human(u):
    if node.data ==7:
        create new node (# You need to call getNode)
    else:
        if not (node.is_dwarf_only or node.is_elves_only or node.is_hobbit_only):
            node.data = node.data+1
            return “Human added”
```

```
def add_dwarf():
    if current_node.data==7:
        create new node (# You need to call getNode)
    else:
        if node.get_elves<3:
            node.data = node.data +1
            return “Dwarf added”


def add_hobbit():
    if current_bode.data == 7:
        create new node (# You need to call getNode)
    else:
        temp = node
        for nodes in tree:
            if temp.data>node.data:
                temp=node.data
        temp.data = temp.data +1
        return “Hobbit added”
```

7. Check the distance to the entrance of the inn

```
def  get_distance(Node *root, int n):
        if (root == NULL)
                return -1;
        int dist = -1;
        if ((root->data == n) || (dist = get_distance(root->left, n)) >= 0 || (dist = get_distance(root->right, n)) >= 0):
                return dist + 1;
        return dist;
```

8. Start a new table (originally empty) at the farthest position from the door

```
class Node {
   public:
      int data
};
Node* getNode(int data) {
   Node* node = new Node;
   node =>data = data;
   return node;
}
Node* root = getNode(1);
```

9. Retrieve the least crowded table

```
def get_least_crowded_table(node):
    if node.data=null
        return
    int least = 0
    data = node.data
    if data< node.next:
        get_least_crowded_table(node.next)
    return data
```

10. Retrieve the current number of tables

```
def left_height(node):
    ht = 0
    while(node):
        ht += 1
        node = node.left
    return ht

def right_height(node):
    ht = 0
    while(node):
        ht += 1
        node = node.right
    return ht
def get_number_tables(root)
    if(root == None):
        return 0
    heightofleft = left_height(root)
    heightofright = right_height(root)
        return 2^height(1<<height) -1
    if(heightofleft == heightofright):
        return (1 << heightofleft) - 1
    return 1 + get_number_tables (root.left) + get_number_tables (root.right)
```

11. Retrieve the number of customers

```
def get_number_diners(root):
int diners = 0
if node.next = none
    return diners+node.data
if root.left:
    diners = diners + root.left.data
    get_number_diners(root.left.next)
else:
    diners = diners + root.right.data
    get_number_diners(root.right.next)
return diners
```

12. Retrieve the number of Elves, Dwarves, Humans, and Hobbits

```
def get_number_elves(root):
if node.data == none:
    return
// Elves only sit with elves, so if only evles condition fulfilled then easy to count
int elves =0
if root.left:
    if root.left.is_elves_only():
        elves = elves + root.left.data
        return get_number_elves(root.left.next)
else:
    if root.right.is_elves_only():
        elves = elves + root.right.data


def get_number_dwarves(root):
int dwarf = 0
if node.data == none:
    return
if root.left:
    for i in root.left:
        if i == dwarf:
```

```
                                                dwarf = dwarf +1
                        return get_number_dwarves(root.left.next)
            else:
                         for i in root.right:
                                    if i == dwarf:
                                                dwarf = dwarf +1
                        return get_number_dwarves(root.right.next)


            def get_number_humans()
            int humans = 0
            if node.data == none:
                        return
            if root.left:
                        for i in root.left:
                                    if i == human:
                                                humans = humans +1
                        return get_number_humans(root.left.next)
            else:
                         for i in root.right:
                                    if i == human:
                                                humans = humans+1
                        return get_number_humans(root.right.next)
```

```
def get_number_hobbits()
int hobbits = 0
if node.data == none:
    return
if root.left:
    for i in root.left:
        if i == hobbit:
            hobbits = hobbits +1
    return get_number_hobbits(root.left.next)
else:
     for i in root.right:
        if i == hobbit:
            hobbits = hobbits +1
    return get_number_ hobbits (root.right.next)
```