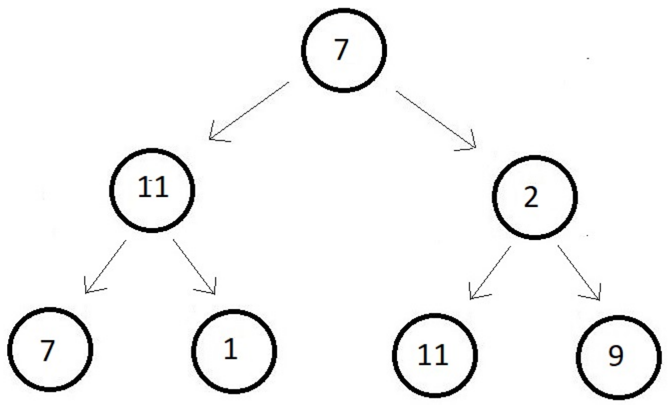
Representation of a Binary Tree

In the previous lecture, we saw the various types of binary trees we have. We looked at their examples, and I am confident that all of you understood everything very clearly. In today's lesson, we'll consider different techniques for representing binary trees in programming, and see which one best suits our needs.
So, the first way to represent a binary tree is by using arrays. We call this ‘array representation’. And this method is not very recommended for representing binary trees. You will very soon know why.

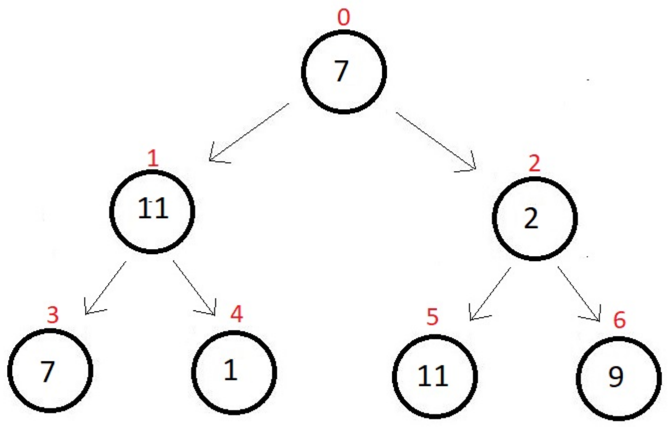
Array representation of Binary trees:

Arrays are linear data structures and for arrays to function, their size must be specified before elements are inserted into them. And this counts as the biggest demerit of representing binary trees using arrays. Suppose you declare an array of size 100, and after storing 100 nodes in it, you cannot even insert a single element further, regardless of all the spaces left in the memory. Another way you’d say is to copy the whole thing again to a new array of bigger size but that is not considered a good practice.

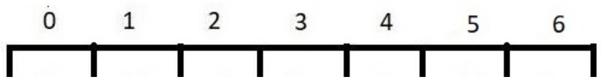
Anyways, we will use an array to represent a binary tree. Suppose we have a binary tree with 7 nodes.

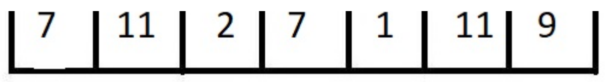


And there are actually a number of ways to represent these nodes via an array. I'll use the most convenient one where we traverse each level starting from the root node and from left to right and mark them with the indices these nodes would belong to.



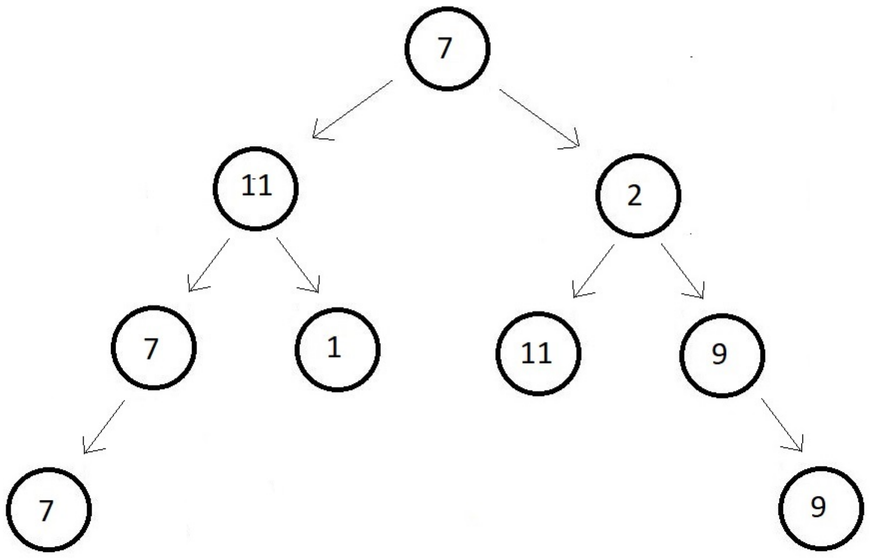
And now we can simply make an array of length 7 and store these elements at their corresponding indices.



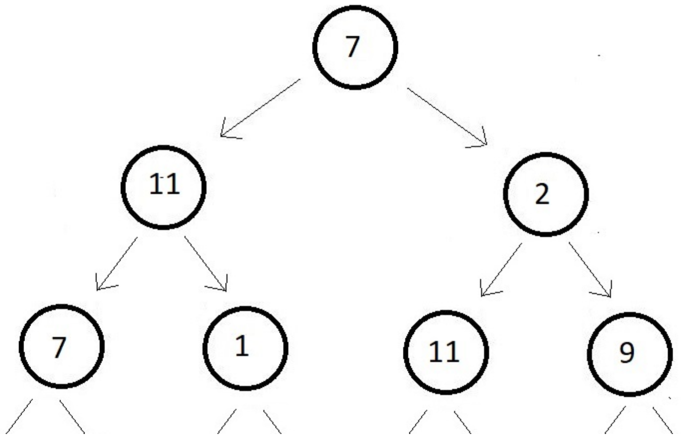


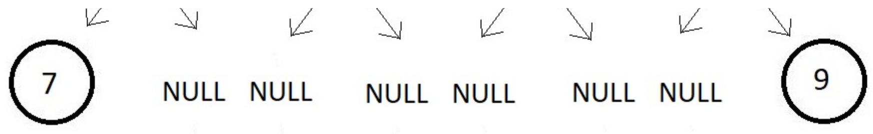
7	11	2	7	1	11	9

And you might be wondering about the cases where the binary is just not perfect. What if the last level has distributed leaves? Then let me tell you, there is a way out for that as well. Let's consider one case here. A binary tree with 9 nodes, and the last two nodes on the extremities of the last level.

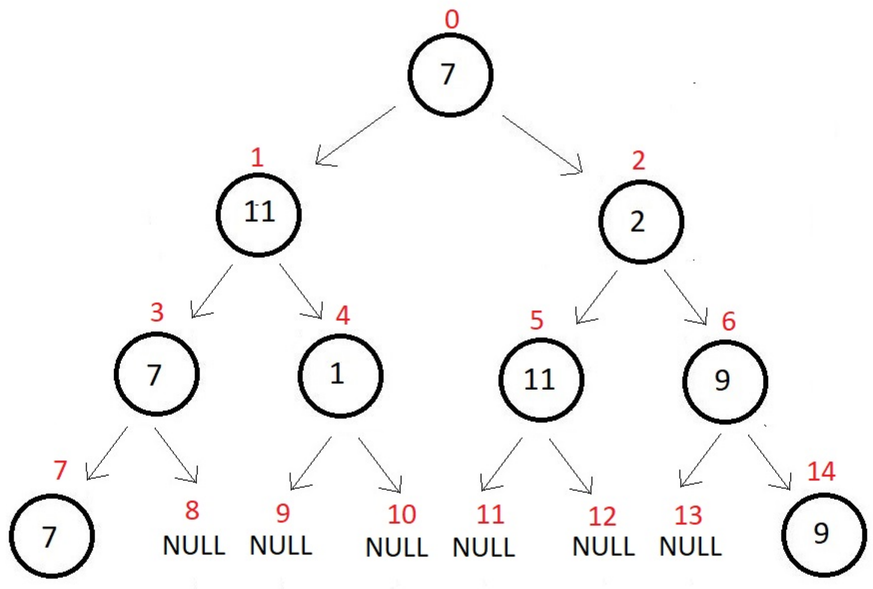


Here, while traversing we get stuck at the 8th index. We don't know if declaring the last node as the 8th index element makes it a general representation of the tree or not. So, we simply make the tree perfect ourselves. We first assume the remaining vacant places to be NULL.

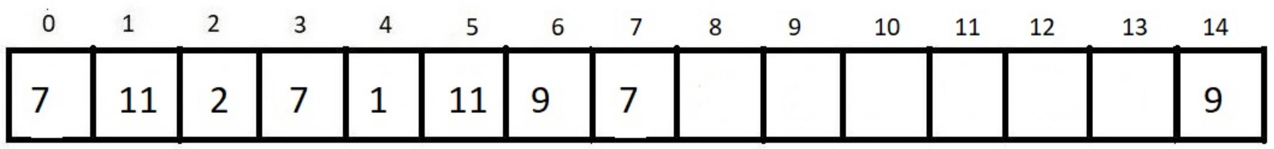




And now we can easily mark their indices from 0 to 14.



And the array representation of the tree looks something like this. It is an array of length 15.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7	11	2	7	1	11	9	7							9

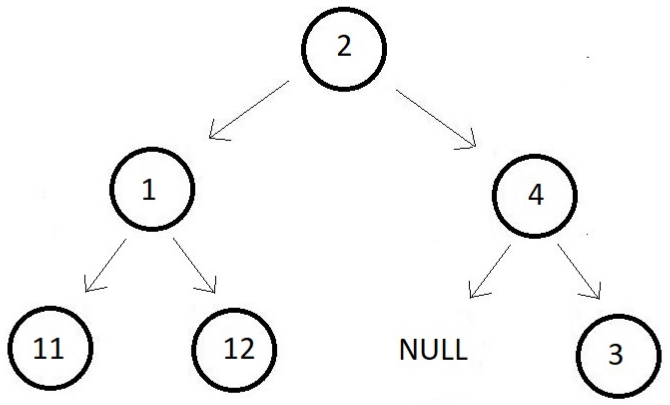
But was this even an efficient approach? Like Binary Trees are made only for efficient traversal and insertion and deletion and using an array for that really makes the process troublesome. Each of these operations becomes quite costly to accomplish. And that size constraint was already for making things worse. So overall, we would say that the array representation of a binary is not a very good choice. And what are the other options?

We have another method to represent binary trees called the linked representation of binary trees. Don't confuse this with linked lists you have studied. And the reason why I am saying that is because linked lists are lists that are linear data structures.

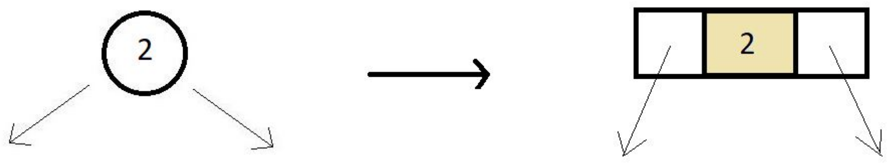
Linked Representation of Binary Trees:

This method of representing binary trees using linked nodes is considered the most efficient method of representation. For this, we use doubly-linked lists. I just hope you recall what doubly-linked lists are. We studied that here in the same playlist Doubly Linked Lists Explained With Code in C Language.

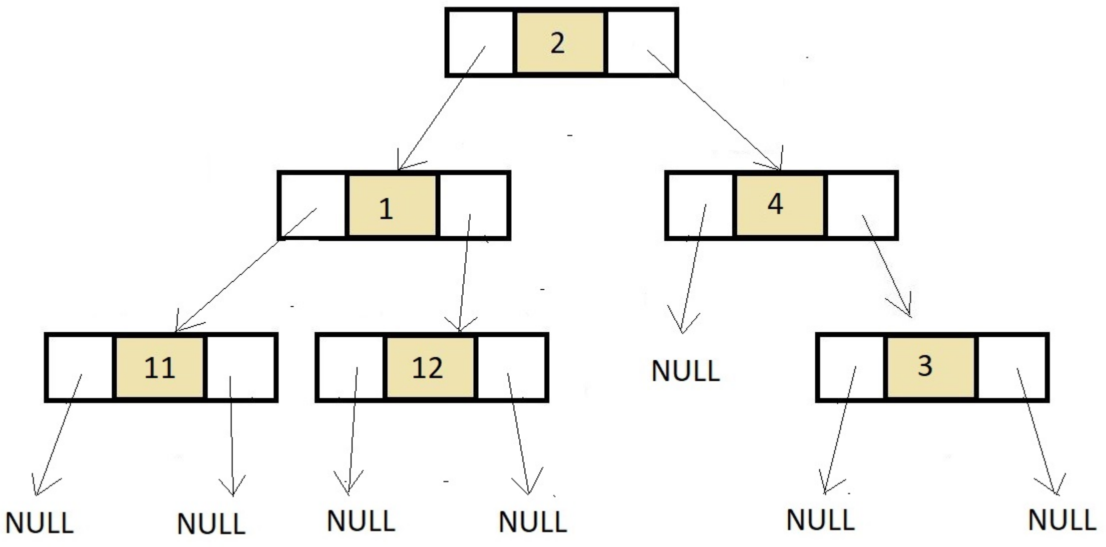
Using links makes the understanding of a binary tree very easy. It actually makes us visualize the tree even. Suppose we have a binary tree of 3 levels.



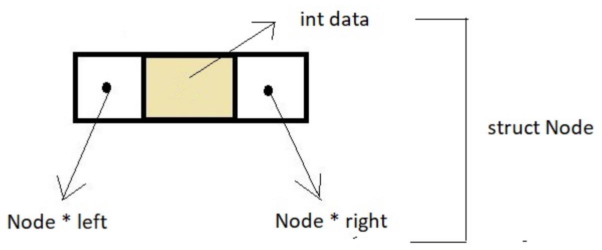
Now if you remember a doubly linked list helped us traversing both to the left and the right. And using that we would create a similar node here, pointing both to the left and the right child node. Follow the below representation of a node here in the linked representation of a binary tree.



You can see how closely this representation resembles a real tree node, unlike the array representation where all the nodes succumbed to a 2D structure. And now we can very easily transform the whole tree into its linked representation which is just how we imagined it would have looked in real life.



So, this was the representation of the binary tree we saw above using linked representation. And what are these nodes? These are structures having three structure members, first a data element to store the data of the node, and then two structure pointers to hold the address of the child nodes, one for the left, and the other for the right.



And let me show you that struct Node definition part in C language:

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

Code Snippet 1: Creating the struct Node

And that was the representation of a binary tree using a linked method. We still have not completely seen the programming part. I hope you could understand and most importantly visualize how the linked representation outperforms the array representation in terms of both implementation and complexity. We will see the programming of this linked representation in the C language in the next lecture.
I appreciate your support throughout. I hope you enjoyed the tutorial. If you genuinely appreciate my work, please let your friends know about this course too. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial where we'll learn to program the linked repression of a binary tree. Till then keep coding.

Previous Next

CodeWithHarry