Whiteboard 17

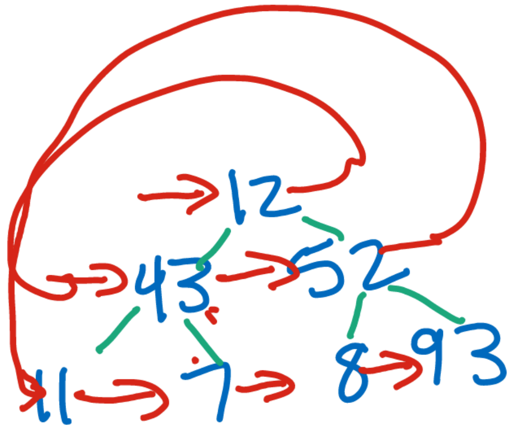
WHITEBOARD 17

Write a breadth first (also known as level-first) Binary Tree method to print every node visited as the tree is traversed.

no built in methods

Visual

Sample Binary Tree

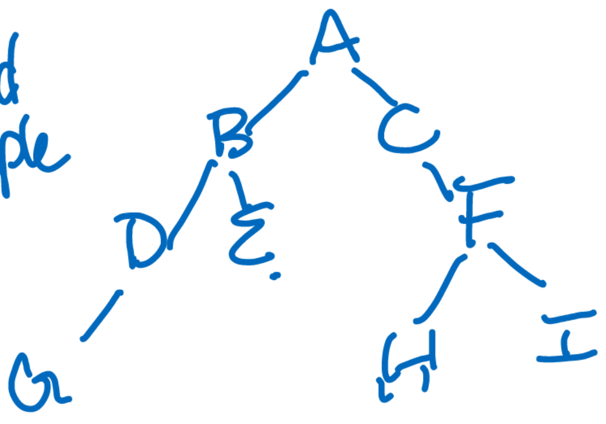


visit path

so output would be
12 43 52 11 7 8 93

Second example

Input tree



Perclass
instantiate an empty Queue & enque/deque to visit each node breadth first

A

temp = A (print)

B C

temp = B (print)

$\boxed{CDE}$

temp = C (print)

$\boxed{DEF}$

temp = D (print)

$\boxed{EFG}$

temp = E (print)

$\boxed{FGHI}$

temp=F

temp=G

temp=H

temp=I

temp = null

ea. step you visit all children of ea node left to right

Algorithm

- initialize a queue w/ a root
- while queue not empty

```
loop [
    • poll node from q
    • add children to back
          of q
    • print node
]
```

Big O

O of n time because while loop

O of 1 space

because data isn't duplicated

Pseudo

- Build enque & deque methods onto custom Queue class
- Open method that takes in a single node
  - instantiate an empty Queue
  - use custom enque to put in 1st node @ root
    make root a temp value

1

```
    print temp value.

    enquie left & right
    of temp value
    dequeue temp value
    Set temp to left
    val in que
    enque left right
    of temp
    print temp
    deque temp value
repeat till temp null
```

①

```
class MyQueue {
    private int MaxSize;
       "     "[] queArray;
       "     int front;
       "     int back;
    private int elements;
  public MyQueue(int e)
      max size = e
      queArray = new int[max]
       front=0; rear=-1; elent=0;
  }
public void enque(Node
node) {
    if (rear == maxsize-1)
       rear =-1;
     QueArray[++rear = node
       elements++;
   }
```

```
public int deque() {
    int temp = queArray
        [front++];
    if(front == maxSize){
        front = 0;
        elements--;
sout temp →   return temp;
}
public Boolean isEmpty(){
    return (elements==0);
}
public int size() {
    return elements;
}
```

2

```
class TreeNode {
        int data;
        Node left;
        Node right;
       public Node(int element)
     {  data = element
        left = right = null;
```

3

```
Class BinaryTree {
     Node root;
  public BinaryTree(){
         r oot = null;

   }
```

```
// actually need to know
level before printing
nodes

public void printLevels(){
    int h= height(root);
    int i;
    for(i ; i<=h ; i++){
    printGivenLevel(root, i);

}
private int height (Node root) {
    if(root ==null) {
        return 0;
    } else
```

```
int lefth = height(root.left);
int righth =   "   (root.right);

if ( lefth  > righth  ) {
     return(lefth +1);
    else
```

// did this wrong

going to code

tests: see visuals