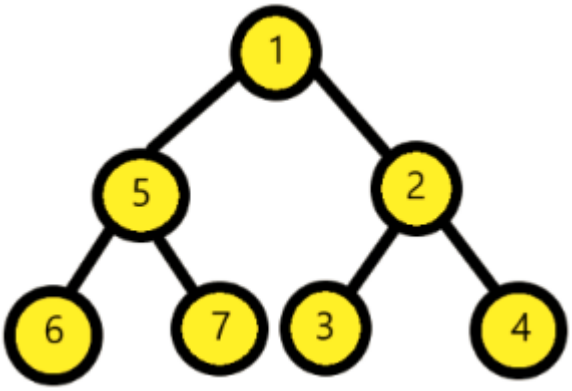
Competitive Programming Lab - 7

Academic year: 2020-2021 **Semester:** Long Sem

Faculty Name: Dr. Ajith Jublison sir **Date:** 7/ 7/ 2022

Student name: Taran Mamidala **Reg. no.:** 19BCE7346

Zigzag traversal :



Q1.) Print the zigzag traversal.

CODE:

```
import java.util.*;
import java.io.*;
public class ZigZagSTraversal {
      static class Node {
      int data;
      Node left, right, newtree;

      Node(int key) {
            data = key;
            left = right = newtree = null;
      }
      }

      static void sZigzagTraversal(Node root) {
      if (root == null)
            return;
```

```
Stack < Node > giventree = new Stack < > ();
Stack < Node > newtree = new Stack < > ();
giventree.push(root);

boolean LorR = true;

while (!giventree.isEmpty()) {
      Node curr = giventree.pop();
      if (curr != null) {
            System.out.print(curr.data + " ");
            if (LorR) {
            if (curr.right != null)
                  newtree.push(curr.right);

            if (curr.left != null)
                  newtree.push(curr.left);
            } else {
            if (curr.left != null)
                  newtree.push(curr.left);

            if (curr.right != null)
                  newtree.push(curr.right);
            }

            if (giventree.isEmpty()) {
            Stack < Node > temp = giventree;
            giventree = newtree;
            newtree = temp;

            LorR = !LorR;
            }
      }
}
}
static void inorder(Node root) {
if (root == null)
      return;

inorder(root.left);
System.out.print(root.data + " ");
inorder(root.right);
}
```

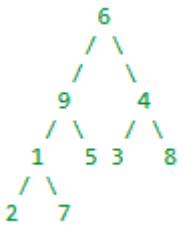
```
        public static void main(String[] args) {
        Node root = new Node(1);
        root.left = new Node(5);
        root.right = new Node(2);
        root.left.left = new Node(6);
        root.left.right = new Node(7);
        root.right.right = new Node(4);
        root.right.left = new Node(3);

        sZigzagTraversal(root);
        }
}
```

Output:

Result
CPU Time: 0.11 sec(s),

```
1 5 2 4 3 7 6
```



Result
CPU Time: 0.11 sec(s), Memo

```
6 9 4 8 3 5 1 2 7
```

Result
CPU Time: 0.13 sec(s), Memory

```
7 6 3 8 4 5 2 9 1
```

Trie :

Q2.) In our problem, we will have to implement a dictionary with the help of a trie so that for a given string word input, the program prints its meaning from a prebuilt dictionary

Example Input
Input 1
Program
Input 2:

Computer

Example Output

Output 1

Programming is the process of creating a set of instructions that tell a computer how to perform a task.

Output 2

A computer is a machine that can be programmed to carry out sequences of arithmetic or logical operations automatically

CODE:

```
class Trie:
      def __init__(self):
      self.isWordsEnd = False
      self.listMap = dict()
      self.wordMeaning = ""
def createNewTrieNode():
      node = Trie()
      node.isWordsEnd = False
      return node

def insertInTheTrie(root, givenStr, meaning):
      if root is None:
      root = createNewTrieNode()
      temp = root
      for ch in givenStr:
      if temp.listMap.get(ch) is None:
            temp.listMap[ch] = createNewTrieNode()
      temp = temp.listMap.get(ch)
      temp.isWordsEnd = True
      temp.wordMeaning = meaning
      return root
def getWordMeaning(root, word):
      if root is None:
      return "Found no meaning as word not present as root is None"
      temp = root
      for ch in word:
      temp = temp.listMap.get(ch)
      if temp is None:
            return "Found no meaning as word not present as char " + ch + " is
None"
      if temp.isWordsEnd == True:
```

```
        return temp.wordMeaning
        return "Found no meaning as word not present!"
if __name__ == "__main__":
        root = None
        root = insertInTheTrie(root, "computer", '''A computer is a machine that
can be instructed to carry out sequences of arithmetic or logical operations
automatically via computer programming''')
        root = insertInTheTrie(root, "map", "a diagrammatic representation of an
area")
        root = insertInTheTrie(root, "programming", "Programming is the process
of creating a set of instructions that tell a computer how to perform a task")
        root = insertInTheTrie(root, "language", "the method of human
communication")
        root = insertInTheTrie(root, "book", '''a written or printed work
consisting of pages glued or sewn together along one side and bound in
covers.''')
        root = insertInTheTrie(root, "science", '''the intellectual and practical
activity encompassing the systematic study of the structure and behaviour of
the physical and natural world through observation and experiment.''')
        givenStr = "programming"
        print(getWordMeaning(root, givenStr))
```

Output:

Result

executed in 0.856 sec(s)

```
Programming is the process of creating a set of instructions that tell a computer how to perform a task
```

```
givenStr = “book"
```

Result

executed in 0.869 sec(s)

```
a written or printed work consisting of pages glued or sewn together along one side and bound in covers.
```

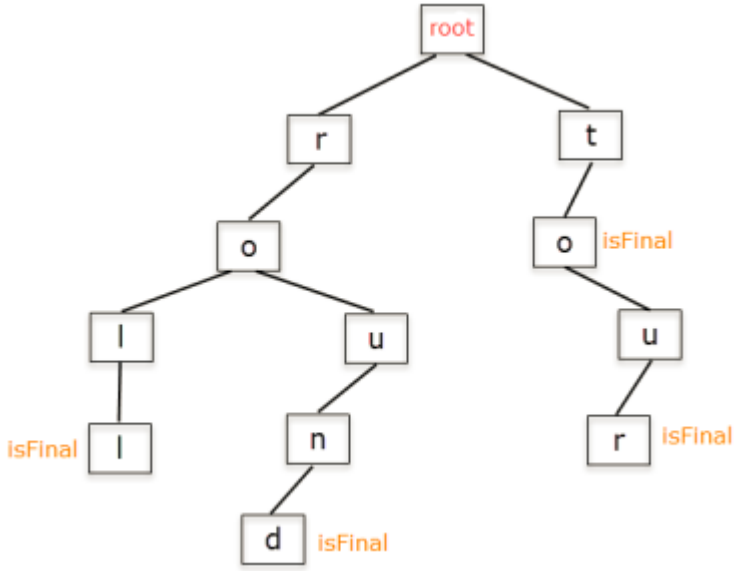
```
        givenStr = "computer"
```

Result

executed in 0.84 sec(s)

```
A computer is a machine that can be instructed to carry out sequences of arithmetic or logical operations automatically via computer programming
```

Q3.) Given a Binary Search Tree. Convert a given BST into a Special Max Heap with the condition that all the values in the left subtree of a node should be less than all the values in the right subtree of the node. This condition is applied on all the nodes in the converted Max Heap.



Example Input

Input 1

```
        4
      /   \
    2       6
  /   \   /   \
1     3 5     7
```

Example Output

Output

```
          7
        /   \
      /       \
    3           6
  /   \   /   \
1     2 4     5

1 2 3 4 5 6 7
```

CODE :

```
import java.util.*;

public class BSTtoMaxHeap {
```

```
static int i;
static class Node {
int data;
Node left, right;
Node(int key) {
      data = key;
      left = right = null;
}
};

static void getInorderOrder(Node root, Vector < Integer > store) {
if (root == null)
      return;
getInorderOrder(root.left, store);
store.add(root.data);
getInorderOrder(root.right, store);
}

static void BSTToMaxHeap(Node root, Vector < Integer > store) {
if (root == null)
      return;

BSTToMaxHeap(root.left, store);
BSTToMaxHeap(root.right, store);
root.data = store.get(i++);
}

static void maxHeapconversion(Node root) {
Vector < Integer > store = new Vector < Integer >();
int i = -1;

getInorderOrder(root, store);
BSTToMaxHeap(root, store);
}

static void getPostOrder(Node root) {
if (root == null)
      return;

getPostOrder(root.left);
getPostOrder(root.right);

System.out.print(root.data + " ");
}
```

```
        public static void main(String[] args) {

        Node root = new Node(4);
        root.left = new Node(2);
        root.right = new Node(6);
        root.left.left = new Node(1);
        root.left.right = new Node(3);
        root.right.right = new Node(7);
        root.right.left = new Node(5);

        maxHeapconversion(root);
        System.out.print("Special Max Heap of Tree is :\n");
        getPostOrder(root);

        }
}
```

Output:

Result

CPU Time: 0.11 sec(s), Memory: 33492 kilobyte(s)

```
Special Max Heap of Tree is :
1 2 3 4 5 6 7
```