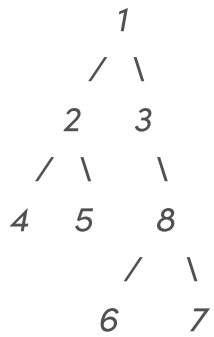
Maximum width of Binary Tree

Given a binary tree, the task is to find the maximum width of the given tree. Width of a tree is maximum of widths of all levels.

Example:

> ***Input:***
>
> 
>
>
> ***Output:*** *3*
> ***Explanation:*** *For the above tree,*
> *width of level 1 is 1,*
> *width of level 2 is 2,*
> *width of level 3 is 3*
> *width of level 4 is 2.*
> *So the maximum width of the tree is 3.*

Level Order Traversal using Queue

When a queue is used, we can count all the nodes in a level in constant time. This reduces the complexity to be a linear one.

In this method do the following:

- **Store all the child nodes** at the current level in the queue.
  - Count the **total number of nodes** after the level order traversal for a particular level is completed.
  - Since the queue now contains all the nodes of the next level, we can easily find out the total number of nodes in the next level by finding the size of the queue.
- Follow the same procedure for the successive levels.
- Store and **update the maximum** number of nodes found at each level.

Below is the implementation of the above approach.

C++ Java

```
// Java program to calculate width of binary tree

/* A binary tree node has data, pointer to left child
and a pointer to right child */
class Node {
    int data;
    Node left, right;

    Node(int item)
    {
        data = item;
        left = right = null;
    }
}
```

```
class BinaryTree {
    Node root;

    /* Function to get the maximum width of a binary tree*/
    int getMaxWidth(Node node)
    {
        int maxWidth = 0;
        int width;
        int h = height(node);
        int i;

        /* Get width of each level and compare
        the width with maximum width so far */
        for (i = 1; i <= h; i++) {
            width = getWidth(node, i);
            if (width > maxWidth)
                maxWidth = width;
        }

        return maxWidth;
    }

    /* Get width of a given level */
    int getWidth(Node node, int level)
    {
        if (node == null)
            return 0;
```

```
    if (level == 1)
        return 1;
    else if (level > 1)
        return getWidth(node.left, level - 1)
            + getWidth(node.right, level - 1);
    return 0;
}

/* UTILITY FUNCTIONS */

/* Compute the "height" of a tree -- the number of
nodes along the longest path from the root node
down to the farthest leaf node.*/
int height(Node node)
{
    if (node == null)
        return 0;
    else {
        /* compute the height of each subtree */
        int lHeight = height(node.left);
        int rHeight = height(node.right);

        /* use the larger one */
        return (lHeight > rHeight) ? (lHeight + 1)
                                   : (rHeight + 1);
    }
}

/* Driver code */
```

```
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();

        /*
        Constructed binary tree is:
             1
            / \
           2   3
          / \   \
```



```
        */
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.root.right.right = new Node(8);
        tree.root.right.right.left = new Node(6);
        tree.root.right.right.right = new Node(7);

        // Function call
        System.out.println("Maximum width is "
                           + tree.getMaxWidth(tree.root));
    }
}
```

```
// This code has been contributed by Mayank Jaiswal
```


Output

```
Maximum width is 3
```

Time Complexity: O(N) where N is the total number of nodes in the tree. Every node of the tree is processed once and hence the complexity is O(N).
Auxiliary Space: O(w) where w is the maximum width of the tree.