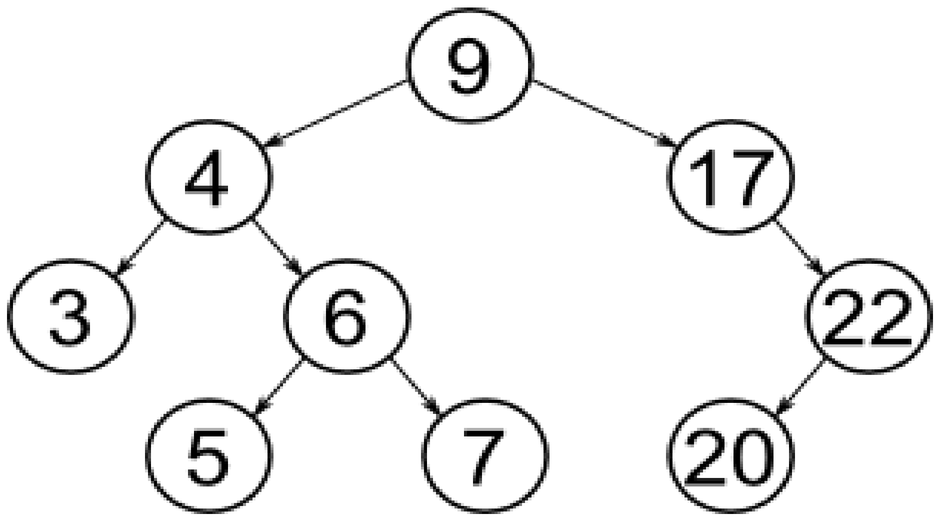


...here, we see the root node is **9**; its height is zero (h0).

...**4** and **17** are at height 1 (h1).

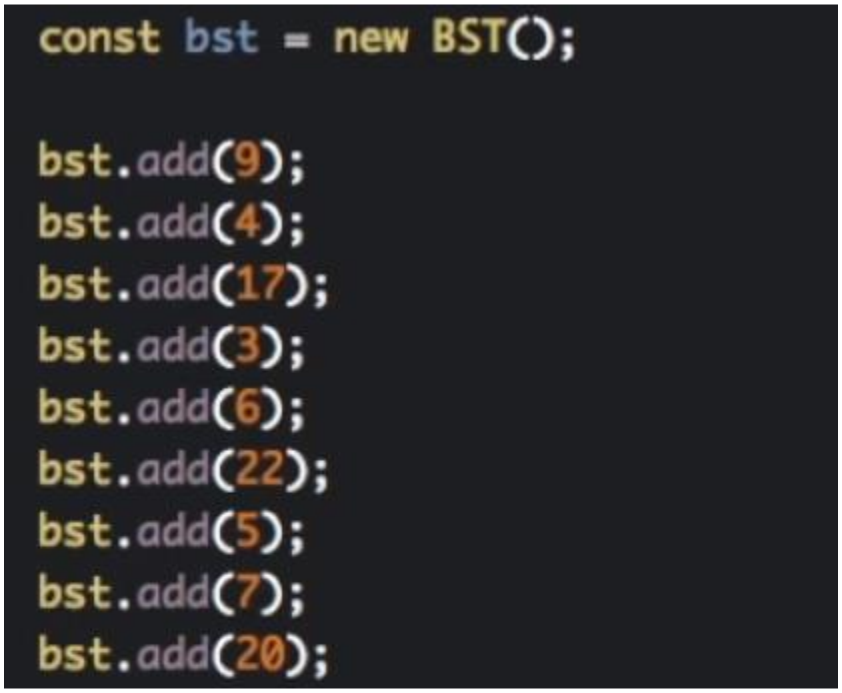
...**3, 6** and **22** are at height 2 (h2).

…the leaf nodes **5, 7** and **20** are at height 3 (h3).

-Maximum height: distance from the root node (**9**) to a leaf node (**5,7** or **20**).

-Minimum height: distance from the root node (**9**) to the first node with less than 2 children (**17**).

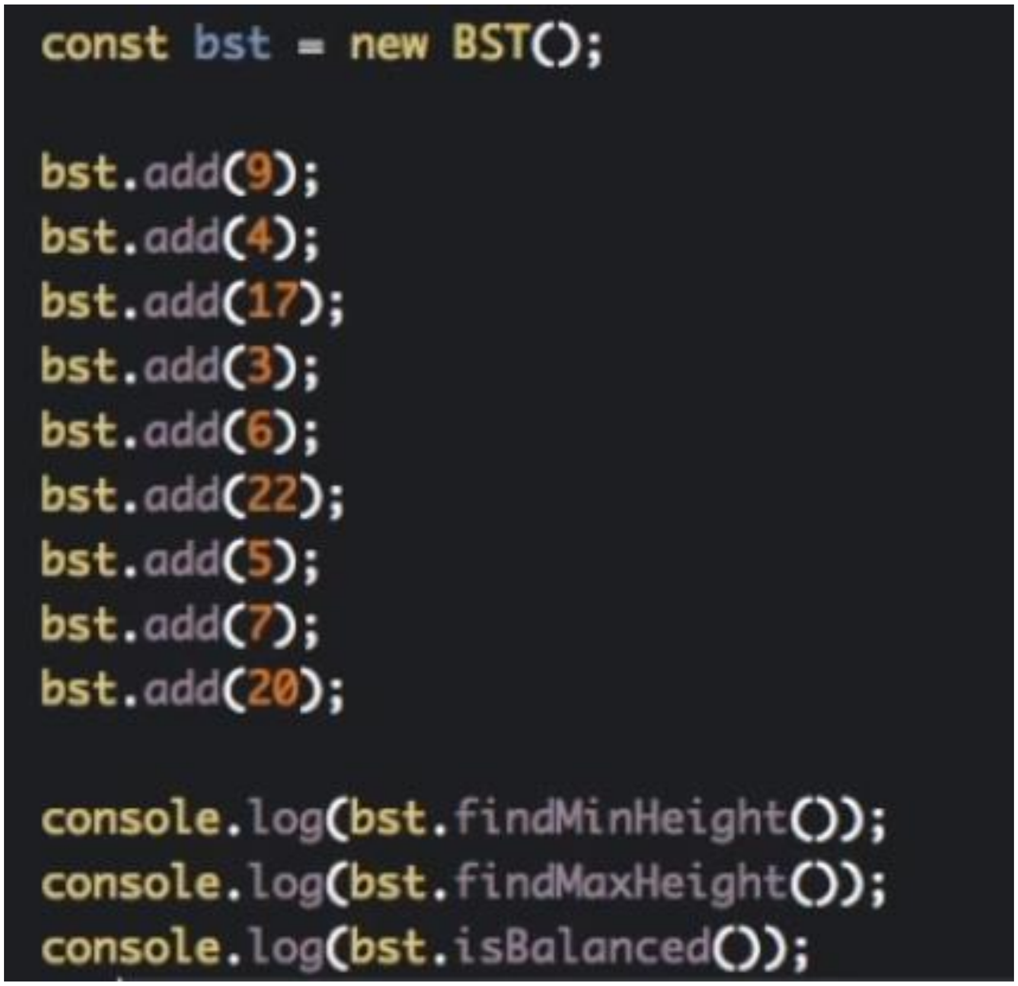
If we type out this binary tree into script:



```
const bst = new BST();

bst.add(9);
bst.add(4);
bst.add(17);
bst.add(3);
bst.add(6);
bst.add(22);
bst.add(5);
bst.add(7);
bst.add(20);
```

Let’s find the minimum height, maximum height, and see if the binary search tree is balanced:

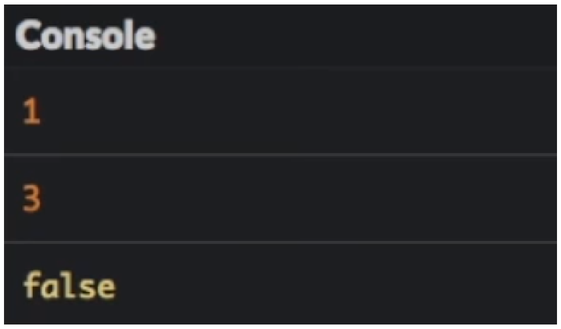


```
const bst = new BST();

bst.add(9);
bst.add(4);
bst.add(17);
bst.add(3);
bst.add(6);
bst.add(22);
bst.add(5);
bst.add(7);
bst.add(20);

console.log(bst.findMinHeight());
console.log(bst.findMaxHeight());
console.log(bst.isBalanced());
```

...the output in the console gives us:

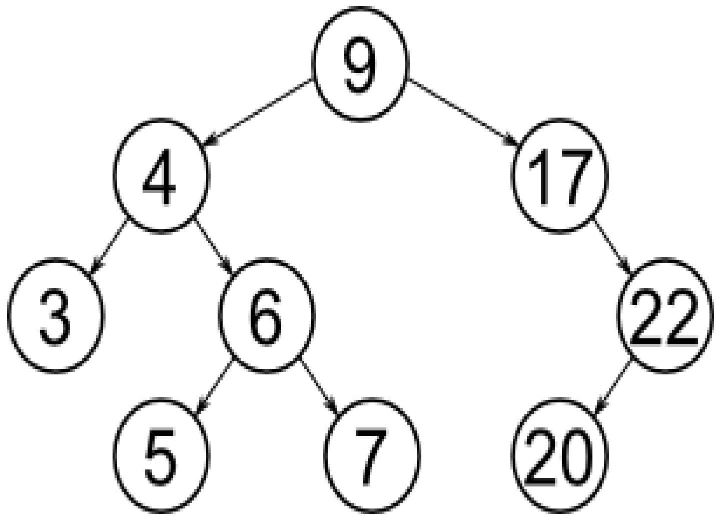


```
Console
1
3
false
```

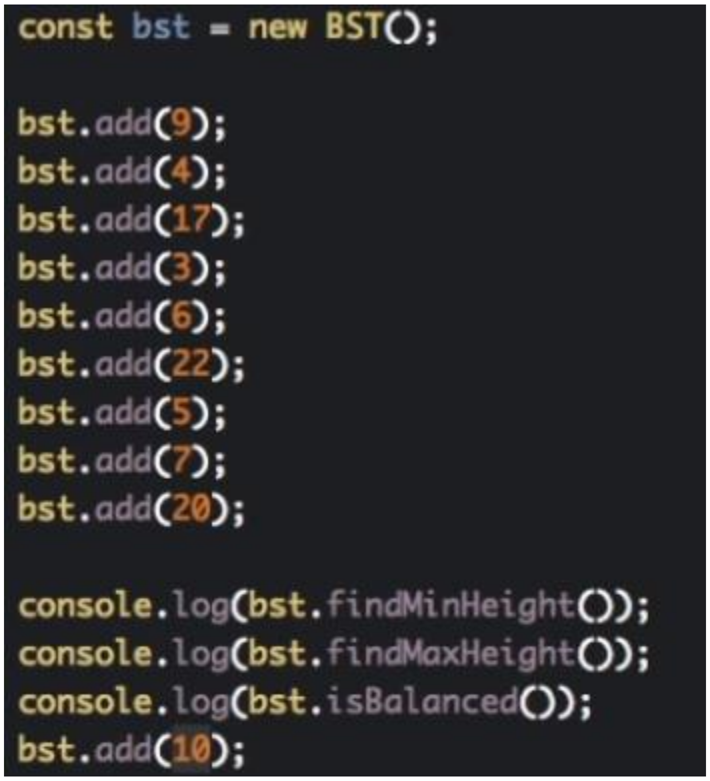
- The distance from **9** (h0) to **17** (h1) is 1.

- The distance from **9** (h0) to **20** (h3) is 3.

- The tree is not balanced, because **4** has 2 children, whereas 17 has 1 child.



Now if we add to to the tree:

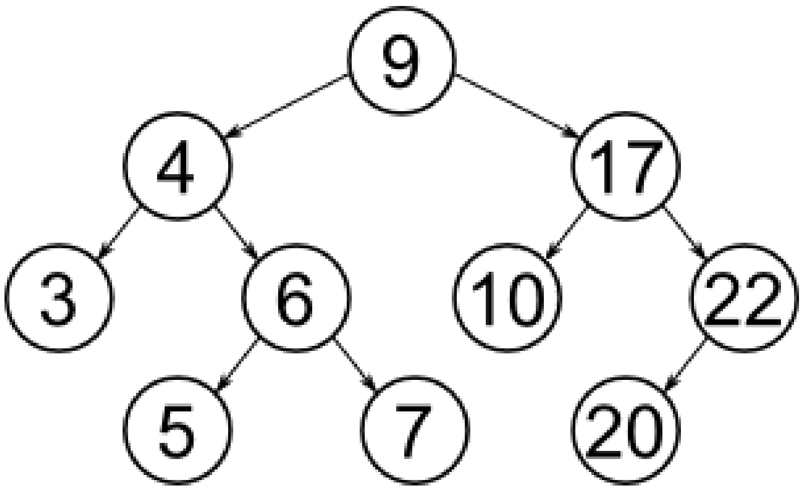


```
const bst = new BST();

bst.add(9);
bst.add(4);
bst.add(17);
bst.add(3);
bst.add(6);
bst.add(22);
bst.add(5);
bst.add(7);
bst.add(20);

console.log(bst.findMinHeight());
console.log(bst.findMaxHeight());
console.log(bst.isBalanced());
bst.add(10);
```

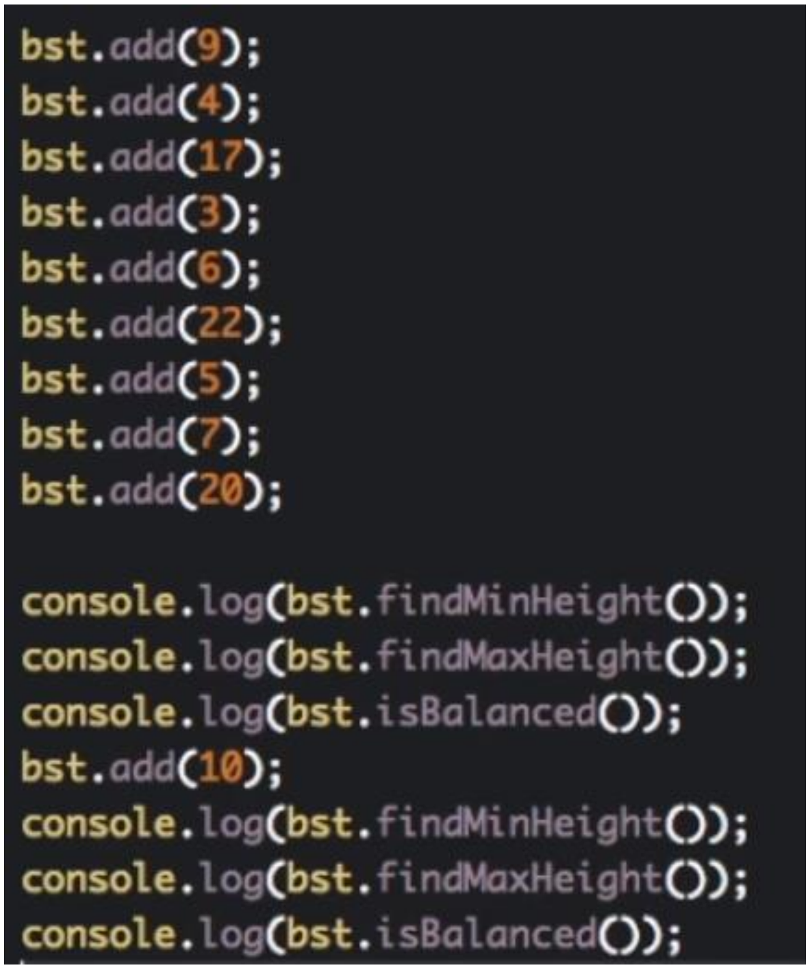
...this is what the tree looks like:



$$\begin{tikzcd}
 & & & 9 \arrow[dll] \arrow[drr] & & & \\
 & 4 \arrow[dl] \arrow[dr] & & & & 17 \arrow[dl] \arrow[dr] & \\
3 & & 6 \arrow[dl] \arrow[dr] & & 10 & & 22 \arrow[dl] \\
 & 5 & & 7 & & 20 &
\end{tikzcd}$$

...the reason 10 gets added to the left of 17 is because 10 is more than 9, but lless than 17.

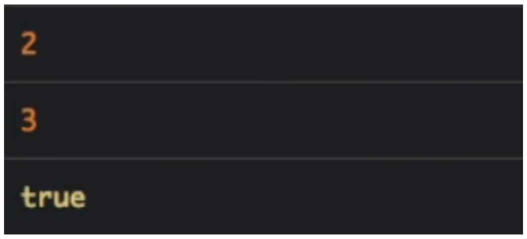
Now let's find the min, max and determine if it's balanced:



```
bst.add(9);
bst.add(4);
bst.add(17);
bst.add(3);
bst.add(6);
bst.add(22);
bst.add(5);
bst.add(7);
bst.add(20);

console.log(bst.findMinHeight());
console.log(bst.findMaxHeight());
console.log(bst.isBalanced());
bst.add(10);
console.log(bst.findMinHeight());
console.log(bst.findMaxHeight());
console.log(bst.isBalanced());
```

…the output is now:

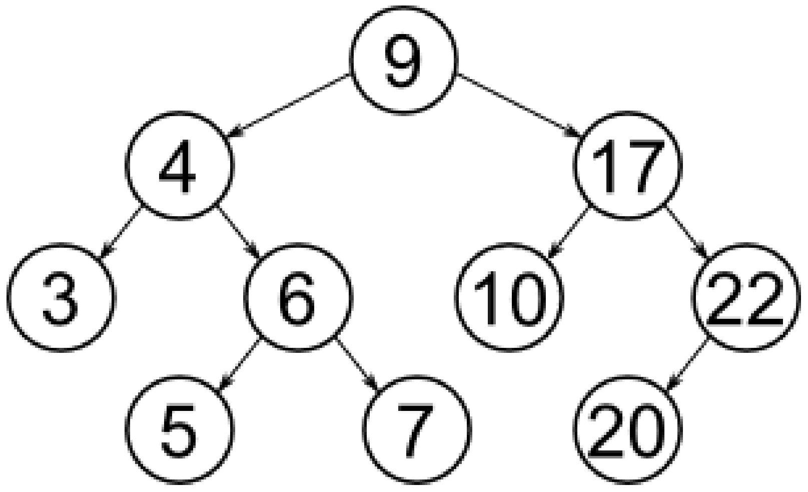


```
2
3
true
```

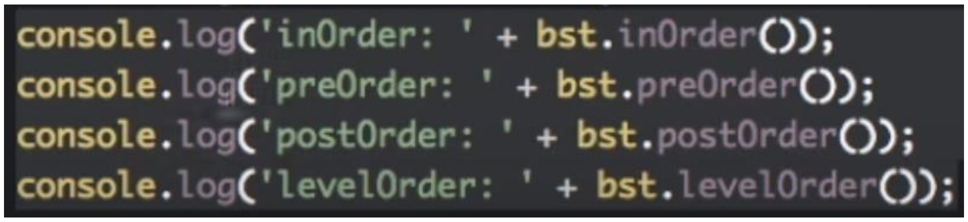
- The distance from **9** (h0) to **10** (h2) is 2.

- The distance from **9** (h0) to **20** (h3) is 3.

- The tree is balanced, because **4** has 2 children and **17** also has 2 children.

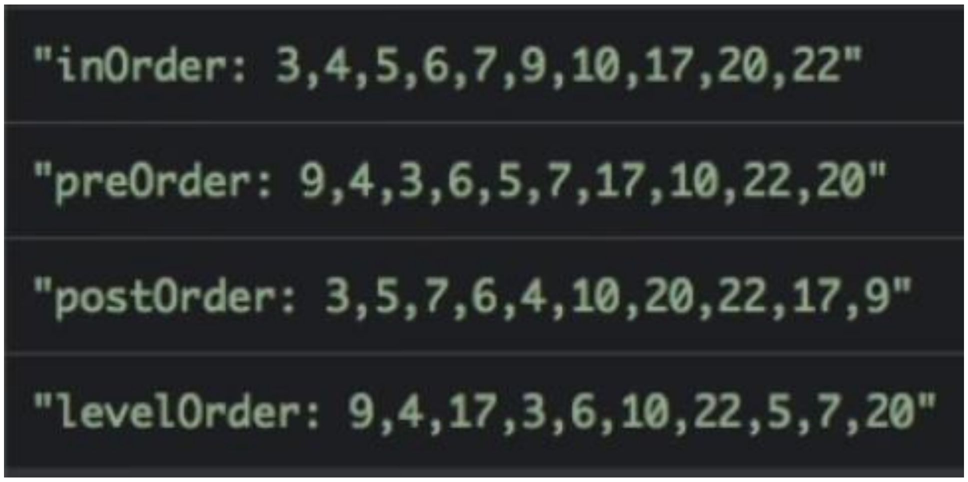


Let’s explore the various ways to traverse the tree:



```
console.log('inOrder: ' + bst.inOrder());
console.log('preOrder: ' + bst.preOrder());
console.log('postOrder: ' + bst.postOrder());
console.log('levelOrder: ' + bst.levelOrder());
```

...here are the outputs:



```
"inOrder: 3,4,5,6,7,9,10,17,20,22"

"preOrder: 9,4,3,6,5,7,17,10,22,20"

"postOrder: 3,5,7,6,4,10,20,22,17,9"

"levelOrder: 9,4,17,3,6,10,22,5,7,20"
```