Theory: Trees in Java

43 minutes 0 / 5 problems solved

Earlier, we considered in detail what trees are. Briefly, a **tree** is a data structure that allows storing data with a **"parent-child"** relationship. For example, the 'boss-subordinate' structure of a company can be represented as a tree. Recall that every tree is a connected graph that does not contain any cycles.

In this topic, we will learn how trees can be stored in computer memory and implement these approaches in Java.

§1. Implementing trees in Java

Often, a **tree** means a **binary tree** since many algorithms are made specifically for them. But first, let's take a look at the data structure for a non-binary tree where nodes can have more than two children:

```
public class TreeNode<T> {
    public T data;
    public TreeNode<T> parent;
    public List<TreeNode<T>> children;

    public TreeNode(T data) {
        this.data = data;
        this.parent = null;
        this.children = new ArrayList<>();

    }

}
```

Having a pointer to the root gives us access to the whole tree. Using the "children" list allows us to assign more than two children to the node.

If we were implementing a **binary tree**, then instead of a list we would have just two children – left and right. This is how `TreeNode` class would look for a binary tree:

```
public class TreeNode<T> {
    public T data;
    public TreeNode<T> parent;
    public TreeNode<T> left;
    public TreeNode<T> right;

    public TreeNode(T data) {
        this.data = data;
        this.parent = null;

        this.left = null;

        this.right = null;

    }

}
```

Using the generic programming approach in implementing the classes above allows us to use different data types for the values of tree nodes. We can store both numeric and symbolic values as well as more complex ones like strings or objects. For instance, in the family tree, each node is an object of the type 'Human', with parameters such as Name, Birthdate, Gender, etc.

To use this data structure, we need to learn how to add elements to our tree. First, let's implement the "Add" function. This function adds a child with a specified value to the current node. As you can see, this method is also a generic method:

Current topic:

Trees in Java

Topic depends on:

✓ Tree

✗ Algorithms in Java

✗ Generics and Object

✗ Generic methods

✗ ArrayList

Topic is required for:

Binary search tree in Java

Binary heap in Java

Table of contents:

↑ Trees in Java

§1. Implementing trees in Java

§2. Another way to store trees in memory

§3. Conclusion

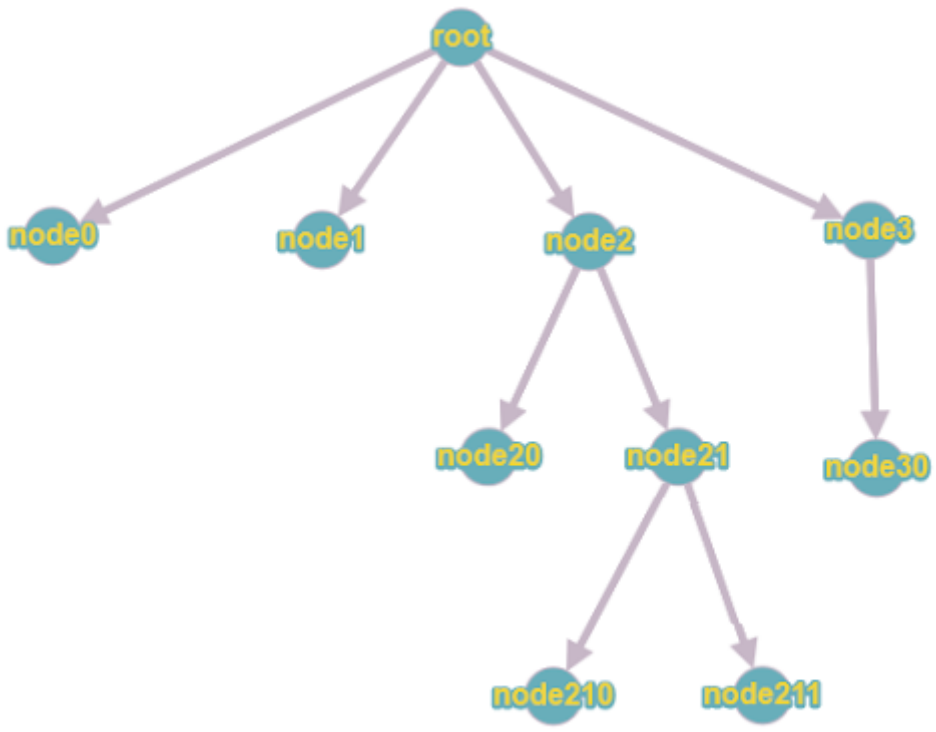
Feedback & Comments

```
public TreeNode<T> add(T value) {
    TreeNode<T> newVertex = new TreeNode<>(value);
    newVertex.parent = this;
    this.children.add(newVertex);
    return newVertex;
}
```

Now let's use this class in practice and create some tree:

```
TreeNode<String> root = new TreeNode<>("root");
{
    TreeNode<String> node0 = root.add("node0");
    TreeNode<String> node1 = root.add("node1");
    TreeNode<String> node2 = root.add("node2");
    {
        TreeNode<String> node20 = node2.add("node20");
        TreeNode<String> node21 = node2.add("node21");
        {

            TreeNode<String> node210 = node21.add("node210");

            TreeNode<String> node211 = node21.add("node211");

        }

    }

    TreeNode<String> node3 = root.add("node3");

    {

        TreeNode<String> node30 = node3.add("node30");

    }

}
```

The code above builds the following tree:

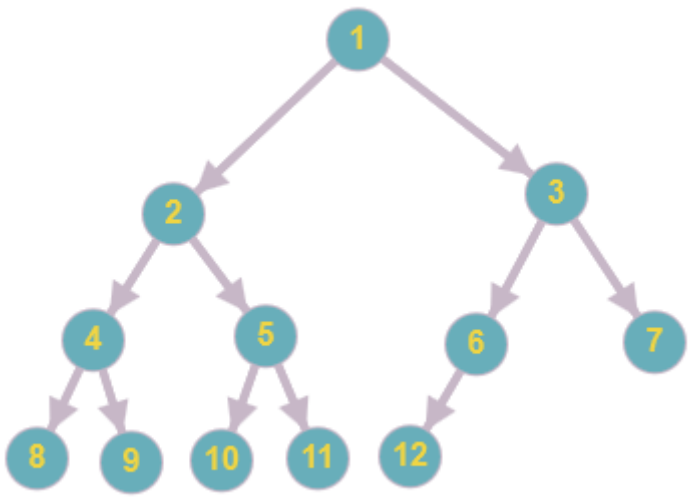


As you may have guessed, curly brackets are not required: they were added as visual clues to reveal the tree's structure.

§2. Another way to store trees in memory

Common ways of describing graphs (connectivity matrix, connectivity list, and incidence matrix) can also be used for describing trees since any tree is by definition a graph.

However, there is an even easier option for binary trees: they can be stored in arrays. If the node is numbered as i, then its children will have numbers $2 \cdot i$ and $2 \cdot i + 1$. Consider the following example:



An array representing this binary tree looks like this:

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

Let's take node 5, for example. If we apply the above formula, its children will have numbers $10 = 5 \cdot 2$ and $11 = 5 \cdot 2 + 1$, which is exactly what we see in the picture.

§3. Conclusion

In this topic, we have learned several ways of how trees can be stored in computer memory and implemented one of these methods in Java. Now, you should be able to apply these methods to efficiently solve problems connected with processing trees. Move on to the practice exercises of this topic and try to solve some of such problems!

Report a typo

26 users liked this theory. **0** didn't like it. **What about you?**

Start practicing

Comments (2) Hints (0) Useful links (0) Show discussion