



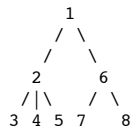
## Tree Traversals

A `_traversal_` is a manner of `_visiting_` each node in a tree once. What you do when visiting any particular node depends on the application; for instance, you might print a node's value, or perform some calculation upon it. There are several different traversals, each of which orders the nodes differently.

Many traversals can be defined recursively. In a `_preorder_` traversal, you visit each node before recursively visiting its children, which are visited from left to right. The root is visited first.

```
class SibTreeNode {
    public void preorder() {
        this.visit();
        if (firstChild != null) {
            firstChild.preorder();
        }
        if (nextSibling != null) {
            nextSibling.preorder();
        }
    }
}
```

Suppose your method `visit()` numbers the nodes in the order they're visited. A preorder traversal visits the nodes in this order.



Each node is visited only once, so a preorder traversal takes  $O(n)$  time, where  $n$  is the number of nodes in the tree. All the traversals we will consider take  $O(n)$  time.

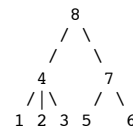
A preorder traversal is a natural way to print a directory's structure. Simply have the method `visit()` print each node of the tree.

```
~jrs/61b
hw
  hw1
  hw2
index.html
lab
  lab1
  lab2
lec
  01
  02
  03
  04
  05
```

In a `_postorder_` traversal, you visit each node's children (in left-to-right order) before the node itself.

```
public void postorder() {
    if (firstChild != null) {
        firstChild.postorder();
    }
    this.visit();
    if (nextSibling != null) {
        nextSibling.postorder();
    }
}
```

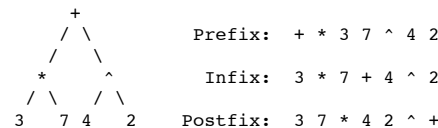
A postorder traversal visits the nodes in this order.



The `postorder()` code is trickier than it looks. The best way to understand it is to draw a depth-two tree on paper, then pretend you're the computer and execute the algorithm carefully. Trust me on this. It's worth your time.

A postorder traversal is the natural way to sum the total disk space used in the root directory and its descendants. The method `visit()` sums "this" node's disk space with the disk space of all its children. In the example above, a postorder traversal would first sum the sizes of the files in `hw1/` and `hw2/`; then it would visit `hw/` and sum its two children. The last thing it would compute is the total disk space at the root `~jrs/61b/`, which sums all the files in the tree.

Binary trees allow for an `_inorder_` traversal: recursively traverse the root's left subtree (rooted at the left child), then the root itself, then the root's right subtree. The preorder, inorder, and postorder traversals of an expression tree will print a `_prefix_`, `_infix_`, or `_postfix_` expression, respectively.



In a `_level-order_` traversal, you visit the root, then all the depth-1 nodes (from left to right), then all the depth-2 nodes, et cetera. The level-order traversal of our expression tree is "+ \* ^ 3 7 4 2" (which doesn't mean much).

Unlike the three previous traversals, a level-order traversal is not straightforward to define recursively. However, a level-order traversal can be done in  $O(n)$  time. Use a queue, which initially contains only the root. Then repeat the following steps:

- Dequeue a node.
- Visit it.
- Enqueue its children (in order from left to right).

Continue until the queue is empty.

A final thought: if you use a stack instead of a queue, and push each node's children in reverse order--from right to left (so they pop off the stack in order from left to right)--you perform a preorder traversal. Think about why.