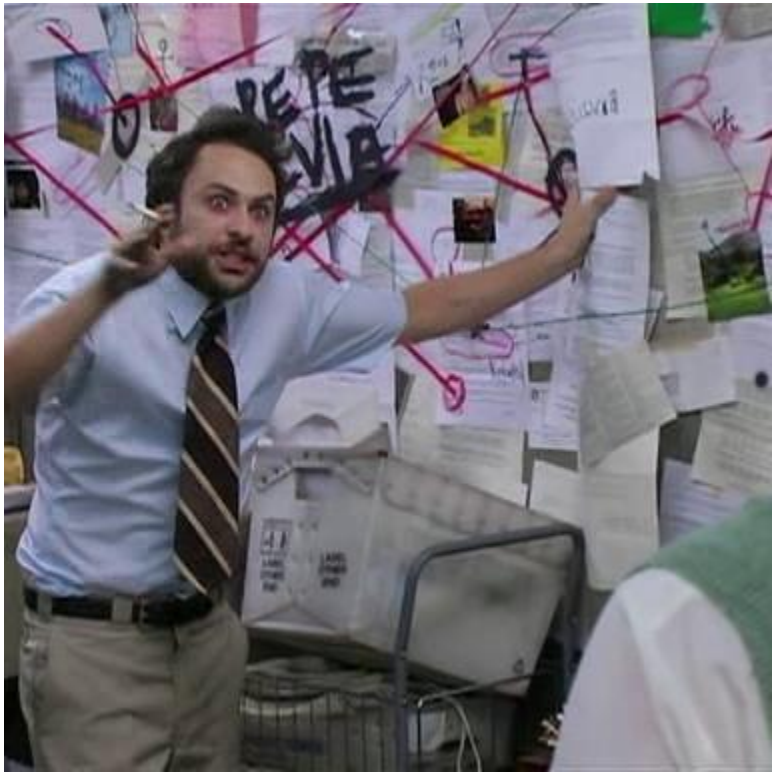
Traversing Trees the Hard Way

The Robson Traversal

By: Davis Silverman

How I felt learning how this algorithm works



Why not just do it the easy way?

- As Robson puts it, when using a stack, the size could be just as large as the tree!
  - AKA the space complexity for a basic Depth-first search is O(n).
- AKA, it may be memory inefficient
- The easy way is useful for understanding how traversals work, and for quick and dirty setups.
- When you need to traverse large trees, you need to break out the big guns.

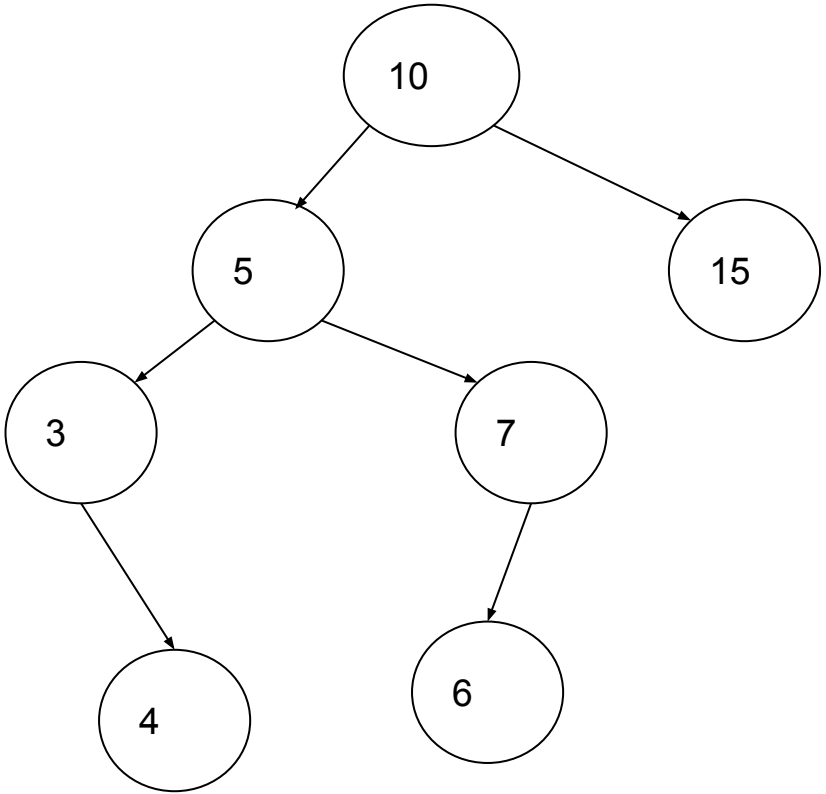
A Mixture of what we have learned

- Robson uses inverted links, similar to the link-inversion method
- It also uses the leaf-pointers, much like the threaded-tree model!
- However, instead of using an extra bit to detect if we should traverse right or up, we use the leaf pointers!
- If a leaf-pointer is pointing to the current node, and both subtrees are non-null, then we know the ascent was from the left, so go right.
- The leaf-pointers act as a pseudo stack!

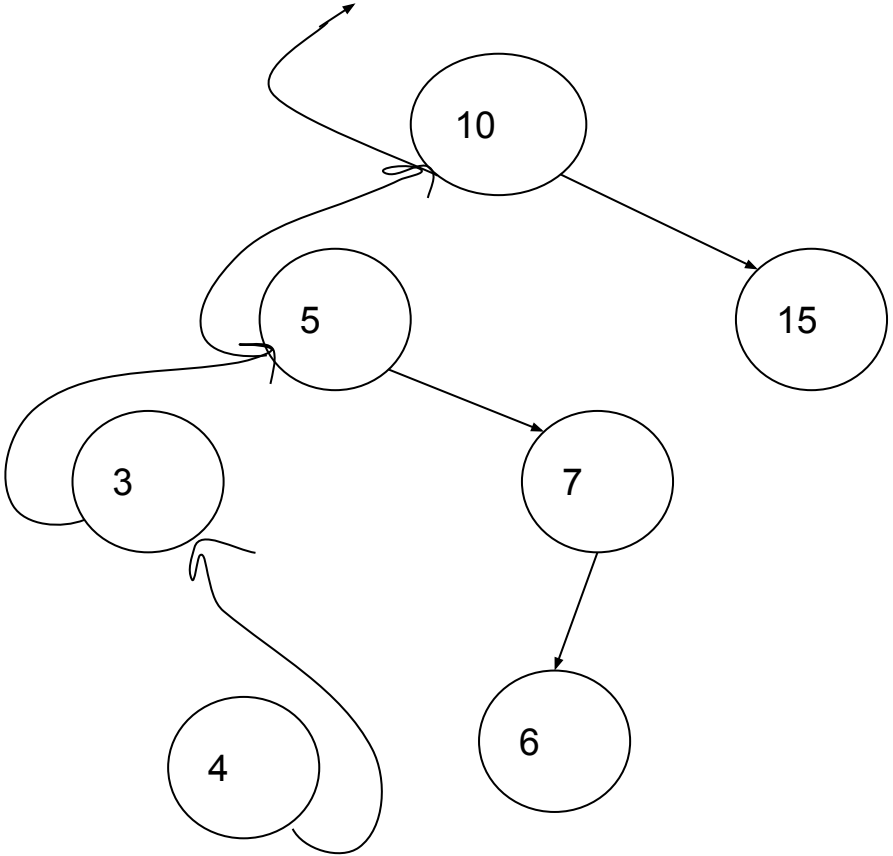
The key to Robson

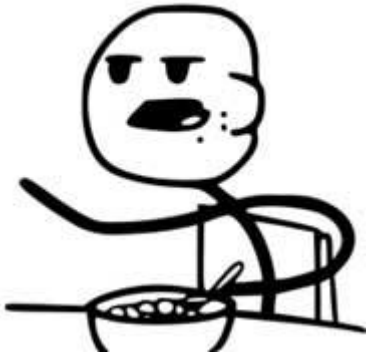
- Careful understanding of what each stacks job is.
- The leaf stack tells you to traverse right or up
- The inner stack (of inverted links) are for traversal and discovery
- The leaf stack is used rarely

Lets run through this algorithm!

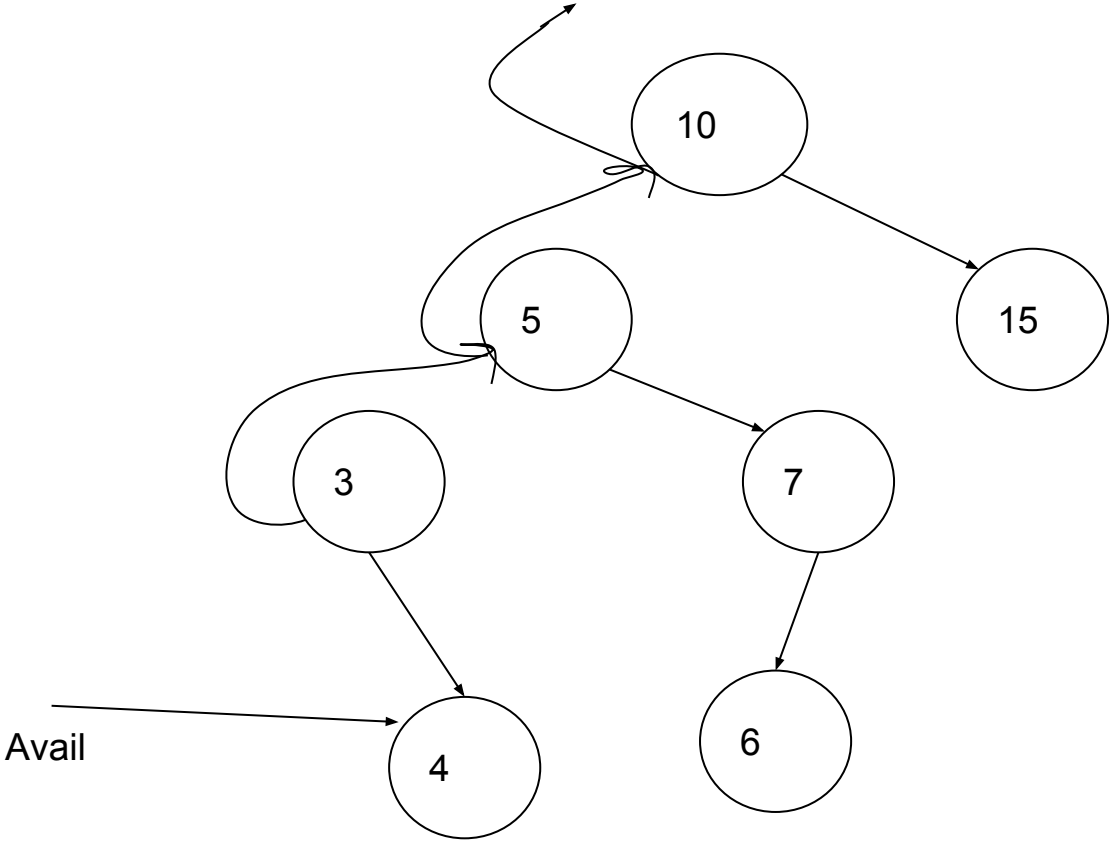


Step 1: Link Inversion!

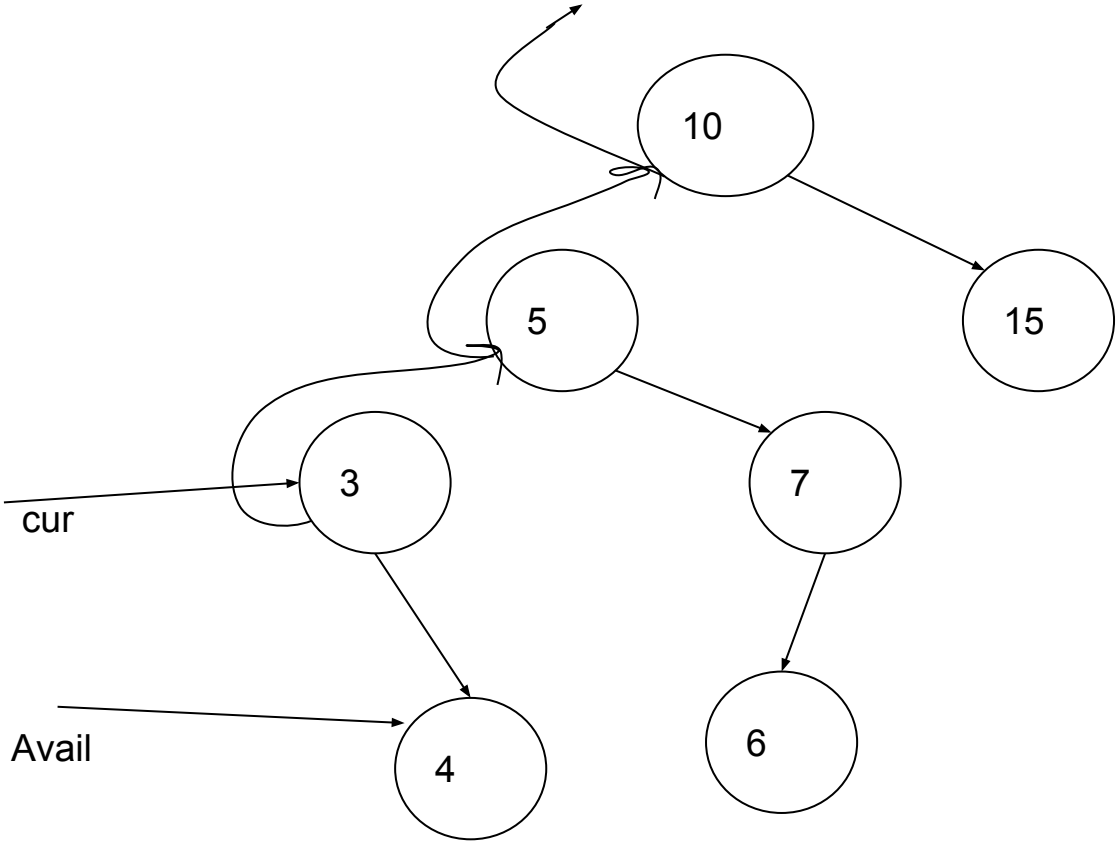


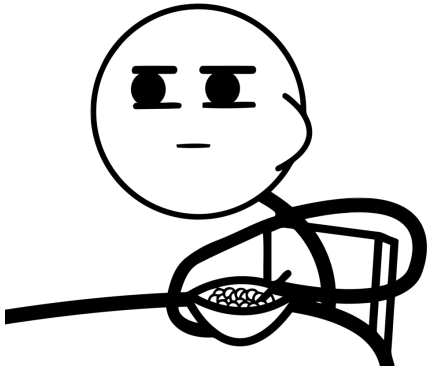


Step 2: Found Leaf, mark it available

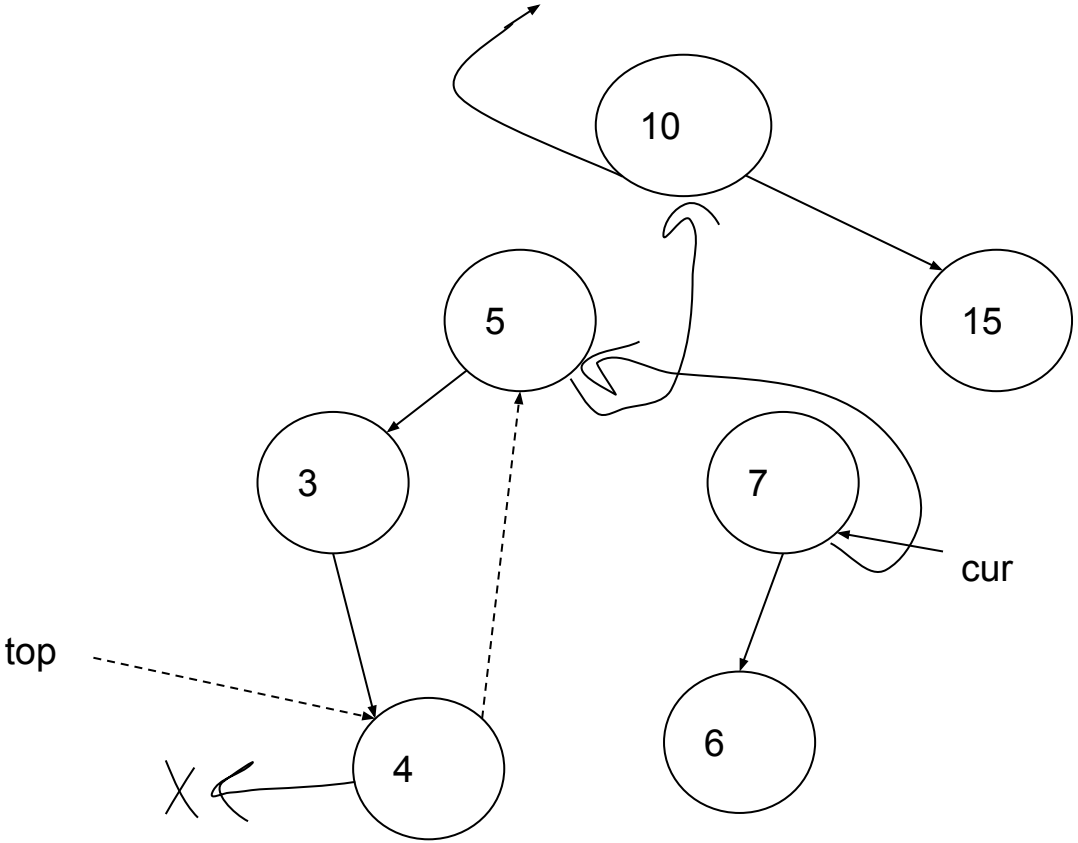


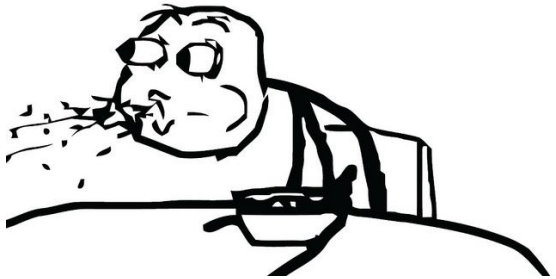
Step 3: Exchange Time!



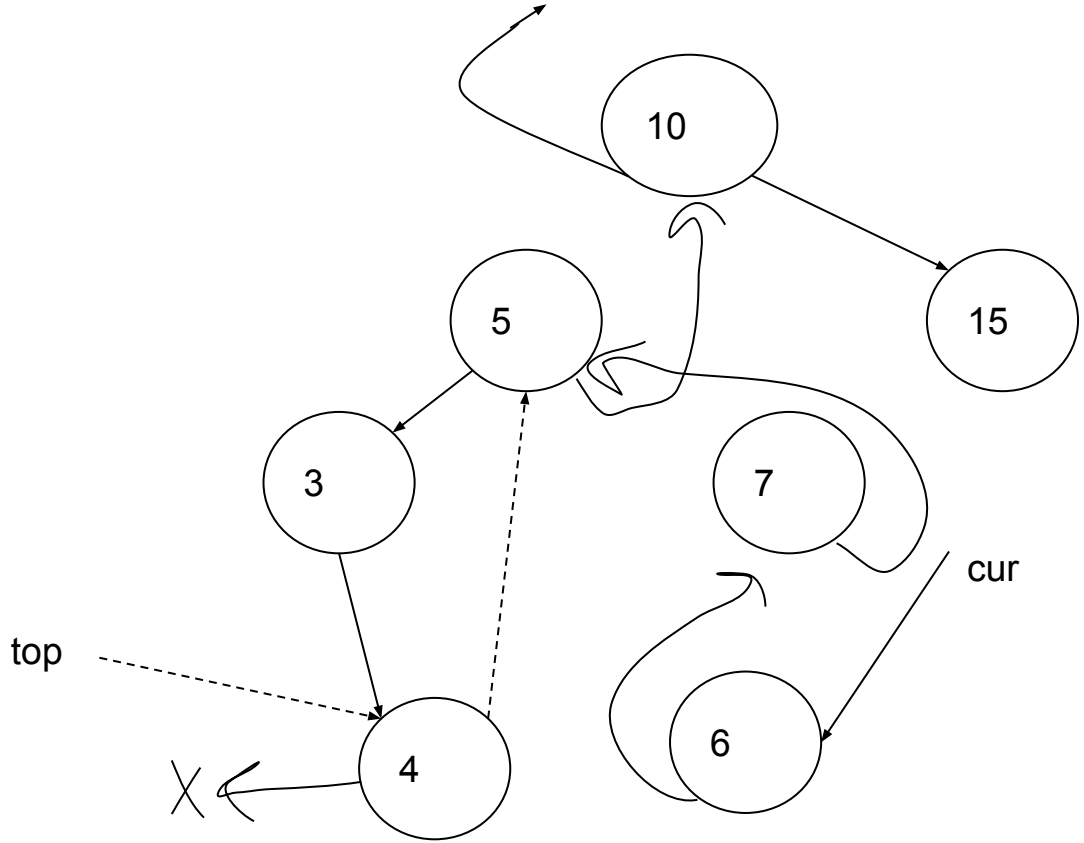


Step 3: Exchange Time!

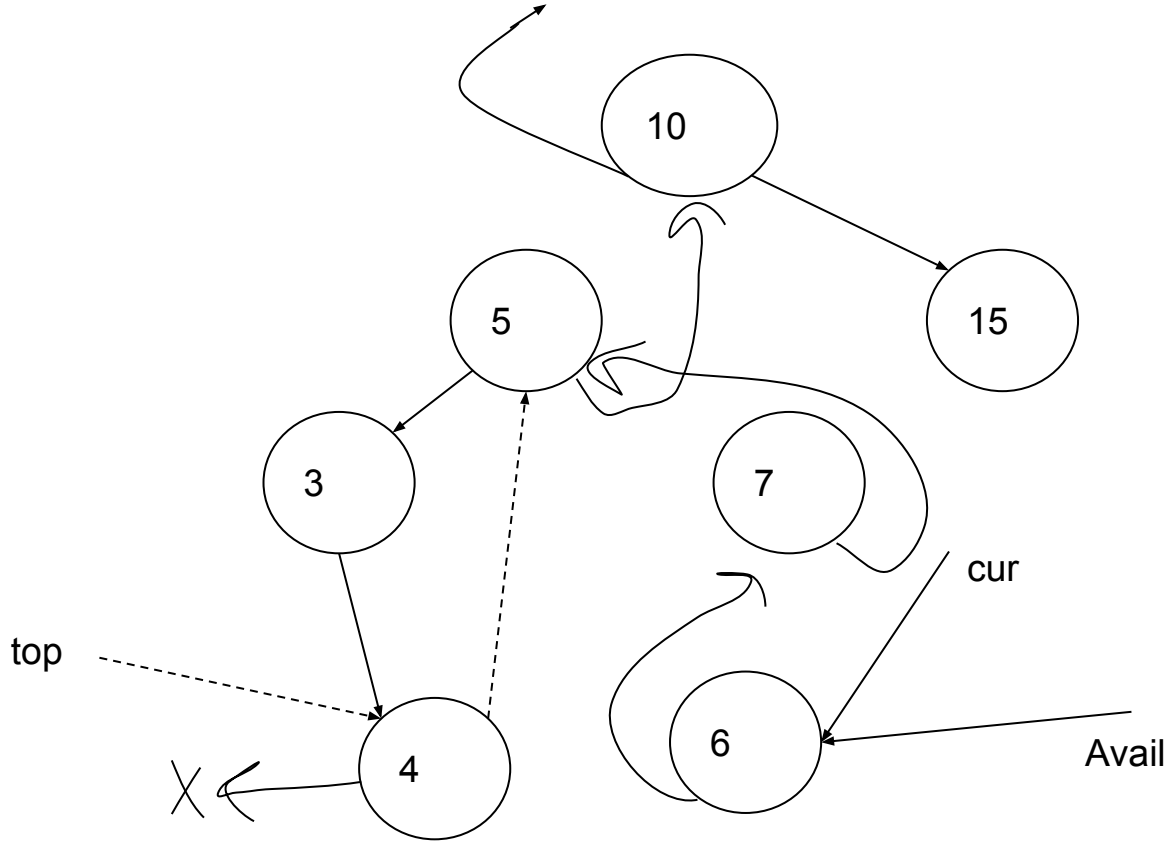




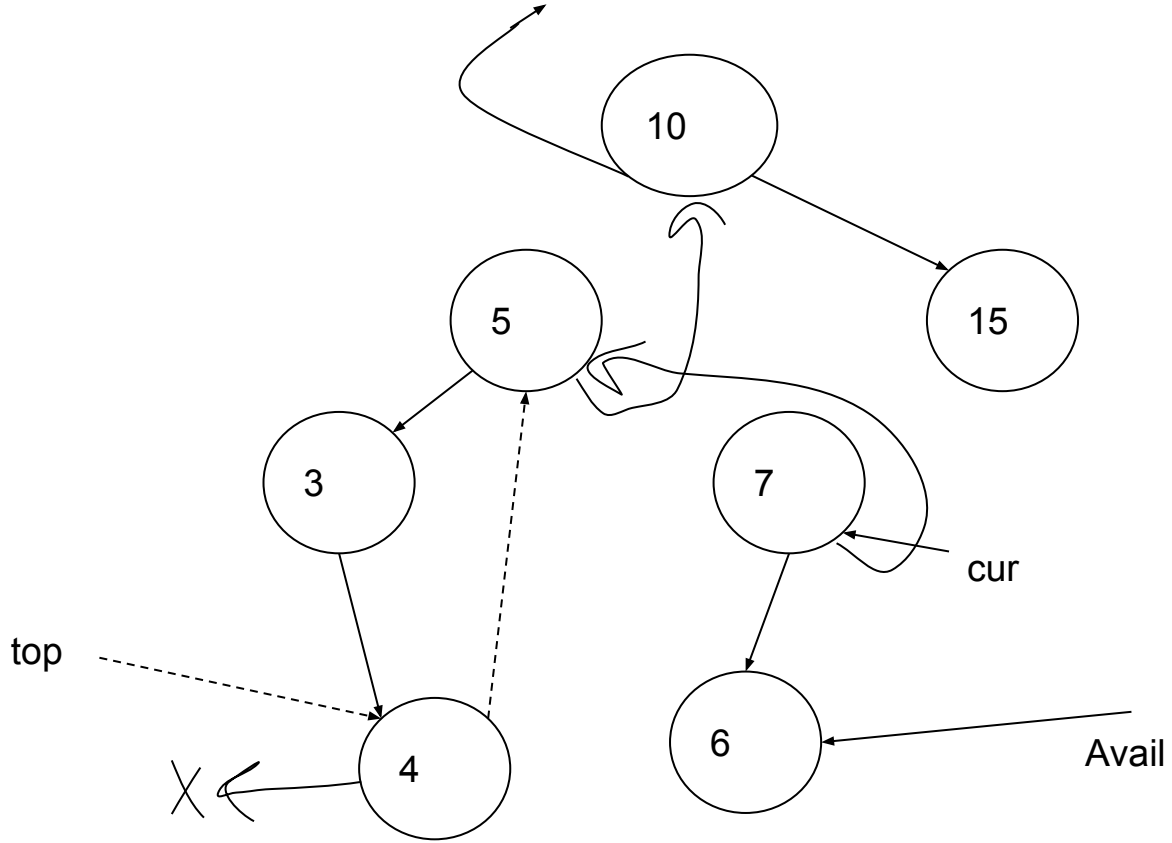
Step 4: Back to Link Inversion!



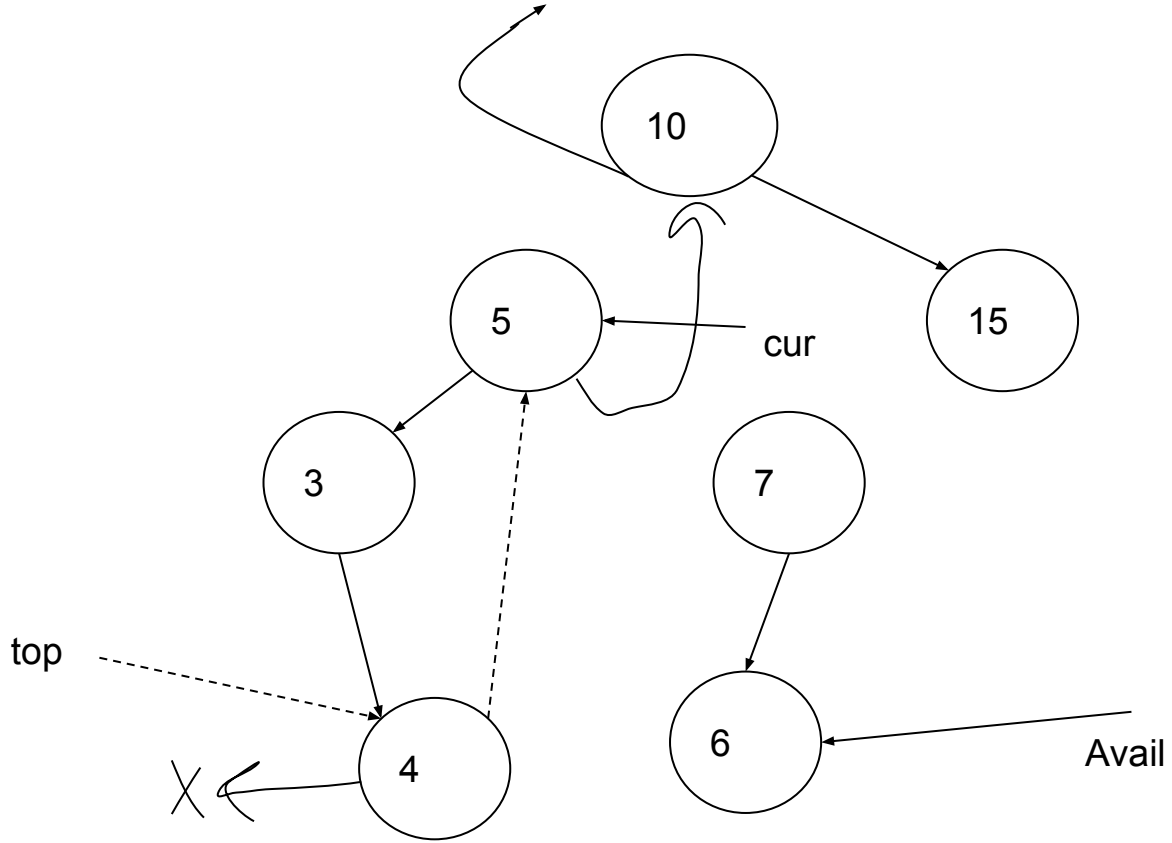
Step 5: Another leaf!



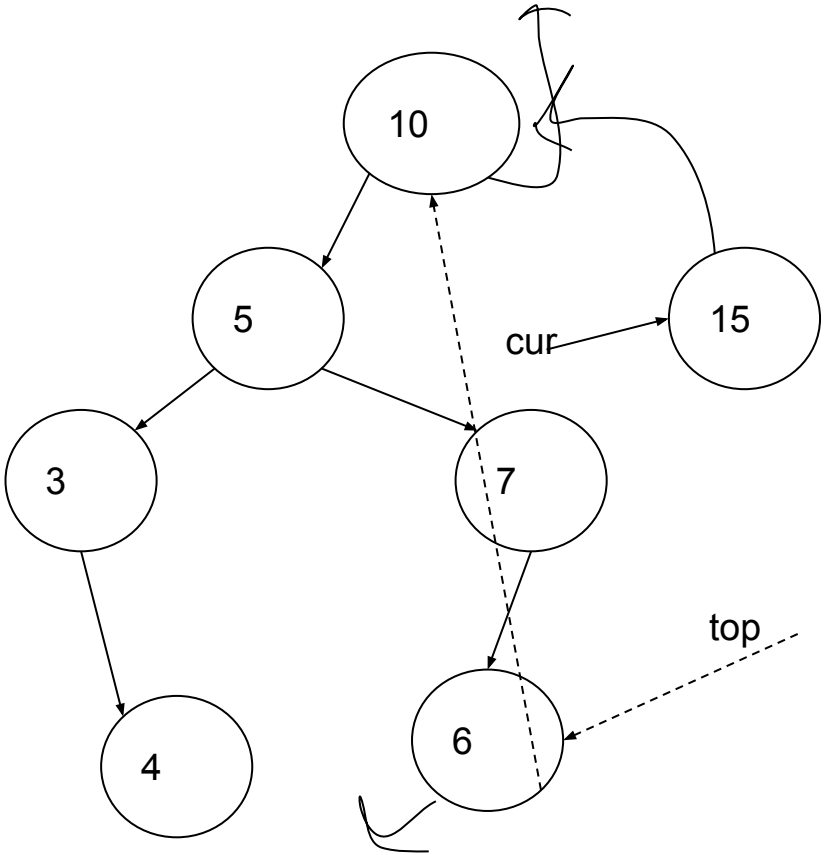
Step 6: Unwinding the inner stack again.



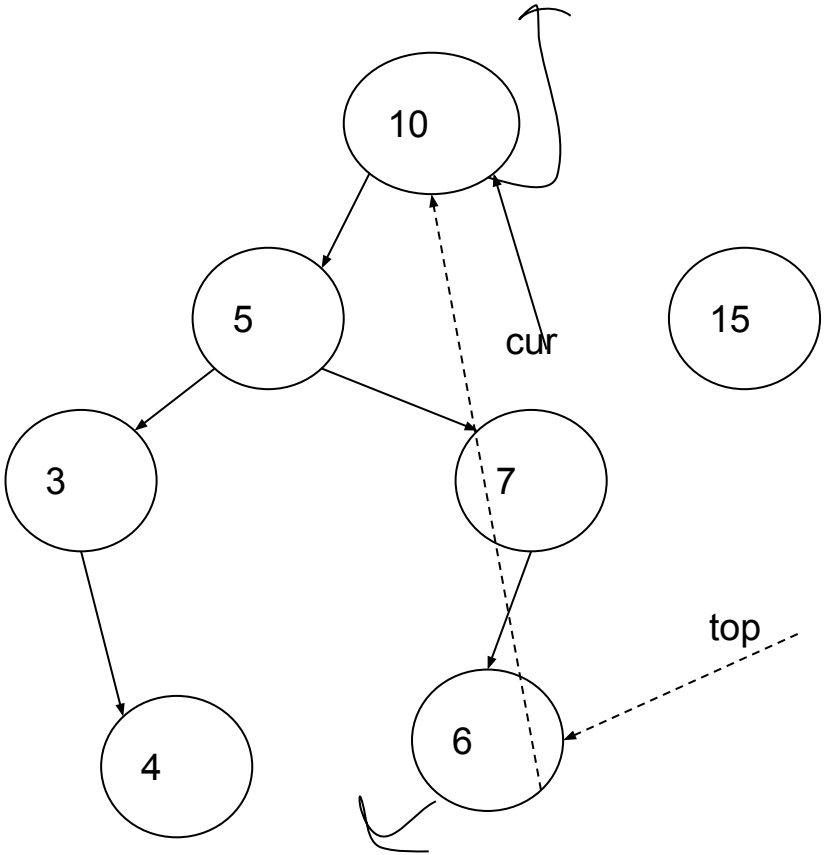
Step 7: Arrived at Leaf Stack Top!



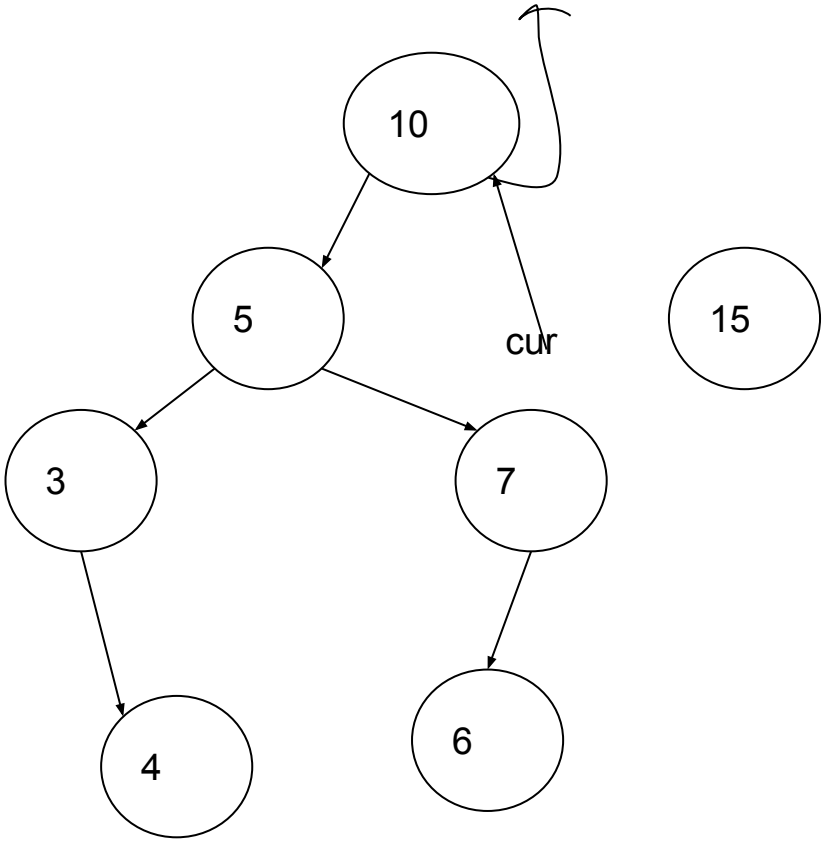
Step 8: Another Exchange



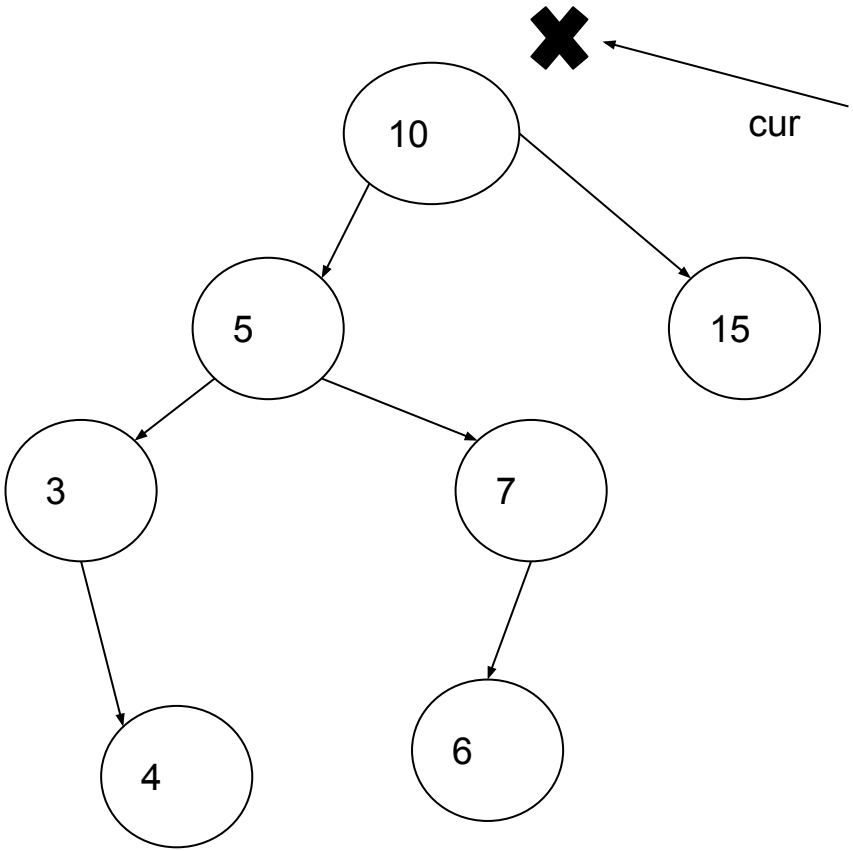
Step 9: Walking Up...



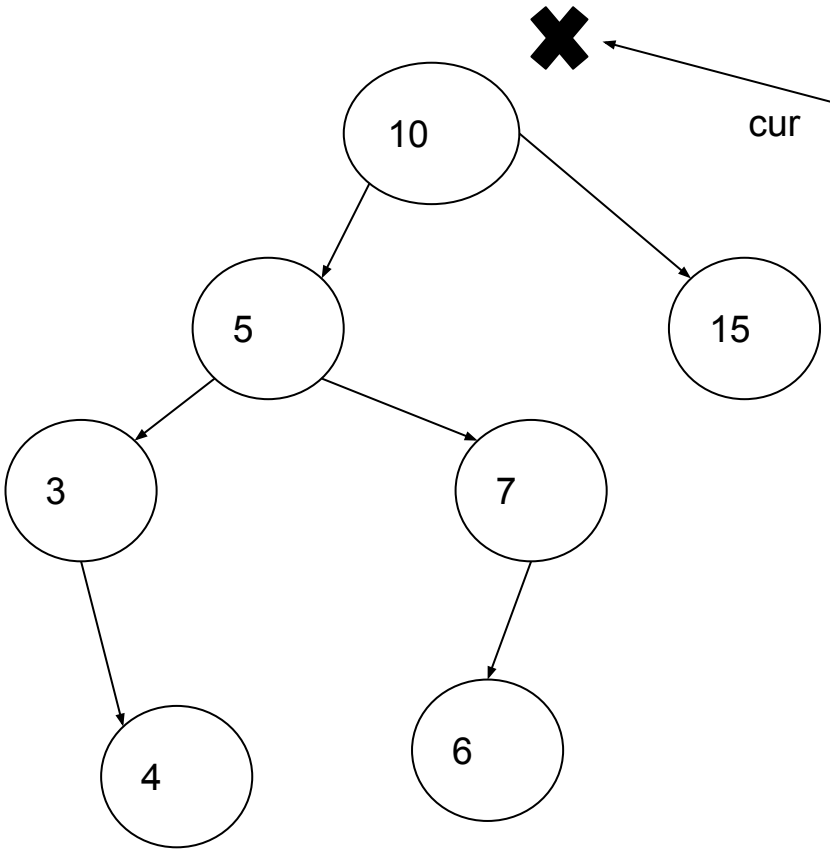
Step 10: Walking Up...



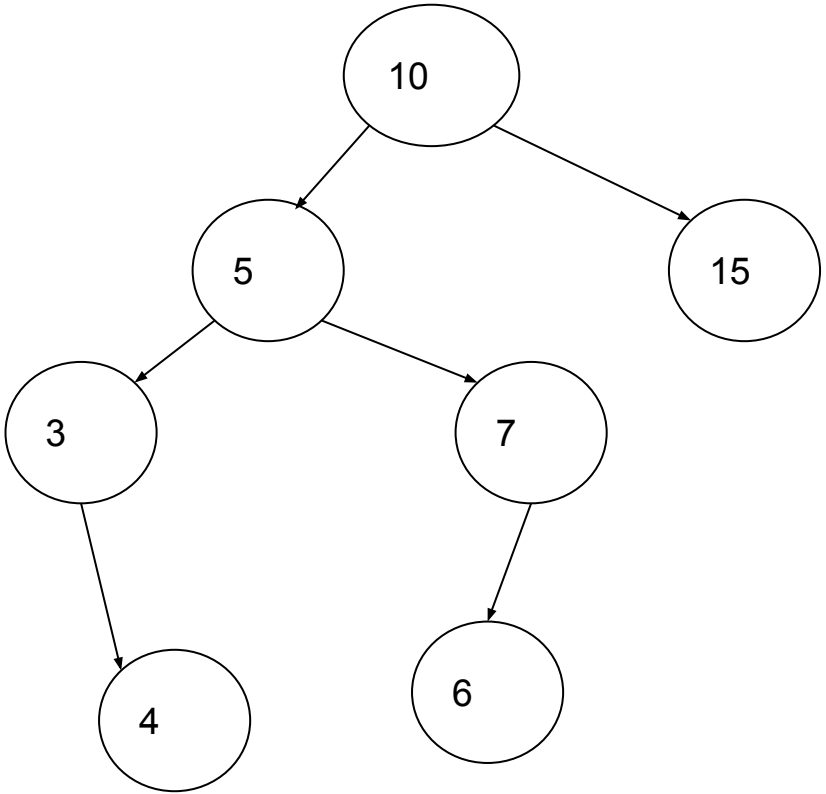
Step 10: Walking Up...



Step 11: cur == null → end



All clean again!



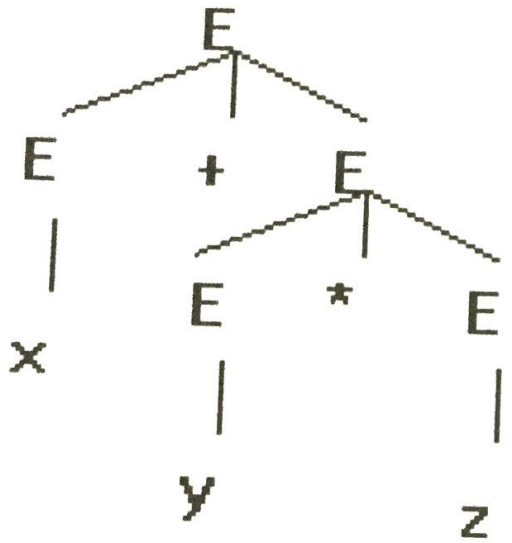
There you have it!

A traversed Binary Tree, in O(n) time, and O(1) space!

Great at parties!

When you can NOT use this traversal.

- When leaves point to things!
  - Expression trees! Where nodes are operators and leaves are variables or constants
  - Think 330 parse trees!



When you can NOT use this traversal.

- To find a single successor!
  - A threaded tree can find a successor in amortized O(1) time, but the Robson and link-inversion methods have no such ability.
  - Robson must complete a whole traversal, and can not be stopped midway through like a threaded tree.
  - Link-inversion and Robson traversals are ‘read-only’, and must be completed before being able to write to the tree again.
  - My professor (Jason Filippou) puts it much better:

[Robson] mutates the tree, which means that if the running thread is interrupted by a signal or an exception or whatever, unless the code implements a handler that immediately fixes the tree all the way to the root before exiting, the tree will be left at an unsafe state. In fact, entire subtrees are in danger of being garbage collected, or are simply left as memory leaks in cases where Garbage Collection is not implemented (e.g basic C/C++).

Sources

- J.M. Robson, An improved algorithm for traversing binary trees without auxiliary stack, Info. Process. Lett. 2 (1973) 12-14.
- D.E. Knuth, Fundamental Algorithms (Addison--Wesley, 1968). p. 562

Perlis and Thornton originally founded Threaded Trees, but I couldn't find a paper/source