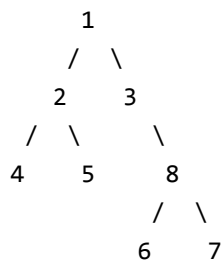
GeeksforGeeks
A computer science portal for geeks
Placements Practice GATE CS IDE Q&A
GeeksQuiz

Maximum width of a binary tree

Given a binary tree, write a function to get the maximum width of the given tree. Width of a tree is maximum of widths of all levels.

Let us consider the below example tree.



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.

Method 1 (Using Level Order Traversal)

This method mainly involves two functions. One is to count nodes at a given level (getWidth), and other is to get the maximum width of the tree(getMaxWidth). getMaxWidth() makes use of getWidth() to get the width of all levels starting from root.

```
/*Function to print level order traversal of tree*/
getMaxWidth(tree)
maxWdth = 0
for i = 1 to height(tree)
  width =   getWidth(tree, i);
```

```
  if(width > maxWdth)
      maxWdth  = width
return width
```

```
/*Function to get width of a given level */
getWidth(tree, level)
if tree is NULL then return 0;
if level is 1, then return 1;
else if level greater than 1, then
    return getWidth(tree->left, level-1) +
    getWidth(tree->right, level-1);
```

C

```
#include <stdio.h>
#include <stdlib.h>

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

/*Function protoypes*/
int getWidth(struct node* root, int level);
int height(struct node* node);
struct node* newNode(int data);

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

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

  return maxWidth;
}

/* Get width of a given level */
int getWidth(struct node* root, int level)
{

  if(root == NULL)
    return 0;
```

```
  if(level == 1)
    return 1;

  else if (level > 1)
    return getWidth(root->left, level-1) +
             getWidth(root->right, level-1);
}


/* 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(struct 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);
   }
}
/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
  struct node* node = (struct node*)
                       malloc(sizeof(struct node));
  node->data = data;
  node->left = NULL;
  node->right = NULL;
  return(node);
}
/* Driver program to test above functions*/
int main()
{
  struct node *root = newNode(1);
  root->left        = newNode(2);
  root->right       = newNode(3);
  root->left->left  = newNode(4);
  root->left->right = newNode(5);
  root->right->right = newNode(8);
  root->right->right->left  = newNode(6);
  root->right->right->right  = newNode(7);

  /*
   Constructed bunary tree is:
          1
        /  \
       2    3
     /  \     \
    4   5     8
              /  \
             6   7
  */
  printf("Maximum width is %d \n", getMaxWidth(root));
  getchar();
  return 0;
}
```

Run on IDE

Java

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

// A binary tree node
class Node {

    int data;
    Node left, right;

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

class BinaryTree {

    static 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);
        }
    }

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

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

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

Run on IDE

Time Complexity: O(n^2) in the worst case.

We can use Queue based level order traversal to optimize the time complexity of this method. The Queue based level order traversal will take O(n) time in worst case. Thanks to Nitish, DivyaC and tech.login.id2 for suggesting this optimization. See their comments for implementation using queue based traversal.

Method 2 (Using Preorder Traversal)

In this method we create a temporary array count[] of size equal to the height of tree. We initialize all values in count as 0. We traverse the tree using preorder traversal and fill the entries in count so that the count array contains count of nodes at each level in Binary Tree.

C

```
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
```

```
{
    int data;
    struct node* left;
    struct node* right;
};

// A utility function to get height of a binary tree
int height(struct node* node);

// A utility function to allocate a new node with given data
struct node* newNode(int data);

// A utility function that returns maximum value in arr[] of size n
int getMax(int arr[], int n);

// A function that fills count array with count of nodes at every
// level of given binary tree
void getMaxWidthRecur(struct node *root, int count[], int level);


/* Function to get the maximum width of a binary tree*/
int getMaxWidth(struct node* root)
{
  int width;
  int h = height(root);

  // Create an array that will store count of nodes at each level
  int *count = (int *)calloc(sizeof(int), h);

  int level = 0;

  // Fill the count array using preorder traversal
  getMaxWidthRecur(root, count, level);

  // Return the maximum value from count array
  return getMax(count, h);
}

// A function that fills count array with count of nodes at every
// level of given binary tree
void getMaxWidthRecur(struct node *root, int count[], int level)
{
  if(root)
  {
    count[level]++;
    getMaxWidthRecur(root->left, count, level+1);
    getMaxWidthRecur(root->right, count, level+1);
  }
}


/* 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(struct 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);
   }
}
/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
  struct node* node = (struct node*)
                      malloc(sizeof(struct node));
  node->data = data;
  node->left = NULL;
  node->right = NULL;
  return(node);
}

// Return the maximum value from count array
int getMax(int arr[], int n)
{
   int max = arr[0];
   int i;
   for (i = 0; i < n; i++)
   {
       if (arr[i] > max)
          max = arr[i];
   }
   return max;
}

/* Driver program to test above functions*/
int main()
{
  struct node *root = newNode(1);
  root->left        = newNode(2);
  root->right       = newNode(3);
  root->left->left  = newNode(4);
  root->left->right = newNode(5);
  root->right->right = newNode(8);
  root->right->right->left  = newNode(6);
  root->right->right->right  = newNode(7);

  /*
   Constructed bunary tree is:
          1
        /  \
       2    3
     /  \     \
    4   5     8
              /  \
             6   7
  */
  printf("Maximum width is %d \n", getMaxWidth(root));
  getchar();
  return 0;
}
```

Run on IDE

Java

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

```
// A binary tree node
class Node {

    int data;
    Node left, right;

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

class BinaryTree {

    static Node root;


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

        // Create an array that will store count of nodes at each level
        int count[] = new int[10];

        int level = 0;

        // Fill the count array using preorder traversal
        getMaxWidthRecur(node, count, level);

        // Return the maximum value from count array
        return getMax(count, h);
    }

    // A function that fills count array with count of nodes at every
    // level of given binary tree
    void getMaxWidthRecur(Node node, int count[], int level) {
        if (node != null) {
            count[level]++;
            getMaxWidthRecur(node.left, count, level + 1);
            getMaxWidthRecur(node.right, count, level + 1);
        }
    }

    /* 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);
        }
    }

    // Return the maximum value from count array
    int getMax(int arr[], int n) {
        int max = arr[0];
```

```
        int i;
        for (i = 0; i < n; i++) {
            if (arr[i] > max) {
                max = arr[i];
            }
        }
        return max;
    }

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

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

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

Run on IDE

Thanks to Raja and jagdish for suggesting this method.

Time Complexity: O(n)

Please write comments if you find the above code/algorithm incorrect, or find better ways to solve the same problem.

141 Comments Category: Trees

Related Posts:

- Check if removing an edge can divide a Binary Tree in two halves
- Check sum of Covered and Uncovered nodes of Binary Tree
- Lowest Common Ancestor in a Binary Tree | Set 2 (Using Parent Pointer)
- Construct a Binary Search Tree from given postorder
- BFS vs DFS for Binary Tree
- Maximum difference between node and its ancestor in Binary Tree
- Inorder Non-threaded Binary Tree Traversal without Recursion or Stack
- Check if leaf traversal of two Binary Trees is same?

(Login to Rate and Mark)

2.6 Average Difficulty : **2.6/5.0**
Based on **14** vote(s)

Add to TODO List
Mark as DONE

Like Share 6 people like this.

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

@geeksforgeeks, Some rights reserved Contact Us! About Us! Advertise with us!