CSc 127B — Introduction to Computer Science II
Fall 2015 (McCann)

http://www.cs.arizona.edu/classes/cs127b/fall15/

Program #11: Decoding Prefix Codes

Due Date: **Wednesday** *December 9th, 2015, at 10:00 p.m. MST*

Overview: There are many ways to encode information. The ASCII sequence is used to encode character relationships, for example. ASCII codes are also fixed-length, hence the terms 7-bit ASCII and 8-bit ASCII. Uniform lengths make turning a sequence of bits into the corresponding ASCII characters very easy.

But what if you wanted variable length codes? Having such a code would permit values that are very popular (say, the letters 'e' and 't' in English prose) to have shorter codes than less-popular values, thus saving space. Without a uniform code length, we need a way to tell when the code for one value stops and the next starts. One way to do this is to create a *prefix code*. In a prefix code, the encoding of any value is never the first part of the encoding of any other value. This means that if we read a pattern and match it to a value, we know that we can stop trying to find a match; there isn't a longer potential match to worry about.

For example, say that the encoding for 'e' is 11, and the encoding for 't' is 110. After reading the pattern 11, we don't know if we should say that we found an 'e', or keep going to see if we'll find the code for 't'. In a prefix code, this situation can't happen. 't' would have to be encoded with a value whose prefix is something other than 11.

One way to create a prefix code is called *Huffman Coding*. It generates binary encodings, which map nicely to binary trees if we allow the left child references to represent 0 and the right to represent 1. You won't be writing the encoder for this assignment. Instead, we'll give you a non-hierarchical representation of the tree and an encoded sequence of values. Your job will be to reconstruct the tree from the given representation, and use it to decode a given sequence of values.

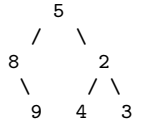
Assignment: Write a complete, well–documented Java program named `Prog11.java` that prompts the user for the complete pathname for an input file. The content of that file is specified in the Data section, below. After building the tree from the representation stored in the file, your program will output the tree's postorder traversal and the decoded sequence of values. Thus, there are two main tasks:

First Task: The first task is to build the decoding tree, which can be done recursively from the preorder and inorder traversals. We know that the first value in a preorder traversal is the tree's root, and we can build a binary tree node to hold it. In the inorder traversal, the values ahead of the root are in the root's left subtree, and those to the right are in the right subtree. Each of those groups of values will be grouped together in the preorder traversal, too, but likely not in the same order. Because each of those two groups came from subtrees, the preorder traversal reveals the root of each subtree. Thus, we can recursively continue the process on the content of the subtrees and thereby reconstruct the original tree.

As usual, an example will help. Here are the preorder and inorder traversals of a binary tree of integers:

Preorder: 5 8 9 2 4 3 Inorder: 8 9 5 4 2 3

The first value in the preorder sequence is 5. Ahead of 5 in the inorder traversal are 8 and 9, and after it are 4, 2, and 3. Thus, 5's left subtree must contain 8 and 9, and the right must contain 4, 2, and 3. But in which configurations? Back to the preorder traversal. 8 is ahead of 9 in the preorder sequence, meaning that 8 is the root of that subtree. In the other group, 2 is the root. We can call the method on itself twice, once for each subgroup, to form those trees. When completed, the resulting tree is:



(Continued . . .)

Second Task: With the tree constructed, the third line of the input file can be decoded. Start at the root of the tree, and read the first character. If '0', follow the left child reference. Otherwise, follow the right. When you reach a leaf node, output the node's value. Return to the tree's root, and continue until the input is exhausted. No recursion required! Note that the values of the internal nodes will never be output.

Data: An input file will be a text file containing three lines of text. The first line is the preorder traversal of the decoding tree. This will be a list of integers, each separated from the next by one or more spaces. Each integer will be unique; there will be no duplicates. The second line is the inorder traversal of the same tree, formatted in the same way as the preorder traversal.

The third line is the encoded sequence of values. The encoding will take the form of a sequence of the characters '0' and '1'. (That is, we're simulating a file of bits. Actually reading a file and accessing its content bit by bit is possible in Java, but doing so would add complexity that we don't need. Instead, we'll just pretend.)

Here's a sample data file, the location of which your program will get from the user:

```
9 0 6 3 2 8 4
6 0 2 3 8 9 4
0001011011100011
```

(We're not showing the corresponding tree; that's for you to figure out!)

Output: Your program is to output two sequences of values. The first is the postorder traversal of the tree you've build from the first two lines of the data file. The second is the decoded sequence of values.

For example, here's output that corresponds to the sample data file shown above:

```
Postorder Traversal:  6 2 8 3 0 4 9
Decoded Sequence:  62448468
```

Turn In: Electronically submit your `Prog11.java` file using the turnin program and the usual naming convention.

Want to Learn More?

- You don't need to know anything about Huffman Coding to do this assignment, we promise. But, if you'd like to learn more, here's one good source:
  `http://planetmath.org/huffmancoding`
- The frequency of leading digits in numbers is surprisingly interesting. In a wide variety of domains, the distribution of the leading digits follows *Benford's Law*. Such non-uniform distributions work well with prefix codes. If you're curious, see:
  `http://mathworld.wolfram.com/BenfordsLaw.html`
- In 1889, a two-page academic paper was published with a list of not only the frequencies we now associate with Benford, but also the frequencies of the second digits, as found in logarithm tables. You might find it interesting. It's accessible at no cost on-line if you're on-campus:
  `http://www.jstor.org/stable/2369148`

Hints, Reminders, and Other Requirements:

- You may be wondering, "Where are all of the class details for the tree, its nodes, etc.?" That's all up to you this time. We've had ten previous programs; by now, you ought to know what to do and how to do it. Impress us!
- Yup, this is the last program. If you get all weepy, just blame cold and flu season. :-)