

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
  
```

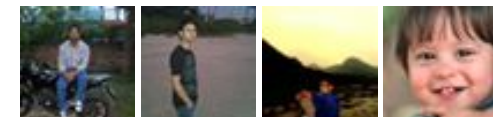
Google™ Custom Search



GeeksforGeeks



52,731 people like [GeeksforGeeks](#).



[Interview Experiences](#)

[Advanced Data Structures](#)

[Dynamic Programming](#)

[Greedy Algorithms](#)

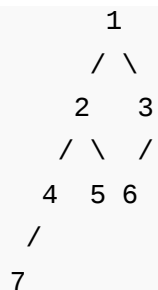
[Backtracking](#)

[Pattern Searching](#)

[Divide & Conquer](#)

[Mathematical Algorithms](#)

[Recursion](#)



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
```

ITT Tech - Official Site

itt-tech.edu

Associate, Bachelor Degree
Programs Browse Programs Now &
Learn More.



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](#)

```

// 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

    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
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);

```

Recent Comments

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

[Backtracking | Set 7 \(Sudoku\) · 35 minutes ago](#)

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

[Flipkart Interview | Set 6 · 55 minutes ago](#)

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

[Software Engineering Lab, Samsung Interview | Set 2 · 55 minutes 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 · 2 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

AdChoices 

[► Binary Tree](#)

[► Linked List](#)

```

// 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)


```

AdChoices 

[► Java to C++](#)

[► Linked Data](#)

[► Creation of Java](#)

AdChoices 

[► Memory Tree](#)

[► Node](#)

[► Queue](#)

```

        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;
}

```

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.



Better Than Hadoop.

HPCC Systems is Big Data Processing and Analytics
Open Source. Proven. Trusted.

 LexisNexis® [Learn More](#) 

Related Topics:

- [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](#)



25



Tweet 3



0

Writing code in comment? Please use [ideone.com](#) and share the link here.

18 Comments [GeeksforGeeks](#)

Sort by Newest ▼



Join the discussion



...on the discussion...



SS • 4 months ago

After looking at the program, it seems to be me that the insert functions seems node is full with left and right already updated, we are just dequeuing it, don't until one with either empty left or right is seen so that the newly created node c

3 ^ | v • Reply • Share ›



shaktiman • 10 months 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](#)

^ | v • Reply • Share ›



abhishek08aug • 11 months ago



Intelligent :D

^ | v • Reply • Share ›



Narendra Soni • a year ago

Thank you.

1 ^ | v • Reply • Share ›



Viky • a year 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/const

I understand it is very basic question. But, it would be very helpful if you can ex

1 ^ | v • Reply • Share ›



Aashish → Viky • a year ago

The reason can be explained as follows:

In the insert() function, root must be updated when first node is being created (if root is empty). So, address of root has been passed.

Please take a closer look. queue has already been created before calling insert(). We are only updating the queue pointer. However, the pointers like front and rear can be updated similarly.

1 ^ | v • Reply • Share ›



debasis sahu • a year ago

Is it possible to do this without using queue ?

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

1 ^ | v • Reply • Share ›



Aashish → debasis sahuo · a year ago

Yes, it can be done without using queue. The solution will be analogous traversal. Start from the level 0, check for the first link which is NULL. If level and so on. The inorder algorithm will check for the NULL links for

1 ^ | v · Reply · Share ›



Madhav → Aashish · a year ago

How to do queue-less level-order traversal?

^ | v · Reply · Share ›



Aashish → Madhav · a year ago

Traverse the tree in such a way that it only outputs the c are traversed, only when the traversal of the data items done as follows:

Do inorder traversal of the tree level by level. Modify the extra argument will be passed to indicate the level. Call 0, 1.... last level.

This approach takes $O(N^2)$ time.

1 ^ | v · Reply · Share ›



Madhav → Aashish · a year ago

Thanks.

Passing level seems to do the trick many times.

1 ^ | v · Reply · Share ›



Bharat Arya · a year ago

In Dequeue() and Enqueue() both, you are incrementing front of queue. Is it co

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

```
++queue->front;
```

1 ^ | v • Reply • Share ›



Aashish → Bharat Arya • a year ago

Yes.

Enqueue() operation:

When the queue is empty, the front(and rear) of the queue needs to be

Dequeue() operation:

If the queue is non-empty and contains more than one element, deletin incremented.

1 ^ | v • Reply • Share ›



algopiggy • a year ago

After this -

```
if (hasBothChild(front))
```

```
    Dequeue(queue);
```

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

shouldn't the "temp" be made the child of the next item in the queue?

^ | v • Reply • Share ›



Aashish → algopiggy • a year ago

Please take a closer look at the algorithm. "temp" is made the child be

The queue is only used to keep track of the parent nodes whose left(ar and right links are assigned, its task is over.

^ | v • Reply • Share ›



Priso → Aashish · a year ago

I guess the inner if-else part can be modified to :

```
if (!front->left)
    front->left = temp;

    // If the right child of this front node doesn't
    // right child as the new node
else (!front->right)
{
    front->right = temp;
    // because we know that front has both left and
    // left child exists because it is complete binary tree
}

Dequeue(queue);
```

let me know if i am wrong.

^ | v · Reply · Share ›



Priso → Priso · a year ago

A small correction!

```
if (!front->left)
    front->left = temp;

    // If the right child of this front node
    // right child as the new node
else (!front->right)
{
    front->right = temp;
    // because we know that front has both left and
    // left child exists because it is complete binary tree
}
```

```
front->right = temp;  
// because we know that front has both l  
// left child exists because it is comple
```

```
Dequeue(queue);  
}
```

1 ^ | v • Reply • Share ›



Aashish ➔ Priso • a year ago

Thanks for the simplification. The function hasBothChilc
clearly specify that the node should be deleted only whe
and right children).

^ | v • Reply • Share ›



Subscribe



Add Disqus to your site

@geeksforgeeks, **Some rights reserved**

Contact Us!

Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team