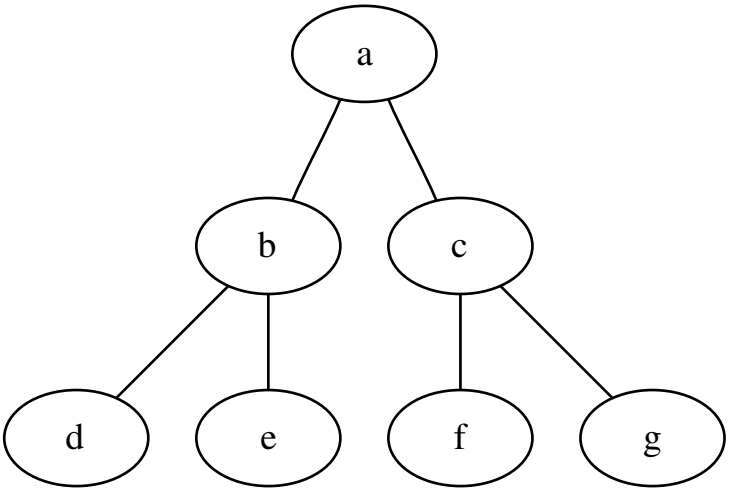
Design of Information Structures - Lab 5

January 25, 2019

1 Introduction

This laboratory explores data structures called binary trees. You will use a binary tree to build a decoder for Morse code, and sort a set of numbers. Copy the files for this laboratory from the CS126 web site (download file `lab5.zip`).

2 Trees



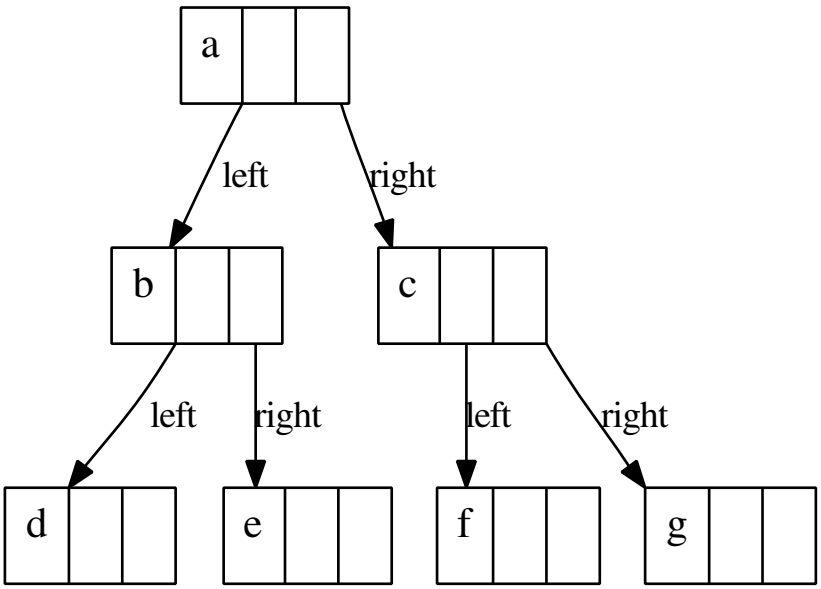
The diagram above shows a simple tree with a root node labelled A, internal nodes B and C, and leaf nodes D, E, F and G. We can say that A is the parent of B and C, that B and C are the children of A. Also, there are three levels in the tree. One thing you will notice about the tree is that all nodes have at most two children, we call these trees binary trees.

Recall lab 3 where we wrote some java code to implement linked lists. Each node of the list was an object and there was an encapsulating object for each list. We will use the same concept to implement trees. The key difference is that

Table 1: Morse Code

Letter	Morse Code		
a	.-	b	-...
c	-.-.	d	-..
e	.	f	..-.
g	–.	h	....
i	..	j	.—
k	-.-	l	.-..
m	–	n	-.
o	—	p	.–.
q	–.-	r	.-.
s	...	t	-
u	..-	v	...-
w	.–	x	-..-
y	-.–	z	–..

instead of keeping references to successor elements in the list, we keep track of the left and right children of each node:



You may want to have a look at the `BinaryTreeNode` class from the source code that you downloaded, it implements a tree node which contains some data and has references to left and right children.

The rest of this lab session will focus on translating normal letters into and out of morse code. The specific character conversions are listed in the following table. Don't worry, you needn't be familiar with these, they are visible overleaf so you can test your source code.

The `Morse` and `Esrom` programs provide command line encoding tools. If you compile and run `Morse` it will translate inputted text into morse code. `Esrom` does a reverse translation, but takes its input from command line arguments. These classes use `Coder` and `Decoder` respectively to do the underlying work. Unfortunately `Coder` and `Decoder` are incomplete, during the next two milestones you shall finish their implementation them.

Exercise 1 *Look at the* `Coder` *class, the* `encode` *method currently returns* `null` *for any character. The* `code` *array contains the Morse Code for each letter indexed sequentially, so that the index of the character 'a' is 0, 'b' is 1, etc. Alter the* `encode` *method so that it returns the correct morse code character for any given alphabetical character.*

Binary trees will be used to translate from morse code strings to letters in `Decoder`. At each node in the tree, left shall correspond to '.' and right to '-'. If you look at `Decoder` you will see that we have started the process of adding the Morse Code Strings by hard coding some of the tree in the constructor.

Exercise 2 *Draw the tree after the constructor has been called on the Decoder class, you will notice that it follows the comment 'There must be a better way to do this'.*

Milestone 1) You have now completed *Milestone 1.*

Exercise 3 *Add another 5 or so characters to the tree in the decoder class, in the same manner as the three example characters are implemented.*

Exercise 4 *Read the* `decode` *method, you need to implement the two missing statements in order to finish this method. You can test this using the* `Esrom` *program.*

Exercise 5 *You can implement the* `decode` *method both iteratively and recursively. If you look at the* `decodeR()` *then you will see a recursive solution, but it is also missing two statements. Both of which are recursive calls to the method. You need to implement these statements and then modify the* `Esrom` *program in order to use the recursive method.* NB: *make sure that you call the publically accessible method from* `Esrom`.

Milestone 2) You have now completed *Milestone 2.*

One of the most common uses of Binary Trees is to keep an efficiently sorted list. If you recall lab 4, we kept a list of strings in a sorted order, but this required a lot of string comparison and sorting everytime a string was added. At this point in time you might wish to examine the BinaryTree.java and BinSort.java classes.

Examine the `add` and `addToSubTree` methods and figure out what the tree which is being built looks like. Running the program `BinSort` will give you some idea of what is happening. Currently, `traversal` does what is known as an inorder traversal. In what order are the parent and its children being visited in an inorder traversal?

Exercise 6 *What order would you need to visit the nodes to print out the numbers in a descending sequence? Change the* `inOrder` *method to make this happen.*

Exercise 7 *For preorder traversal the order is: parent, left-child, right-child. Implement a preorder traversal by completing the* `preOrder` *method. Test it out by changing the call statement in* `traversal`*. What's the result of running the BinSort program now?*

Milestone 3) You have now completed *Milestone 3*.
