# **FAQs for Binary Trees and Parrots Spring 2020**

[Binary Tree Example 1](#_Toc6766486)

[FAQ 1: How do I know where to add a node/parrot/key in a binary tree? 1](#_Toc6766487)

[FAQ 2: How do I add a node/parrot to a binary tree in code? 1](#_Toc6766488)

[FAQ 3: What is level-order traversal? 2](#_Toc6766489)

[FAQ 4: How do I do a level-order traversal of a binary tree in code? 2](#_Toc6766490)

[FAQ 5: How exactly does the level-order traversal algorithm work. 3](#_Toc6766491)

### Binary Tree Example



### FAQ 1: How do I know where to add a node/parrot/key in a binary tree?

Use the binary search tree property: nodes with keys less than the current node go in its left sub-tree. Nodes with keys greater than the current node go in its right sub-tree.

Example: say you want to add a node with key 3 to the binary tree shown above:

* Start at the root node, its value is 15 and 3 is less than 15 so go left.
* This brings you to a node with key 8 and 3 is less than 8, so go left again.
* This brings you to a node with key 2 and 3 is greater than 2, so you want to go right.
  + The 2 node has NULL on its right branch – you’ve reached a leaf node.
  + This is the place to add the 3, on the right branch of the 2 node.

### FAQ 2: How do I add a node/parrot to a binary tree in code?

Your binary tree ***insert*** method should look similar to what was presented during the lecture.

**public** **boolean** insert(**Parrot** parrotToAdd) {

**if** root is null **then**

set root to new parrotToAdd

**else**

TreeNode parent = null;

TreeNode current = root;

**while** (haven’t reached bottom of the tree)

**//** *Where should we place the new parrot node?*

**if** (parrotToAdd compareTo current node parrot < 0) **then**

*// Go to the left*

Move parent and current each down one to the left

**else** **if**(parrotToAdd compareTo current node parrot > 0)**then**

*// Go to the right*

Move parent and current each down one to the right

**else**

*// Found a duplicate node - do not* insert

**return** **false**;

**end**

**end while**

**if** (parrotToAdd compareTo parent node parrot < 0) **then**

*// add parrot to left sub-tree*

**else**

*// add parrot to right sub-tree*

**end**

**end**

**return** **true**;

**end method**

### FAQ 3: What is level-order traversal?

It is when you read the values of the nodes left-to-right on each level, starting at the root.

For the example binary tree on page 1 (before adding the 3), the level order traversal is:

15 8 38 2 12 20 51 1 14 16 32 48 72 39

### FAQ 4: How do I do a level-order traversal of a binary tree in code?

Visit the nodes on each level of the tree from left to right starting at the root.

As you visit each node, you must keep track of the nodes on the next level underneath, so you can visit them when you are done with the current level. To do this use a queue of nodes, where as you visit each node, first add its **left child to the queue**, and then its **right child**. If the right or left is null, then there is nothing to add to the queue.

Your binary tree ***levelOrder*** method should look similar to code from lecture:

***public void levelOrder****()* {

**if** root is not null **then**

*// Load queue with the root*

Create a queue

Add root to the queue

**while** queue is not empty

*// Pull the next node off the queue, save in a temporary node*

TreeNode tempNode = queue.remove()

Make parrot in node sing, that is, print its word in the song

*// Add tempNode’s left and right children to the queue, left 1st*

**if** tempNode’s left is not null **then**

add the left child to the queue

**end**

**if** tempNode’s right is not null **then**

add the right child to the queue

**end**

**end** **while**

**end** **if**

**end method**

### FAQ 5: How exactly does the level-order traversal algorithm work.

Let’s apply the algorithm to a specific tree showing the queue for each iteration.

We’ll use this tree:



**Walking the above code using given tree**

|  |  |
| --- | --- |
| **Add root** | **15** |

|  |  |  |
| --- | --- | --- |
| **Remove 15 – print “Oh”** | **8** | **38** |

Add left child - 8

Add right child - 38

|  |  |  |  |
| --- | --- | --- | --- |
| **Remove 8 – print “beautiful”** | **38** | **2** | **12** |

Add left child - 2

Add right child - 12

|  |  |  |  |
| --- | --- | --- | --- |
| **Remove 38 – print “for”** | **2** | **12** | **51** |

Add left child - none

Add right child - 51

|  |  |  |
| --- | --- | --- |
| **Remove 2 – print “spacious”** | **12** | **51** |

Add left child – none

Add right child - none

|  |  |  |
| --- | --- | --- |
| **Remove 12 – print “skies”** | **51** | **14** |

Add left child – none

Add right child - 14

|  |  |  |
| --- | --- | --- |
| **Remove 51 – print “for”** | **14** | **48** |

Add left child – 48

Add right child – none

|  |  |
| --- | --- |
| **Remove 14 – print “amber”** | **48** |

Add left child – none

Add right child – none

|  |  |
| --- | --- |
| **Remove 48 – print “waves”** | **39** |

Add left child - 39

Add right child – none

|  |  |
| --- | --- |
| **Remove 39 – print “of grain”** |  |

Add left child - none

Add right child - none

**List of printed song words: Oh beautiful for spacious skies for amber waves of grain**