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

Linked complete binary tree & its creation

A complete binary tree is a binary tree where each level ‘l’ except the last has 2^l nodes and the nodes at the last level are all left aligned. Complete binary trees are mainly used in heap based data structures.
The nodes in the complete binary tree are inserted from left to right in one level at a time. If a level is full, the node is inserted in a new level.

Below are some of the complete binary trees.

```
      1
    /  \
   2    3

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

Below binary trees are not complete:

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

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

Complete binary trees are generally represented using arrays. The array representation is better because it

doesn't contain any empty slot. Given parent index i, its left child is given by 2 * i + 1 and its right child is given by 2 * i + 2. So no extra space is wasted and space to store left and right pointers is saved. However, it may be an interesting programming question to create a Complete Binary Tree using linked representation. Here Linked mean a non-array representation where left and right pointers(or references) are used to refer left and right children respectively. How to write an insert function that always adds a new node in the last level and at the leftmost available position?

To create a linked complete binary tree, we need to keep track of the nodes in a level order fashion such that the next node to be inserted lies in the leftmost position. A queue data structure can be used to keep track of the inserted nodes.

Following are steps to insert a new node in Complete Binary Tree.

1. If the tree is empty, initialize the root with new node.

2. Else, get the front node of the queue.

…….If the left child of this front node doesn't exist, set the left child as the new node.

…….else if the right child of this front node doesn't exist, set the right child as the new node.

3. If the front node has both the left child and right child, Dequeue() it.

4. Enqueue() the new node.

Below is the implementation:

```
// Program for linked implementation of complete binary tree
#include <stdio.h>
#include <stdlib.h>

// For Queue Size
#define SIZE 50

// A tree node
struct node
{
    int data;
    struct node *right,*left;
};

// A queue node
struct Queue
{
    int front, rear;
    int size;
    struct node* *array;
};

// A utility function to create a new tree node
struct node* newNode(int data)
{
    struct node* temp = (struct node*) malloc(sizeof( struct node ));
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// A utility function to create a new Queue
struct Queue* createQueue(int size)
{
```

```
    struct Queue* queue = (struct Queue*) malloc(sizeof( struct Queue ));

    queue->front = queue->rear = -1;
    queue->size = size;

    queue->array = (struct node**) malloc(queue->size * sizeof( struct node* ));

    int i;
    for (i = 0; i < size; ++i)
        queue->array[i] = NULL;

    return queue;
}

// Standard Queue Functions
int isEmpty(struct Queue* queue)
{
    return queue->front == -1;
}

int isFull(struct Queue* queue)
{  return queue->rear == queue->size - 1; }

int hasOnlyOneItem(struct Queue* queue)
{  return queue->front == queue->rear;  }

void Enqueue(struct node *root, struct Queue* queue)
{
    if (isFull(queue))
        return;

    queue->array[++queue->rear] = root;

    if (isEmpty(queue))
        ++queue->front;
}

struct node* Dequeue(struct Queue* queue)
{
    if (isEmpty(queue))
        return NULL;

    struct node* temp = queue->array[queue->front];

    if (hasOnlyOneItem(queue))
        queue->front = queue->rear = -1;
    else
        ++queue->front;

    return temp;
}

struct node* getFront(struct Queue* queue)
{  return queue->array[queue->front]; }

// A utility function to check if a tree node has both left and right children
int hasBothChild(struct node* temp)
{
    return temp && temp->left && temp->right;
}

// Function to insert a new node in complete binary tree
void insert(struct node **root, int data, struct Queue* queue)
{
    // Create a new node for given data
```

```
    struct node *temp = newNode(data);

    // If the tree is empty, initialize the root with new node.
    if (!*root)
        *root = temp;

    else
    {
        // get the front node of the queue.
        struct node* front = getFront(queue);

        // If the left child of this front node doesn’t exist, set the
        // left child as the new node
        if (!front->left)
            front->left = temp;

        // If the right child of this front node doesn’t exist, set the
        // right child as the new node
        else if (!front->right)
            front->right = temp;

        // If the front node has both the left child and right child,
        // Dequeue() it.
        if (hasBothChild(front))
            Dequeue(queue);
    }

    // Enqueue() the new node for later insertions
    Enqueue(temp, queue);
}

// Standard level order traversal to test above function
void levelOrder(struct node* root)
{
    struct Queue* queue = createQueue(SIZE);

    Enqueue(root, queue);

    while (!isEmpty(queue))
    {
        struct node* temp = Dequeue(queue);

        printf("%d ", temp->data);

        if (temp->left)
            Enqueue(temp->left, queue);

        if (temp->right)
            Enqueue(temp->right, queue);
    }
}

// Driver program to test above functions
int main()
{
    struct node* root = NULL;
    struct Queue* queue = createQueue(SIZE);
    int i;

    for(i = 1; i <= 12; ++i)
        insert(&root, i, queue);

    levelOrder(root);

    return 0;
```

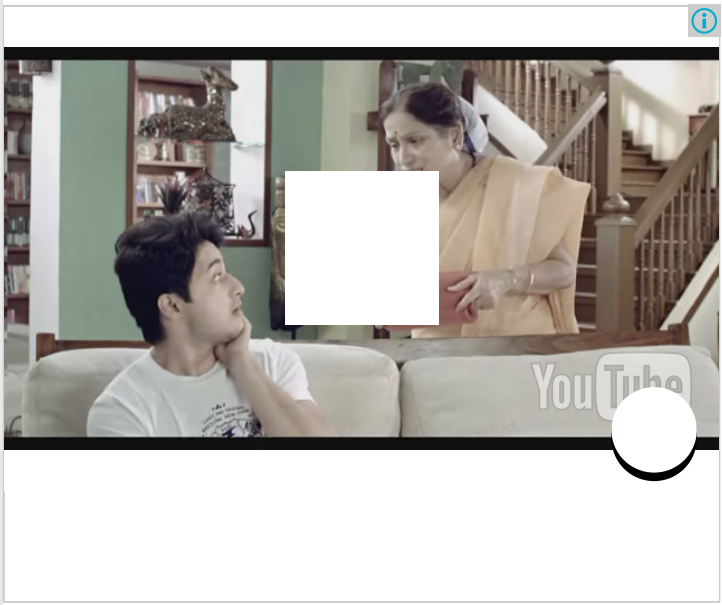
```
}
```

Run on IDE

Output:

```
1 2 3 4 5 6 7 8 9 10 11 12
```

This article is compiled by **Aashish Barnwal** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



60 Comments Category: Trees Tags: Complete Binary Tree

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)

3.2 Average Difficulty : **3.2/5.0**
Based on **4** vote(s)

Add to TODO List

Mark as DONE

Like Share 25 people like this.

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

60 Comments GeeksforGeeks 1 Login

Recommend 1 Share Sort by Newest

Join the discussion…

Ranjit Gouda • 4 months ago

Good one appreciate it...:)

• Reply • Share ›

Yeshwanth Selvaraj • 7 months ago

http://ideone.com/o48rb8 -->> fast and simple implementation with the help of an array of structures..

Based on the fact that array with even index always points to right node and array with odd pints to left node. So create a tree accordingly with the address of the corresponding parent node.

for example:

2 (i) + 1 = x; (value of the left tree)

so,

i = (x - 1) / 2; (i is the index of the parent node..)

where i is the corresponding root node.. let arr be the array storing the address of each nodes. Since i is odd in the above,

arr[i] -> left = newNode(ANY VALUE); The implementation is also quiet simple -->>http://ideone.com/o48rb8

similarly calculate for even index of the arrays.. and use 2(i)+2 instead..

• Reply • Share ›

Raghav Khandelwal Yeshwanth Selvaraj • 9 days ago

You actually implemented the heap data structure!

• Reply • Share ›

NIKHIL SINGH • 9 months ago

after dequeue there must be recall of insert function for the same element to be inserted ...isnt it?? or queue front should called again such that front->left=temp;

,..isnt it?? or queue.front should called again such that front->left=temp;

1 • Reply • Share ›

code down ➚ NIKHIL SINGH • 8 months ago

not required. check the last If. Its not else if.

1 • Reply • Share ›

NIKHIL SINGH ➚ code down • 8 months ago

ok got it thanks :-)

• Reply • Share ›

Aakash Parashar • 9 months ago

Easy STL implementation.

```
#include <iostream>

#include <queue>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

struct node

{

    int data;
```

see more

1 • Reply • Share ›

Arshu Jain • 9 months ago

Another approach is:-
for root check the minimum depth in leftsubtree and minimum depth in right subtree where the new node can be inserted.
l= min depth in left subtree where it can be inserted.
r= ,in height in right subtree where it can be inserted.
if(l>r)
insert in rightsubtree.
else
insert in leftsubtree.

1 • Reply • Share ›

Learner_187 ➦ Arshu Jain • 9 months ago
Good approach :)
• Reply • Share ›

Ayush • 9 months ago
struct node* *array;
what does this mean plz explain
• Reply • Share ›

DS+Algo • 10 months ago
No need to use hasBothChild() function.

```
else if (!front->right)
front->right = temp;
if (hasBothChild(front))
Dequeue(queue);
```

It can be replaced by this:

```
else
{
front->right=temp;
Dequeue(queue);
}
```

Because we are already keeping track of node with both child i.e. when we are filling its right, we dequeue it.
6 • Reply • Share ›

Mayank Kunawat • 10 months ago
whooa...! very intelligent code (tough to understand at first glance) !!!
• Reply • Share ›

ashish • a year ago
Code not good. Could have been smaller and better to understand. Seriously not good. Use less space.
• Reply • Share ›

sad • a year ago
idiotic code i have ever seen
• Reply • Share ›

Vishal Sharma • a year ago
According to the above code, In the else part of insert function,
first we checked left child for null. If the left child of this front node doesn't exist, new node is inserted here

is inserted here.

If left child of front node exists, right child of the front node is checked for null. If If the right child of this front node doesn't exist, new node is inserted here.

If front node has both childs, the queue is dequeued.
and new node is enqueued to the queue.

considering this example,

1
/ \
2 3
/ \ /
4 5 6

if we insert an element in this tree, the new element will be enqueued to the queued and will not be inserted anywhere in the tree (according to the code).

To my suprise, the code is working fine. please explain how insertion will be performed in the above tree according to the above code.

• Reply • Share ›

debugger ➚ Vishal Sharma • a year ago

At this point of time, suppose new element is 7.
Here queue will be, {3,4,5,6}.
6 is left child of 3 and right child of 3 is NULL.

So,at this step:
1)7 will be assigned right child of 3.

2)3 will be dequeued.

3)7 will be enqueued.

After above three steps, Queue is {4,5,6,7}
Tree is

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

• Reply • Share ›

Aparna • a year ago

I guess we need to add a "while" loop in the else part. Because here the coed just

dequeue's the queue. However it does not consider the next element.

• Reply • Share ›

NIKHIL SINGH ➚ Aparna • 9 months ago

exactly...or we can recall q.front and front->left=temp
but still code running fine on ide

• Reply • Share ›

code down ➚ NIKHIL SINGH • 8 months ago

not required. check the last If. Its not else If

• Reply • Share ›

Guest ➚ Aparna • a year ago

-

• Reply • Share ›

Devesh Krishnani • a year ago

I am new with pointers so my question may seem a bit childish

struct node * *array;
explain this plz

• Reply • Share ›

Harsh ➚ Devesh Krishnani • a year ago

This is notation for a pointer to a pointer.

• Reply • Share ›

sabertooth • a year ago

```
struct node * buildTree(int array[], int index, int len)
{
if(index >= len) return null;
struct node* root = newNode(array[index]);
root->left = NULL;
root->right = NULL;
if(2 * index + 1 < len){
root->left = buildTree(array, 2 * index + 1, len);

}
if(2 * index + 2 < len){
root->right = buildTree(array, 2 * index + 2, len);
}
return root;
}
```

1 • Reply • Share ›

Pratik • 2 years ago
Where are you linking temp to the tree in case hasBothChild(front) has been satisfied?
• Reply • Share ›

Pratik ➚ Pratik • 2 years ago
sorry, got the thing now. Noticed it a bit late :)
• Reply • Share ›

Batman_ • 2 years ago
But this program will not for more than 50 elements ??

dequeue is only incrementing the front ++queue->front;

please explain....
• Reply • Share ›

Karandeep ➚ Batman_ • a year ago
We need the logic for a circular queue by dividing by mod 50 ?
• Reply • Share ›

Dheeraj Arya • 2 years ago
This question can also be done with the use of array, see this solution also.
http://dj8aprilwrites.blogspot...
• Reply • Share ›

Mr. Lazy ➚ Dheeraj Arya • a year ago
The question itself is about without the use of array! :P
1 • Reply • Share ›

ANA • 2 years ago
nice :D
• Reply • Share ›

samthebest • 2 years ago
very nice code @geeks ...my code would always start enquing from the root for a new insert but ur code (Y)
• Reply • Share ›

SS • 2 years ago
After looking at the program, it seems to be me that the insert functions seems incomplete/incorrect. When the front node is full with left and right already updated, we are just dequeueing it, dont we need to keep traversing the queue until one with either empty left or right is seen so that the newly created node can be added to it.

10 • Reply • Share ›

KD SS • a year ago
I also think the logic is goofed up. From what I understand the queue represents the tree. If you dequeue it you loose the root and keep losing the elements starting from root.
• Reply • Share ›

KD KD • a year ago
Ok I got it now. The queue does not hold the entire tree but a part of it. The queue actually contains only those nodes that have either the left or right or both as empty. Once the node has left and right setup is is dequeued from the queue. The insertion in queue has the breadth first traversal order in tree.
1 • Reply • Share ›

ss KD • 8 months ago
what i also dnt understand is when a node is full then it is just enqued , the element is not added .. how is it working
• Reply • Share ›

samthebest SS • 2 years ago
the code is correct!
1 • Reply • Share ›

vaibhav samthebest • 2 years ago
but how???... I am having the same doubt...i mean when front has both right and left child, how is the new node becoming the front->left->left??
• Reply • Share ›

Chirag Bansal vaibhav • a year ago
@vaibhav front will always have its one of the children node null.
• Reply • Share ›

Ayush Chirag Bansal • 9 months ago
why ? kindly explain
• Reply • Share ›

Chirag Bansal Ayush • 9 months ago
because as soon as its both children are inserted it is dequeued from the queue..!!and as it is a complete binary tree..so nodes will fill up in level order fashion...!
• Reply • Share ›

Ayush ➚ Chirag Bansal • 9 months ago

thanx ambala boy :)

• Reply • Share ›

Pratik ➚ vaibhav • 2 years ago

I'm having the same doubt too. there should be a recursive call, in case both the children exist, with the root as the new front of the queue obtained after dequeueing.
EDIT-
I just noticed, the dequeue is happening the moment the right child is populated. So it will work fine :)

2 • Reply • Share ›

code down ➚ Pratik • 8 months ago

exactly!! it looks like "else if" if not noticed

• Reply • Share ›

shaktiman • 3 years ago

```
 /*
#include <iostream>
#include <queue>
using namespace std;

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



// A utility function to create a new tree node
struct node* newNode(int data)
{
    struct node* temp = new (struct node);
    temp->data = data;
    temp->left = temp->right = NULL;
```

see more

• Reply • Share ›

abhishek08aug • 3 years ago

Intelligent :D

1 • Reply • Share ›

Narendra Soni • 3 years ago
Thank you.
1 • Reply • Share ›

Viky • 3 years ago
Slight confusion ..
inside main function ..
insert(&root, i, queue);

1. &root is used because we are constructing the tree. Am i right?
2. Why we passed just queue and not &queue .. and queue got updated/constructed ..

I understand it is very basic question. But, it would be very helpful if you can explain me ..
1 • Reply • Share ›

Aashish ➦ Viky • 3 years ago
The reason can be explained as follows:
In the insert() function, root must be updated when first node is being created(or in other words, tree is empty). So, address of root has been passed.
Please take a closer look. queue has already been created before call to insert(). So, there is no need of updating the queue pointer. However, the pointers like front and rear can be updated.
1 • Reply • Share ›

debasis sahoo • 3 years ago
Is it possible to do this without using queue ?

```
/* Paste your code here (You may delete these lines if not writing code) */
```

1 • Reply • Share ›

Aashish ➦ debasis sahoo • 3 years ago
Yes, it can be done without using queue. The solution will be analogous to the queue-less level-order traversal. Start from the level 0, check for the first link which is NULL. If no such link found, check for the next level and so on. The inorder algorithm will check for the NULL links from left to right order at a given level.
1 • Reply • Share ›

Load more comments

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