

CS 228: Introduction to Data Structures

Lecture 29

Monday, November 7, 2016

From Postfix to Expression Trees

We can build the expression tree for a postfix expression using a stack (the code is in the `ExpressionTree` class posted on Blackboard).

Algorithm

Scan the expression character by character from left to right. For each character `c`:

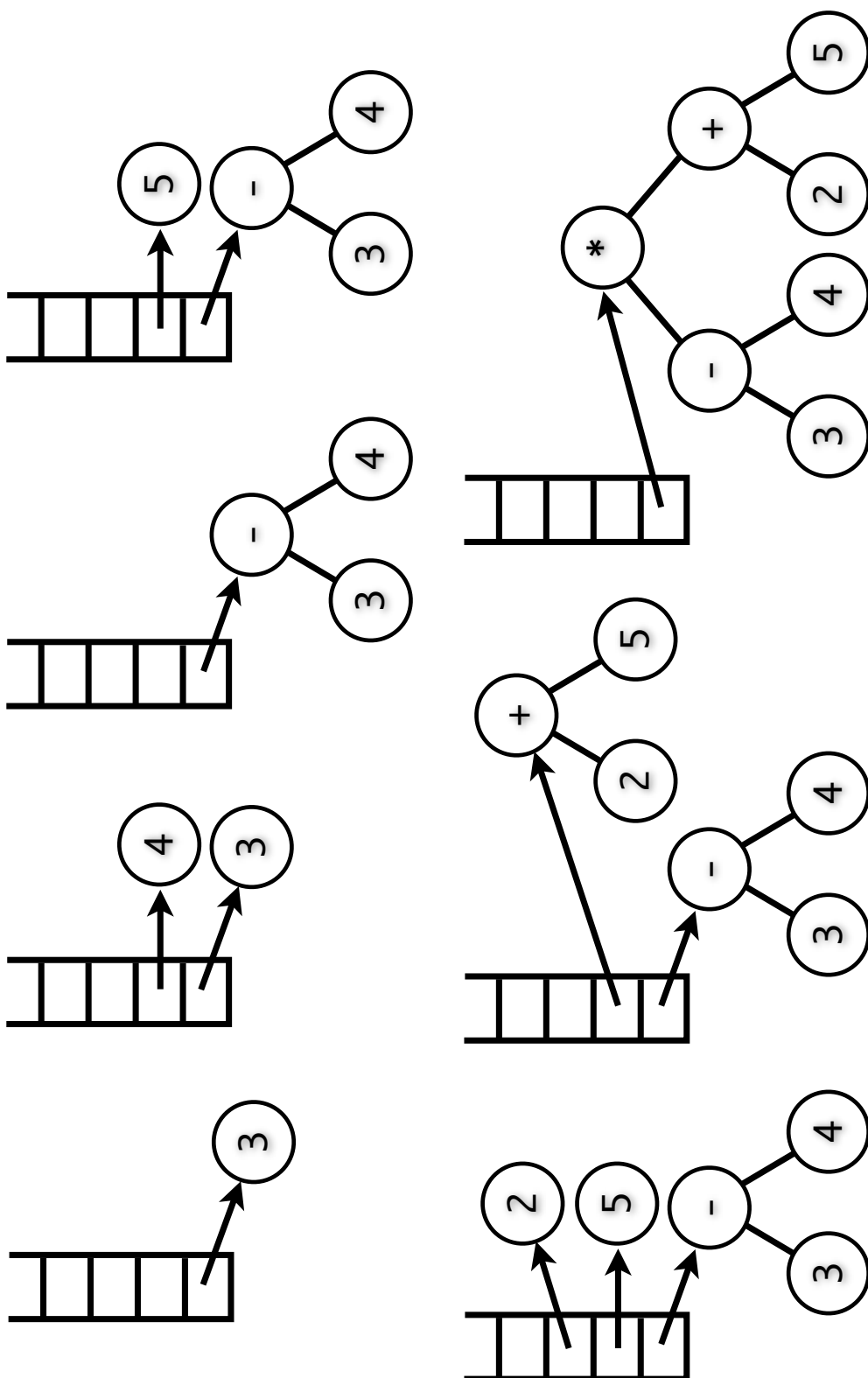
1. If `c` is a blank, skip it.
2. If `c` is a digit, create a node for it and push it on the stack.
3. If `c` is an operator,
 - 3.1. create a node `n` for `c`,
 - 3.2. pop the top of stack and make it `n`'s right subtree,
 - 3.3. pop again, and make it `n`'s left subtree.

At the end, the stack contains a reference to the root of the expression tree.

The `makeFromPostfix()` method in `ExpressionTree` implements this algorithm.

Example. The next page shows how the algorithm handles the expression

3 4 - 5 2 + *



Sets

In mathematics, a **set** is a collection of elements with no duplicates. The Java Set interface, which extends Collection, models this mathematical abstraction. It defines methods for the standard set operations, such as containment, union, and set difference. Our implementations will build on Java's AbstractSet, an abstract class that provides a partial implementation of the Set class. AbstractSet extends AbstractCollection, except that all of the methods and constructors in subclasses of this class must obey the additional constraints imposed by the Set interface; for instance, the add() method must not permit addition of multiple instances of an object to a set.

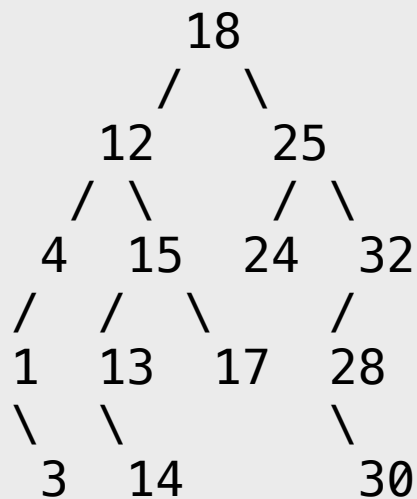
Binary Search Trees

We first consider sets where the element type is Comparable. Thus, the elements can be ordered so that there is a minimum element, a maximum element, and any element — other than the min and max — has a successor (next) and a predecessor (previous). In sets like this, the elements are often called **keys**.

A **binary search tree** (BST) is one way to implement a set of comparable elements. A BST is a binary tree whose nodes hold the keys; the keys must satisfy the following.

Binary Search Tree Property: For any node X , every key in the left subtree of X is less than X 's key, and every key in the right subtree of X is greater than X 's key.

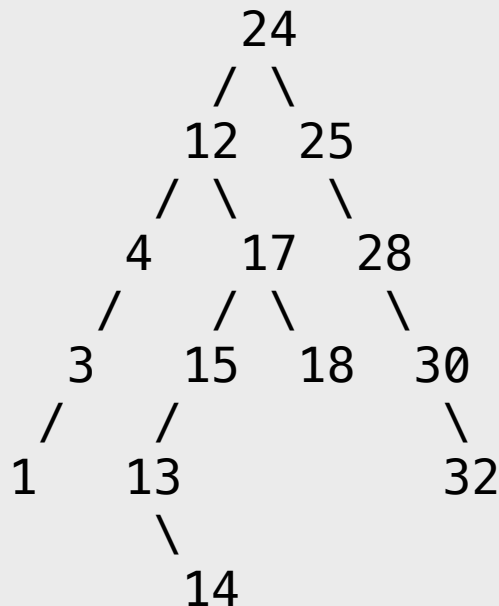
Example. Consider the BST below.



For instance, the root is 18, its left subtree (rooted at 12) contains numbers from 1 to 17, and its right subtree (rooted at 25) contains numbers from 24 to 30.

When a node has only one child, that child is either a left child or a right child, depending on whether its key is smaller or larger than its parent's key.

Note. The BST representation of a set is not unique. Here's another way to represent the set of keys from the previous example:



You should be able to verify the following:

Fact. An inorder traversal of a binary search tree visits the nodes in sorted order.

In this sense, a search tree maintains a sorted list of entries. However, operations on a binary search tree are usually much faster than the same operations on a sorted linked list because you typically only traverse a small fraction of the nodes of a tree. Before explaining why, we need to provide a few implementation details.

Representing a Set as a BST

Our implementation of sets via BSTs builds on Java's `AbstractSet`, an abstract class that extends `AbstractCollection`, providing a partial implementation of the `Set` class. The details of the tree representation are hidden within the `Node` inner class. A `Node` is like `TreeNode`, except that, in addition to references to the left and right children, it has a parent reference, which is used to implement removal and iteration efficiently.

```
public class
BSTSet<E extends Comparable<? super E>>
extends AbstractSet<E>
{
    protected Node root;
    protected int size;

    protected class Node
    {
        public Node left;
        public Node right;
        public Node parent;
        public E data;
    }
}
```

```
public Node(E key, Node parent)
{
    this.data = key;
    this.parent = parent;
}
}
```