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.

```
         1
        /  \
       2    3
     /  \     \
    4    5     8
              /  \
             6    7
```

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.

Google™ Custom Search

GeeksforGeeks

52,731 people like GeeksforGeeks.

Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion
Geometric Algorithms

```
/*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;
```

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

```c
  /* 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);
}
```

```c
/* 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;
}
```

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.

Recent Comments

affiszerv Your example has two 4s on row 3, that's why it...

Backtracking | Set 7 (Sudoku) · 43 minutes ago

RVM Can someone please elaborate this Qs from above...

Flipkart Interview | Set 6 · 1 hour ago

Vishal Gupta I talked about as an Interviewer in general,...

Software Engineering Lab, Samsung Interview | Set 2 · 1 hour ago

@meya Working solution for question 2 of 4f2f round....

Amazon Interview | Set 53 (For SDE-1) · 1 hour ago

sandeep void rearrange(struct node *head) {...

Given a linked list, reverse alternate nodes and append at the end · 3 hours ago

Neha I think that is what it should return as, in...

Find depth of the deepest odd level leaf node · 3 hours ago

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.

```
#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
```

```c
  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);
  }
}
```

```c
/* 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;
}
```

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.

Related Tpoics:

- Print a Binary Tree in Vertical Order | Set 2 (Hashmap based Method)
- Print Right View of a Binary Tree
- Red-Black Tree | Set 3 (Delete)
- Construct a tree from Inorder and Level order traversals
- Print all nodes at distance k from a given node
- Print a Binary Tree in Vertical Order | Set 1
- Interval Tree
- Check if a given Binary Tree is height balanced like a Red-Black Tree

5 Tweet 0 4

Writing code in comment? Please use **ideone.com** and share the link here.

39 Comments GeeksforGeeks

Sort by Newest

Join the discussion

Join the discussion...

AlienOnEarth · 4 days ago

We can also accomplish this using level order traversal in o(n) time.

Algorithm:

1.) traverse the tree in level order fashion.
2.) while traversing, keep track of max number of nodes in each level (that wo
3.) return the value with maximum number of nodes per level.

Reply · Share ›

Suryabhan Singh · 7 months ago

another O(n) solution

```
void lvlorder(struct node *s)
{
    queue<struct node="" *=""> q;
    int max=1,lvl=1;
    struct node * temp=s;
    q.push(s);
    int curr=1,next=0;
    while(!q.empty())
    {
        temp=q.front();
        q.pop();
        curr--;
        if(temp)
```

see more

Reply • Share ›

Guest • 7 months ago

another O(n) solution

```
void lvlorder(struct node *s)
{
    queue<struct node="" *=""> q;
    int max=1,lvl=1;
    struct node * temp=s;
    q.push(s);
    int curr=1,next=0;
    while(!q.empty())
    {
        temp=q.front();
        q.pop();
        curr--;
        if(temp)
        {
            q.push(temp->l);
```

see more

Reply • Share ›

pavansrinivas • 7 months ago

Using LevelOrder in JAVA

```
void widthOfTree(){
        Node temp = root;
        Queue<node> q = new LinkedList<node>();
        int wid = 0;
        int max_wid = 0;
        q.add(temp);
        q.add(null);
        while(!q.isEmpty()){
            temp = q.remove();
            if(temp==null){
                if(!q.isEmpty()){
                    q.add(null);
                }
                if(wid>max_wid){
                    max_wid = wid;
                }
```

see more

^ | ⌄ • Reply • Share ›

krishna • 8 months ago

int treedia(btn* node, int &dia)
{
if(node==0) return 0;
int l = treedia(node->left,dia);
int r = treedia(node->right,dia);
int ch = l+r+1;
dia = dia < ch ? ch : dia;
return l>= r ? l+1: r+1;
};

^ | ⌄ • Reply • Share ›

Chandu • 8 months ago

// Using Level Order Traversal(using) only but some what simplified better :)
#include
int width(struct node *root)

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

list q;
q.push_back(root);

int max_width = 0;
int wid=0;

struct node *end_node = root;

while(!q.empty())
{
struct node *temp = q.front();
q.pop_front();
wid++;
```

see more

Reply • Share ›

Avinash Abhi • 9 months ago

Using level order it can be easily solved in O(n).

```
maxWidth(Node* root)
{
int max_width=0;
queue<Node*> Queue;
Queue.push(root);
while(! Queue.empty())
{

int count=Queue.size();.

max_width=max(max_width, count);
```

while(count--).

{.

Node* temp=Queue.front();.

see more

Reply • Share ›

Ujjwal Arora • 10 months ago

```
 int ar[height_of_tree] = {0};

void findWidth(Tree *node, int i)
{
    if(node==NULL)
        return;

    ar[i]++;
    findWidth(node->left,i+1);
    findWidth(node->right,i+1);
}

main()
{
findWidth(root,0);

cout<< max of ar[];
}
```

Reply • Share ›

Durga Guntoju • a year ago

```
#include
#include
struct node
{
int data;
struct node* left;
struct node* right;
};
void getMaxWidth(struct node* root,int *Widths,int level)
{
if(root)
{
Widths[level]++;
getMaxWidth(root->left,&Widths[0],level+1);
getMaxWidth(root->right,&Widths[0],level+1);
}
}
int height(struct node* node)
```

see more

^ | ˅ • Reply • Share ›

Rajneesh • a year ago

```
int MaximumWidth(BinaryTree *root) {
 if (!root) return 0;

 queue<BinaryTree*> nodesQueue;
 int max_so_far=1;
 nodesQueue.push(root);
 int nodesInCurrentLevel = 1;
 int nodesInNextLevel = 0;
```

```
while (!nodesQueue.empty()) {
  BinaryTree *currNode = nodesQueue.front();
  nodesQueue.pop();
  nodesInCurrentLevel--;
  if (currNode) {
    nodesQueue.push(currNode->left);
    nodesQueue.push(currNode->right);
    nodesInNextLevel += 2;
  }
```

see more

• Reply • Share ›

prateek • 2 years ago

```
[sourcecode language="C++"]
#include <queue>
#define DUMMY NULL
int max_width(Node * root)
{
queue<Node *>q;
int max_size=0,i=0;
q.insert(root);
q.insert(DUMMY);
while(!q.empty()){
Node *x=q.dequeue();
if(x!=DUMMY){
if(x->left) q.insert(x->left);
if(x->right) q.insert(x->right);
i++;
}else{
q.insert(DUMMY);
if(max_size<i){
max_size=i;
}
i=0;
}
}
return max_size;
}
```

Reply • Share ›

prateek prateek • 2 years ago

sorry....just add one condition check around initial inserts to the queue

```
if(root){
q.insert(root);
q.insert(DUMMY);
}
```

now it handles the base case also.....rest code is as it is given above ir
then plz lemme know..:)

Reply • Share ›

Kumar Prashant prateek • 4 months ago

u have inqueued the null only one time. for the first level only.
i think in the else condition, u should add one more statement li
levels.

Reply • Share ›

GeeksforGeeks • 2 years ago

@Raja and @jagdish: Thanks for suggesting a new method. We have include

@Nitish, @DivyaC and @tech.login.id2: Thanks for suggesting the optimizatic
a note for it

a note for it.

Reply • Share ›

Raja • 3 years ago

```
 N = height of the tree;
static int COUNT[N]; initialize all elements to 0.

computeWidth(root,level)
{
   if( root == null ) return 0;
   COUNT[level]++;
   computeWidth(root->left,level+1);
   computeWidth(root->right,level+1);
}
int MAX = 0;
int max_width()
{
  for(int i =1 ;i<=N;i++){
   if ( MAX < COUNT[i]){
        MAX = COUNT[i];

 return MAX;
}
```

Correct me if i&#039m wrong....

Reply • Share ›

Jagdish • 3 years ago

```
 int Width(Node * root, int level, Hashtable ht, int * max)
{
        if(root == null) return max;

        if(!ht.haskey(level))
                ht.add(level, 0)

        ht[level] ++;

        if(max < ht[level])
                max = ht[level];

        width(root->left, level + 1, ht);
        width(root->right, level + 1, ht);

        return max;

}
```

Reply • Share ›

darkprince • 3 years ago

This could be done with BFS . Just a little modification is required such that w
children present between the parent . A queue will be required.
Correct me if i am wrong .

Reply • Share ›

amit darkprince • 2 years ago

I agree to this. However an additional space requirement, if using FIFO
an issue.

1 • Reply • Share ›

Dhanasekar • 3 years ago

There is another way i could think of

have two queue S1 and S2, both are empty.

Enque the S1 with root.
S1 is active, S2 is inactive

```
count = 0;
max_width = 0;
deque the Node from active queue
enque node->left into inactive queue
enque node->right into inactive queue
count++
repeat this until the active queue is empty.
when active queue becomes empty
if(max_width<count)
max_width = count
swap the active and inactive queues and set count=0;
repeat this until both the queue becomes empty.
```

the maximum queue size i.e space complexity would be O(m) where m is the
run time would be O(n) where n is the number of nodes in the tree.

• Reply • Share ›

Mike Hang • 3 years ago
I would use the same basic idea of BFS with additional variable, say count.

1. Set count to 1 before the loop
2. Decrease count by 1 after dequeue
3. Add left and right node if there are any
4. Check if count == 0. This flag tells we are switching the level
- Reset count to queue size

• Reply • Share ›

bsh • 4 years ago

I dont know whether it will work!!!

But cant we modify "Print nodes at k distance from root" to know the length.

if(k==0){

push(root->data);

return;}

Stack pointer value will give u width of binary tree at k level.

Reply • Share ›

vibhav3008 • 4 years ago

```
 void findwidth(node *root, int width[],int level)
{
    if(root!=NULL)
    {
        width[level]++;
        findwidth(root->left,width,level+1);
        findwidth(root->right,width,level+1);
    }
}

void findmaxwidth(node *root)
{
    cout<<"came here"<<endl;
    int h=height(root);
    int widthlist[h];
    for(int i=0;i<h;i++)
    {
        widthlist[i]=0;
```

see more

Reply • Share ›

aravindh • 4 years ago

what will happen if the i/p is a left-skewed / right-skewed tree??
since at each level der s 1ly one node ... how 2 proceed?

Reply • Share ›

tech.login.id2 • 4 years ago

Done easily by using Level-Order-Traversal
O(logN) space is required by the queue but that offsets the recursive stack us
This one is more efficient than the given solution.

```
int find_max_width (Tree *node) {

   if (!node)
       return 0;

   q->push (node);

   int w = 1;
   int max_w = 0;
   Tree *lev_start_node = node;


   while (q.empty() == false) {
```

see more

Reply • Share ›

cyberWolf ➦ tech.login.id2 • a year ago

It gives SegFault because if there are some NULL nodes, they are alsc
them using q.first() and try to access their 'left' and 'right', it dumps cor

```
   while (q.empty() == false) {
    while (q.empty() == false) {

            node = q.first();
            if(node->left)
                    q->push (node->left);

            if(node->right)
                    q->push (node->right);

            if (lev_start_node == node || q.empty() == true) {
                    if(node->left)
                            lev_start_node = node->left;
                    else if(node->right)
                            lev_start_node = node->right;
                    if (max_w < w)
```

see more

^ | v • Reply • Share ›

shek8034 ➦ cyberWolf • 11 months ago

Nice one.... Thanks :)

^ | v • Reply • Share ›

tech.login.id2 • 4 years ago

I guess the level-order traversal solutions given by Nilesh and DivyaC are both as optimized solution to the given solution.

^ | v • Reply • Share ›

tech.login.id2 • 4 years ago

There is no need to compute the height of the tree.
The loop using it can be terminated when width returned is zero.

Also, a version using Queues is better because it will save repeated traversing

What we need is BFS and some kind of manipulation of 2-3 integers like level,

Reply • Share ›

Sandeep → tech.login.id2 • 4 years ago

Good one!! We will make the suggested changes.

Reply • Share ›

Gagan Arora • 4 years ago

I am not able to understand the height() function.
Wont it always return the value 1.

Reply • Share ›

Sandeep → Gagan Arora • 4 years ago

It correctly returns the height. See http://geeksforgeeks.org/?p=64... fo

Reply • Share ›

sunny • 4 years ago

if tree is sparse...then is the above solution effective?

Reply • Share ›

Anshul • 4 years ago

If anyone is interested in teaching data structure and solving problem on hourl
anshubansal2000 at yahoo.com

thanks

Reply • Share ›

Anony → Anshul • 3 years ago

u got anyone ??? :P

Reply • Share ›

Nilesh • 4 years ago

This is the modified function that returns maxwidth with use of a queue. It basi
dummy node to figure out if a level ends.

```
 #include <queue>
int maxWidth(struct node* root){
  if(root == NULL) return 0;
  int width=1;
  std::queue<struct node*> myqueue;
  myqueue.push(root);
  // Adding dummy Node to signify the end of a level
  myqueue.push(NULL);

  int count=0;
  while(!myqueue.empty()){
    struct node *temp=myqueue.front();
    myqueue.pop();
    // temp = NULL denotes end of one level
    if(temp == NULL){
```

see more

Reply • Share ›

geek4u Nilesh • 4 years ago

You should have enqueue/dequeue operations on a queue, not push/p

Reply • Share ›

Nilesh geek4u • 4 years ago

Please refer this

http://www.cplusplus.com/reference/stl/queue/

It has 2 function for FIFO queue as "push" .. which pushes the
"pop" which removes the element at the front of the queue. I ha

Reply • Share ›

DivyaC • 4 years ago

```
int maxWidth(Node root){// total algo n+log n
        if(root==null) return 0;
        Set<Node> levelEnds=getLevelEnds(root);
        int width=1;
        Queue<Node> Q=new Queue();
        Q.add(root);

        int count=0;
        while(!Q.empty){
                Node temp=Q.dequeue();
                count++;
                if(levelEnds.contains(temp)){
                        if(width< count){
                                width=count;
                        }
                        count=0;
                }
                if(temp.left!=null) Q.add(temp.left);
```

see more

Reply • Share ›

Nil DivyaC • 4 years ago

The above solution wont work because of the getLevelEnds being wro
mirror of the one give in the example above, the function getLevelEnds
in it which is wrong.

Reply • Share ›