Are duplicate keys allowed in the definition of binary search trees?

I'm trying to find the definition of a binary search tree and I keep finding different definitions everywhere.

Some say that for any given subtree the left child key is less than or equal to the root.

Some say that for any given subtree the right child key is greater than or equal to the root.

And my old college data structures book says "every element has a key and no two elements have the same key."

Is there a universal definition of a bst? Particularly in regards to what to do with trees with multiple instances of the same key.

EDIT: Maybe I was unclear, the definitions I'm seeing are

1) left <= root < right

2) left < root <= right

3) left < root < right, such that no duplicate keys exist.

data-structures computer-science binary-tree

edited Aug 17 at 18:01

laike9m
3,631 2 15 39

asked Nov 19 '08 at 3:45

Tim Merrifield
1,532 3 20 29

10 Answers

Many algorithms will specify that duplicates are excluded. For example, the example algorithms in the MIT Algorithms book usually present examples without duplicates. It is fairly trivial to implement duplicates (either as a list at the node, or in one particular direction.)

Most (that I've seen) specify left children as <= and right children as >. Practically speaking, a BST which allows either of the right or left children to be equal to the root node, will require extra computational steps to finish a search where duplicate nodes are allowed.

It is best to utilize a list at the node to store duplicates, as inserting an '=' value to one side of a node requires rewriting the tree on that side to place the node as the child, or the node is placed as a grand-child, at some point below, which eliminates some of the search efficiency.

You have to remember, most of the classroom examples are simplified to portray and deliver the concept. They aren't worth squat in many real-world situations. But the statement, "every element has a key and no two elements have the same key", is not violated by the use of a list at the element node.

So go with what your data structures book said!

Edit:

Universal Definition of a Binary Search Tree involves storing and search for a key based on traversing a data structure in one of two directions. In the pragmatic sense, that means if the value is <>, you traverse the data structure in one of two 'directions'. So, in that sense, duplicate values don't make any sense at all.

This is different from BSP, or binary search partition, but not all that different. The algorithm to search has one of two directions for 'travel', or it is done (successfully or not.) So I apologize that my original answer didn't address the concept of a 'universal definition', as duplicates are really a

distinct topic (something you deal with after a successful search, not as part of the binary search.)

edited Nov 19 '08 at 5:28

answered Nov 19 '08 at 4:08

Chris
2,750 11 13

What are the disadvantages of using a list at the node? – Pacerier Jun 12 '12 at 19:18

In a BST, all values descending on the left side of a node are less than (or equal to, see later) the node itself. Similarly, all values descending on the right side of a node are greater than (or equal to) the nodes value.

Some BSTs may choose to allow duplicate values, hence the "or equal to" qualifiers above.

The following example may clarify:

```
            |
      +--- 14 ---+
      |          |
+--- 13    +--- 22 ---+
|          |          |
1          16    +--- 29 ---+
               |          |
              28         29
```

This shows a BST that allows duplicates - you can see that to find a value, you start at the root node and go down the left or right subtree depending on whether your search value is less than or greater than the node value.

This can be done recursively with something like:

```
def hasVal (node, srchval):
    if node == NULL:
         return false
    if node.val == srchval:
        return true
    if node.val > srchval:
        return hasVal (node.left, srchval)
    return hasVal (node.right, srchval)
```

and calling it with:

```
foundIt = hasVal (rootNode, valToLookFor)
```

Duplicates add a little complexity since you may need to keep searching once you've found your value for other nodes of the same value.

edited Jan 18 '12 at 4:24

answered Nov 19 '08 at 4:04

paxdiablo
433k 105 836 1302

For the duplicate case, can you just check if the right child is the same as the current node in the node.val == srchval: clause, and then if so go right? – bneil Feb 11 '13 at 16:43

If your binary search tree is a red black tree, or you intend to any kind of "tree rotation" operations, duplicate nodes will cause problems. Imagine your tree rule is this:

left < root <= right

Now imagine a simple tree whose root is 5, left child is nil, and right child is 5. If you do a left rotation on the root you end up with a 5 in the left child and a 5 in the root with the right child being nil. Now something in the left tree is equal to the root, but your rule above assumed left < root.

I spent hours trying to figure out why my red/black trees would occasionally traverse out of order, the problem was what I described above. Hopefully somebody reads this and saves themselves

hours of debugging in the future!

answered Dec 4 '08 at 4:27

Rich
3,085 3 31 59

So what's the solution? – Pacerier Jun 12 '12 at 19:29

8 Don't rotate when you have equal nodes! Traverse down to the next level and rotate that. – Rich Jun 19 '12 at 15:51

All three definitions are acceptable and correct. They define different variations of a BST.

Your college data structure's book failed to clarify that its definition was not the only possible.

Certainly, allowing duplicates adds complexity. If you use the definition "left <= root < right" and you have a tree like:

```
    3
  /   \
2       4
```

then adding a "3" duplicate key to this tree will result in:

```
    3
  /   \
2       4
  \
   3
```

Note that the duplicates are not in contiguous levels.

This is a big issue when allowing duplicates in a BST representation as the one above: duplicates may be separated by any number of levels, so checking for duplicate's existence is not that simple as just checking for immediate childs of a node.

An option to avoid this issue is to not represent duplicates structurally (as separate nodes) but instead use a counter that counts the number of occurrences of the key. The previous example would then have a tree like:

```
    3(1)
  /     \
2(1)    4(1)
```

and after insertion of the duplicate "3" key it will become:

```
    3(2)
  /     \
2(1)    4(1)
```

This simplifies lookup, removal and insertion operations, at the expense of some extra bytes and counter operations.

answered Dec 6 '13 at 14:32

duilio
51 1 2

Working on a red-black tree implementation I was getting problems validating the tree with multiple keys until I realized that with the red-black insert rotation, you have to loosen the constraint to

```
left <= root <= right
```

Since none of the documentation I was looking at allowed for duplicate keys and I didn't want to rewrite the rotation methods to account for it, I just decided to modify my nodes to allow for multiple values within the node, and no duplicate keys in the tree.

answered Apr 11 '11 at 20:56

Jherico
16k 4 34 64

Any definition is valid. As long as you are consistent in your implementation (always put equal nodes to the right, always put them to the left, or never allow them) then you're fine. I think it is most common to not allow them, but it is still a BST if they are allowed and place either left or right.

answered Nov 19 '08 at 3:47

SoapBox
15.3k 1 28 66

if you have a set of data which contains duplicate keys, then those items should all be stored within the 1 node on the tree via a different method (linked list, etc). the tree should only contain unique keys. – nickf Nov 19 '08 at 3:55

Should perhaps, but it is not required for the properties of a search tree. – SoapBox Nov 19 '08 at 3:57

#1 condition of a BST: "each node (item in the tree) has a distinct value;" en.wikipedia.org/wiki/Binary_search_tree – nickf Nov 19 '08 at 3:58

Wikipedia isn't the end all source of knowledge. Explain to me what is invalid about having duplicate keys in a BST other than wasting space... it maintains all important properties. – SoapBox Nov 19 '08 at 4:01

1 Also note from the wiki that the right subtree contains values "greater than or equal to" the root. Hence the wiki definition is self-contradictory. – SoapBox Nov 19 '08 at 4:02

Those three things you said are all true.

- Keys are unique
- To the left are keys less than this one
- To the right are keys greater than this one

I suppose you could reverse your tree and put the smaller keys on the right, but really the "left" and "right" concept is just that: a visual concept to help us think about a data structure which doesn't really have a left or right, so it doesn't really matter.

answered Nov 19 '08 at 3:52

nickf
242k 126 485 604

Duplicate Keys • What happens if there's more than one data item with the same key? – This presents a slight problem in red-black trees. – It's important that nodes with the same key are distributed on both sides of other nodes with the same key. – That is, if keys arrive in the order 50, 50, 50, • you want the second 50 to go to the right of the first one, and the third 50 to go to the left of the first one. • Otherwise, the tree becomes unbalanced. • This could be handled by some kind of randomizing process in the insertion algorithm. – However, the search process then becomes more complicated if all items with the same key must be found. • It's simpler to outlaw items with the same key. – In this discussion we'll assume duplicates aren't allowed

One can create a linked list for each node of the tree that contains duplicate keys and store data in the list.

answered Jul 29 '13 at 22:17

mszlazak
11 3

> 1.) left <= root < right
>
> 2.) left < root <= right
>
> 3.) left < root < right, such that no duplicate keys exist.

I might have to go and dig out my algorithm books, but off the top of my head (3) is the canonical form.

(1) or (2) only come about when you start to allow duplicates nodes *and* you put duplicate nodes in the tree itself (rather than the node containing a list).

answered Nov 19 '08 at 5:22 community wiki Robert Paulson

The elements ordering relation <= is a total order so the relation must be reflexive but commonly a binary search tree (aka BST) is a tree without duplicates.

Otherwise if there are duplicates you need run twice or more the same function of deletion!

answered Feb 13 '12 at 18:56

tyranitar

835 12 33