5 Binary Search Trees; Graphics and Interactions

Binary Search Trees

We have the following data definition:

A *Binary Search Tree* is one of

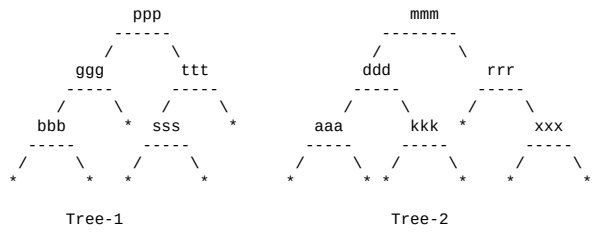
- *Empty Tree*
- a *Node*

A *Node* consists of

- *data* of the type String,
- *left Binary Search Tree*,
- *right Binary Search Tree*.

Additionally, every *Node* has the property that all data in the *left* subtree contains strings that appear in the dictionary before the data in the *Node*, and all data in the *right* subtree are strings that appear in the dictionary after the *data* in this node. The string that is identical to the field *data* can be in either the *left* or in the *right* subtree — it is not important.

The following are examples of such trees:



Your task is to design classes that represent this data as Java classes, and to design several methods to manipulate this data. You will also need an auxilliary classes that represent a list of Strings. You may read more about binary search tree in HtDP.

5.1 Problem

Design the classes that represent binary search trees of Strings. Make examples of data. ∎

5.2 Problem

Design the method *same* that determines whether two binary search trees are the same, i.e. they have the same structure and contain the same Strings. ∎

5.3 Problem

Design the method *insert* that insert a String into a binary search tree, preserving the tree property stated earlier.

Java provides the following method for String comparison:

```
// compare this String with that String lexicographically
// return <0  --- if 'this' is before 'that'
// return 0   --- if 'this' is the same String as 'that'
// return >0  --- if 'this' is after 'that'
int compareTo(String that) ...
```

If you do not understand how to do it, try some examples by hand, or read about the problem in HtDP. ∎

5.4 Problem

Design the classes that represent a list of Strings. Add the method *sort* that sorts the list in lexicographical order. Add the method *same* that determines whether two lists contain the same Strings in the same order. You should just modify your solutions to the previous homework. ∎

5.5 Problem

Design the method *inorder* that produces a list of Strings that appear in the nodes of this tree, ordered so that all strings in the left subtree appear in the list before the data in the root node, and all strings in the right subtree appear in the list after the data in the root node.

For example, our two examples would produce the lists in the following order:

```
Tree-1:  bbb ggg ppp sss ttt
Tree-2:  aaa ddd kkk mmm rrr xxx
```

The method consumes a list of Strings, initially empty, that represents the list of strings that come after all the nodes in this subtree have been entered

into the list. So, for our `Tree-2`, in the *Node* `ddd`, the given list of strings will contain (`mmm rrr xxx`). The nodes in this subtree, `ddd`, `aaa`, and `kkk`, still have to be added to the list. It is clear, that when we start at the node `mmm`, there is nothing in the list.

Again, make examples until you understand the problem. Follow the design recipe! ∎

5.6 Problem

Design the method *contains* for both, the classes that represent the binary search trees, and the classes that represent lists of *Strings*. The method determines whether the given String appears in the tree or list repsectively. ∎

5.7 Problem

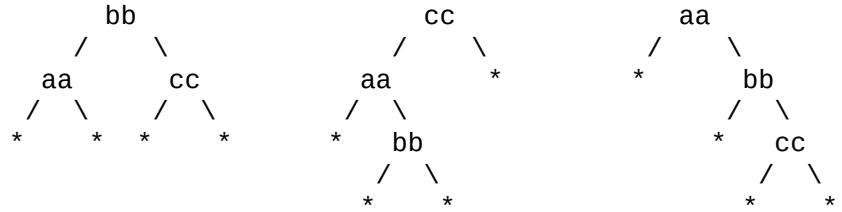
Explore the power of the methods you designed through the following examples:

1. insert the same items into a binary search tree several times in different order, then produce the result from the *inorder* method and check that they are the same.

2. insert the same items into a list of Strings, again several times in a different order, and sort these lists.

3. design a comparison between a binary search tree and a list of Strings to determine whether they contain the same Strings.

4. design the method *sameData* for the classes that represent binary search trees, that determines whether two trees contain the same Strings, not necessarily organized the same way. For example, any pair of the following three trees would pass this test:



5. design the method *buildTree* for the classes that represent binary search trees that consumes a list of Strings and produces a binary search tree, with all Strings in the list inserted in the tree. Verify that you produced the correct list by comparing the result of invoking the *inorder* method with the given list in sorted order. ∎

5.8 Problem: Challenge - will not be graded

Design the method *delete* that deletes the given String from the binary search tree, while preserving the binary search tree property. If the tree does not contain the String, the method just returns the original tree. ∎

5.9 Analytical Problem

The height of the binary tree is the longest path from the root *Node* to an *Empty Tree*. So for example, `Tree-1` and `Tree-2` in the first set of examples both have height 3, while the first tree in the **Problem 5.7** has height 2.

1. What is the smallest and what is the greatest height of a binary serch tree with 63 nodes?, with 31 nodes?

2. Make examples of the binary search tree of minimum and maximum height with 15 nodes.

3. Show all possible binary search trees that contain the following *Strings* and no others: `aa bb cc dd` ∎

Maze Game

Refer to **Problem 2.3** and **Problem 3.9** from Assignments 2 and 3.

5.10 Problem

Design the methods that computes the maximum money the player can win, when starting with the given amount. ∎

5.11 Problem

Design the methods that determines whether two rooms are the same. Two wizard rooms are considered the same if they use the same formula to compute the reward. ∎

5.12 Problem

Design the method *nextMove* for the class that represents the girl player. The method produces a new girl player. If the game is over, the method just returns the girl player unchanged. Otherwise, the girl's fortune is either increased by the elf or a wizard, or plundered by the monster. The method consumes a String `"left"` or String `"right"` that determines the choice of doors, left or right, and the player moves to that room. The player should not move anywhere, if the input is different from these two values. Of course, there are no doors in the wizard room. ∎

5.13 Problem

Design the methods that draw the three different rooms for the Maze Game, i.e. the *Monster* room, the *Elf* room, and the *Wizard* room in the *World*. They can be just rectangles of different color, but you may use some creativity along the way.

This problem will not be graded, but you need some graphical representation of the different rooms to proceed with the rest of the problems. ∎

5.14 Problem

Design the methods that draw some representation of the amount of money the girl player has. It can be as simple as a yellow circle of increasing or

decreasing radius, or a fancy pile of gold nuggets.

This problem will not be graded, but you need some graphical representation of the money to proceed with the rest of the problems. ∎

5.15 Problem

Add the girl player to the World. Define the two methods needed to play the interactive game — *draw* and *move* and *onKeyEvent*.

- The *draw* method should paint some background, draw the current fortune of the girl player, and draw the room where the player is currently.
- The *move* method produces a new *World* with the player that has made the next move, given a String with values `"left"` or `"right"` as its argument. If the argument is not one of these two strings, the world remains the same.
- The *onKeyEvent* method takes one argment, a String that represents the key that has been hit. The left and right arrow keys are represented by Strings `"left"` and `"right"`. This method just invokes the above *move* method with the argument it received.

You shold now be able to play the game. ∎