

Homework 3

Due March 5

Problem 3.1

We have seen depth-first-traversal on a tree, as in Algorithm 3.1.1:

Algorithm 3.1.1 DFT(tree T)

```

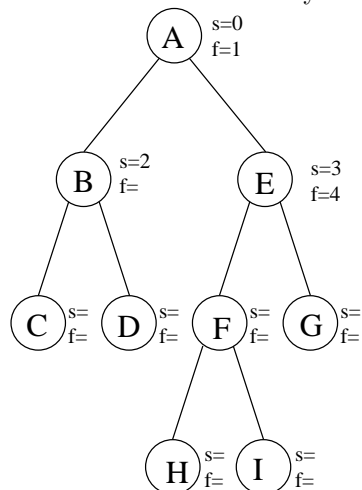
 $S \leftarrow$  new Stack
 $S.push(T.root)$ 
while ! $S.isEmpty()$  do
   $u \leftarrow S.pop()$ 
   $u.visit()$ 
  if  $u.hasLeft()$  then
     $S.push(u.left())$ 
  end if
  if  $u.hasRight()$  then
     $S.push(u.right())$ 
  end if
end while
return

```

Suppose we modify DFT as in Algorithm 3.1.2 on the next page.

When we are finished with a depth-first traversal of the tree, every node has a “start” and “finish” time noted.

(a) Hand-simulate this on the following tree, marking the start and finish times. We’ve done a few for you. Fill in the remainder.



(b) Pick a pair of nodes p and q and look at the time intervals $p.start$ through $p.finish$ and $q.start$ through $q.finish$. How are they related? Repeat for several

Algorithm 3.1.2 DFT(tree T)

```
 $S \leftarrow$  new Stack
int  $t = 0$ 
 $T.root.start = t$ 
 $t \leftarrow t + 1$ 
 $S.push(T.root)$ 
while ! $S.isEmpty()$  do
   $u \leftarrow S.pop()$ 
   $u.finish \leftarrow t$ 
   $t \leftarrow t + 1$ 
   $u.visit()$ 
  if  $u.hasLeft()$  then
     $k \leftarrow u.left()$ 
     $k.start \leftarrow t$ 
     $t \leftarrow t + 1$ 
     $S.push(k)$ 
  end if
  if  $u.hasRight()$  then
     $k \leftarrow u.right()$ 
     $k.start \leftarrow t$ 
     $t \leftarrow t + 1$ 
     $S.push(k)$ 
  end if
end while
return
```

pairs of nodes, being sure to try several varieties: siblings, descendants, etc. Make a conjecture about how the time intervals are related.

(c) Give a clear and coherent explanation for why they are related this way.

Problem 3.2

Recall that a binary tree is *proper* if each node in the tree has either zero or two children. The *balance factor* of an internal node v of a proper binary tree is the difference between the heights of the left and right subtrees of v . Show how to specialize the Euler tour traversal to print the label of each internal node, followed by its the balance factor. You may presume that the label of the node p is given by $p.label$. You need specify only the internals of the `visitLeft`, `visitBelow`, and `visitRight` functions.

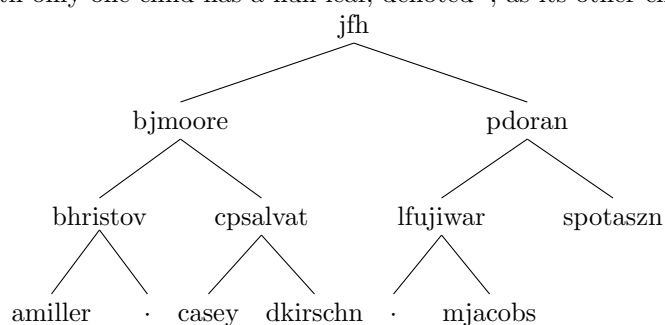
Problem 3.3

(a) Show how to implement the (standard) queue ADT using only a priority queue and one additional integer instance variable.

- (b) What is the running time of the enqueue and dequeue operations, assuming the priority queue is implemented as a binary heap?

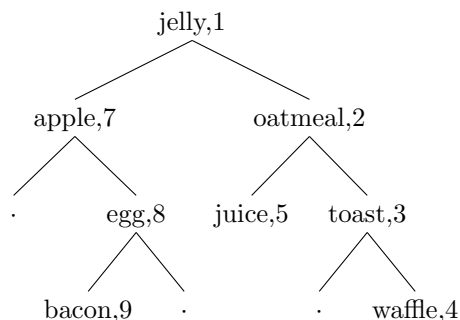
Problem 3.4

A binary search tree (BST) is one in which each node is greater than all its left descendants and less than all its right descendants, like the one shown below. In this case, the node contents are strings, and we're using dictionary order. We'll assume throughout this problem that all the entries are distinct: the same string never appears twice. (Note that we draw trees so that a non-leaf node with only one child has a null leaf, denoted \cdot , as its other child.)



- (a) Write the sequence of nodes encountered in an inorder traversal of this tree.
 (b) Explain why the inorder traversal of any BST is always in sorted order.
 (c) Consider the following data, where each datum consists of a key (a string) and a number:
 (toast, 3) (apple, 7) (egg, 8) (waffle, 4) (jelly, 1) (juice, 5) (bacon, 9)
 (oatmeal, 2)

We can put these into a binary tree as shown below:



We've constructed the tree cleverly, so that it is a BST when we look at only the keys, and it's in heap-order when we look at only the numbers (i.e., each node's number is less than the numbers of its children).

Construct a tree with the same properties, but using the following data instead:

(train, 7) (truck, 12) (canoe, 4) (dogsled, 1) (sedan, 8) (maglev, 2)
(kayak, 10) (magic carpet, 3) (raft, 9) (subway, 5) (chariot, 6) (biplane, 11)

- (d) Show that if you're given a list of key-number data with all the keys distinct, and all the numbers distinct, there's *always* a tree that's a BST with respect to the keys, and in heap order with respect to the numbers. Do this by writing pseudocode for constructing such a tree. Assume that the input is provided as a list of pairs, and that a pair has `getKey` and `getNum` methods. You may assume that you can compare strings or numbers with "`<`" or "`>`".
- (e) Explain why the tree constructed in part (d) is unique.

Problem 3.5

Robin keeps organized by assigning everyday tasks priority values, and then performing the most important task first (lowest value). For instance, on a normal everyday basis, cleaning the apartment or washing dirty dishes might be less important than watching television. However, when Robin's friend Lulu comes to town, priorities change. Wanting to give Lulu a good impression means that having a clean apartment or getting reservations at a nice restaurant might be more important. In this problem we will design a data structure called a dual-priority queue, that can order Robin's tasks according to two different priority values — one for most of the time, and another for when Lulu comes to town.

(a) Associated with a task T are two numbers $T.N$ (the "normal priority") and $T.L$ (the "Lulu" priority). Describe briefly how to build a data structure using two heaps, so that `removeMinN()`, `removeMinL()`, and `insert(T)` all run in $O(\log n)$ time when there are n tasks in the structure. `removeMinN()` should remove and return the task T with minimal $T.N$; likewise `removeMinL()` should remove and return the T with minimal $T.L$.

(b) Give pseudocode for `getMinL()` and `insert(T)`

Problem 3.6

- (a) Implement in Python the `BinarySearch` on an array of sorted integers in increasing order.
- (b) Implement in Python Depth-First Traversal and Breadth-First Traversal on a the `LinkedBinaryTree` you implemented for Homework 2.

The TAs will post in the Google Group and/or MOTD additional instructions on how to start this problem and how to hand it in, and will perhaps provide a few test cases for you as well.