

Forest Play Ground Revisited- Binary Trees

Please understand, according to Wikipedia, Tree terminology is not well-standardized and varies in the literature. You should read the descriptions carefully in this problem to ensure that you are solving the given problem.

- A rooted binary tree has a root node and every node has at most two children.

Methods for storing binary trees

Arrays

Binary trees can also be stored in breadth-first order as an implicit data structure in arrays, and if the tree is a complete binary tree, this method wastes no space. In this compact arrangement, assuming the root has index 1, a node with index i , has children at indices $2i$ (for the left child) and $2i + 1$ (for the right), while its parent (if any) is found at index $\lfloor \frac{i}{2} \rfloor$. (In this case, $\lfloor x \rfloor$ is the largest (Closes to positive infinity) integer value smaller than or equal to x . For example, $\lfloor 2.9 \rfloor = 2$. For example, the rooted binary tree in Figure 1 and Figure 2 would be represented by the flowing lines of code:

```
int[] figure1 = {-1, 1, 2, 3 , 4, 5, 6, 7};  
int[] figure2 = {-1, 12, 23, 18 , 11, 43, 12, 27,  
                 56, 78, -1, -1, 32, 98};
```

Note: In this problem, every node will contain a **positive** int value. The value -1 will represent a nonexistent node.

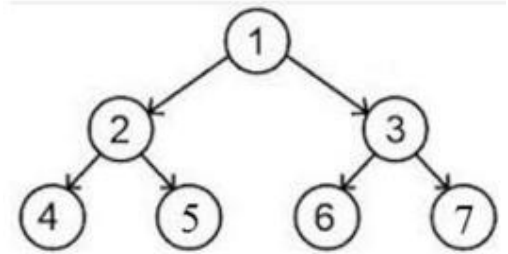


Figure 1

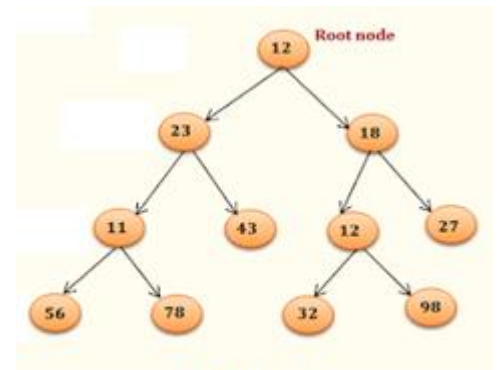
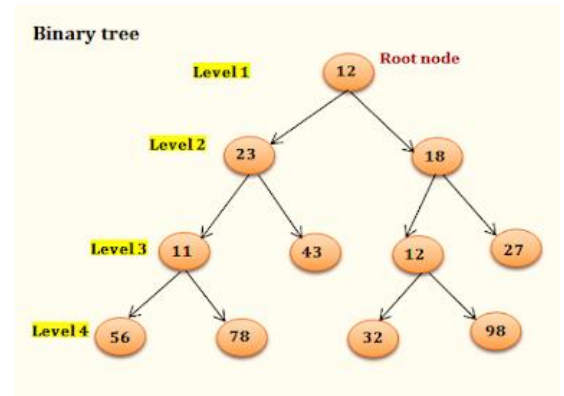


Figure 2

In this problem you are to complete the `ForestPlayGroundRevisited` class which implements the functionality of a rooted binary tree. You may implement this class in any manner, but the rooted binary tree and the algorithms described in this problem assume an array as data structure used to store the rooted binary tree.

The Level of a binary tree starts from root node with value 1. The root of a binary tree is the only node on level 1. Every time we move from top of the tree towards the bottom the level is increase by one. The important thing to remember is when talking about level, it starts from 1 and the level of the root is 1. For example, the rooted binary tree in Figure 2 shows level 1 through level 4



The `getLevel(int lev)` method returns an `int[]` containing every value on level `lev`. The values in the `int[]` being returned should be the same order as the binary tree. `null` values are NOT included in the `int[]` being returned.

The following table show sample results of the `getLevel` method.

The following code	Returns
<pre>int[] figure2 = {-1, 12, 23, 18, 11, 43, 12, 27, 56, 78, -1, -1, 32, 98}; ForestPlayGroundRevisited t = new ForestPlayGroundRevisited(figure2); int[] sol = t.getLevel(3);</pre>	
<code>sol.length;</code>	4
<code>sol[0];</code>	11
<code>sol[1];</code>	43
<code>sol[2];</code>	12
<code>sol[3];</code>	27

Here is another example using the same binary tree.

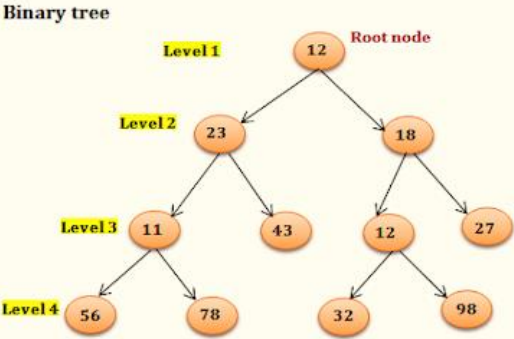
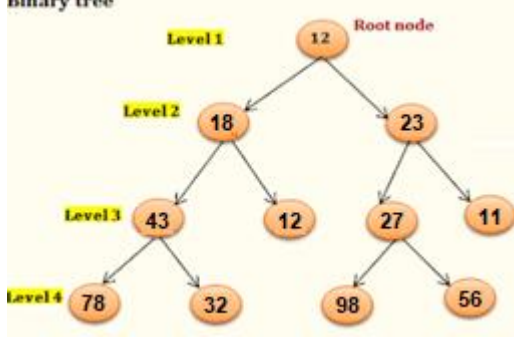
The following code	Returns
<pre>int[] figure2 = {-1, 12, 23, 18, 11, 43, 12, 27, 56, 78, -1, -1, 32, 98}; ForestPlayGroundRevisited t = new ForestPlayGroundRevisited(figure2); int[] sol = t.getLevel(4);</pre>	
<code>sol.length;</code>	4
<code>sol[0];</code>	56
<code>sol[1];</code>	78
<code>sol[2];</code>	32
<code>sol[3];</code>	98

The `getLevelWithHighestAverage()` returns the level number with the largest average. Remember to only include valid nodes in the level. That is, ignore the -1s.

The following table show sample results of the `getLevelWithHighestAverage` method.

The following code	Returns
<pre>int[] figure2 = {-1, 12, 23, 18, 11, 43, 12, 27, 56, 78, -1, -1, 32, 98}; ForestPlayGroundRevisited t = new ForestPlayGroundRevisited(figure2);</pre>	
<code>t.getLevelWithHighestAverage()</code>	4

The `shiftLevelLeft()` returns an `int[]` with all nodes on each level shifted one node to the left. Empty nodes should remain empty. That is, the nodes with a value -1 should remain in the same locations. Consider the following table with the given Binary tree both before and after the call `shiftLevelLeft`.

Binary tree before the call <code>shiftLevelLeft</code>	Binary tree after the call <code>shiftLevelLeft</code>
<p>Binary tree</p>  <pre> graph TD L1[12] --> L2L[23] L1 --> L2R[18] L2L --> L3L1[11] L2L --> L3L2[43] L2R --> L3R1[12] L2R --> L3R2[27] L3L1 --> L4L1[56] L3L1 --> L4L2[78] L3R1 --> L4R1[32] L3R1 --> L4R2[98] </pre>	<p>Binary tree</p>  <pre> graph TD L1[12] --> L2L[18] L1 --> L2R[23] L2L --> L3L1[43] L2L --> L3L2[12] L2R --> L3R1[27] L2R --> L3R2[11] L3L1 --> L4L1[78] L3L1 --> L4L2[32] L3R1 --> L4R1[98] L3R1 --> L4R2[56] </pre>

The following table show sample results of the `shiftLevelLeft` method.

The following code	Returns
<pre>int[] figure2 = {-1, 12, 23, 18, 11, 43, 12, 27, 56, 78, -1, -1, 32, 98}; ForestPlayGroundRevisited t = new ForestPlayGroundRevisited(figure2); sol = t.shiftLevelLeft();</pre>	
<code>sol[0]</code>	-1
<code>sol[1]</code>	12
<code>sol[2]</code>	18
<code>sol[3]</code>	23
<code>sol[4]</code>	43

The rest of the table continues on the next page

sol[5]	12
sol[6]	27
sol[7]	11
sol[8]	78
sol[9]	32
sol[10]	-1
sol[11]	-1
sol[12]	98
sol[13]	56