Great Theoretical Ideas In C.S.                                      Guru: Noah Falk
Assignment 4                                      Due: February 11, 2003

1 Overview

Your assignment will be to write a binary tree compressor and decompressor, commonly referred to as a codec. You will do this by first creating a method of enumerating binary trees. Then you will make use of this enumeration to help you compress and decompress trees, optimally in fact. The golden ideas of this assignment:

- Anything with an inductive definition can be enumerated.
- Anything that can be enumerated can be compressed by giving its index in the enumeration.

If you remember anything from this assignment make it the two previous sentences. We’ll define exactly what they mean a bit further on.

2 Working with Binary Trees

You may recall we introduced graphs in recitation 2. Rooted, ordered, binary trees are a specific class of such graphs. Rooted means that we can distinguish the root node from other nodes. Ordered means that the left sub-tree can be distinguished from the right sub-tree. We will use the following inductive definition:

A binary tree is either

- empty (no nodes) or,

- has a root node, and a left and right binary tree which are called the sub-trees. There is an edge between the root of each non-empty sub-tree and the new root.

You can also find definitions for these and many other tree related terms at http://www.nist.gov/dads/HTML/binarytree.html.

A tree traversal defines an ordering of the nodes in a tree. The three common orderings are pre-order, in-order, and post-order. For pre-order the root is first, the nodes in the left sub-tree come next, and the nodes in the right sub-tree after those. To determine the ordering among the nodes of a sub-tree we recursively apply the rule. For in-order traversal first order the left sub-tree, then the root, then the right sub-tree. Post-order gives the left sub-tree, then the right sub-tree, and last the root. Here are examples of the various traversals.

```
  1
 / \ 
2  5
 / \ 
3 4 6
```

Pre-order

```
4
/ \ 
2 5
/ \ 
1 3 7
```

In-order

```
8
/ \ 
3 7
/ \ 
1 2 6
```

Post-order
Consider the following method to represent a tree without resorting to ascii art. First we label all the vertices using an in-order traversal of the tree. Then we can describe that tree uniquely by outputting the number at each node using a pre-order traversal. So for example the tree above can be represented by the sequence of numbers 4, 2, 1, 3, 5, 7, 6, 8. For this assignment, any time a tree needs to be input or output from your code it will be represented by an array of integers in this fashion. Consider how you could reconstruct the tree from that sequence of numbers before tackling the written problem below. Also note that while in-order and post-order can also be used to uniquely represent trees, pre-order and post-order doesn’t always work. Not all traversals are created equal.

3 The Written Problem (20 points)

Describe in words how to go from the sequence (pre-order output of an in-order numbering) to the tree it came from. Explain why all your choices are forced and conclude this representation is 1-1. However it is not onto as there are sequences of natural numbers which do not come from any binary tree. Do not worry though, we will always test your program using sequences which do represent a tree.

4 The Programming Problem

For the first part of the programing assignment you will implement the following functions:

Vector enumerateTrees(int num_nodes);
int countTrees(int num_nodes);

The first function should produce an enumeration of all trees with num_nodes number of nodes. Intuitively an enumeration is a listing but formally an enumeration is a 1-1 onto correspondence from the natural numbers to the things being enumerated. When enumerating finite sets of some size n we will require a correspondence from the first n natural numbers. For our program a java Vector is a clear such correspondence, an index i corresponds to the tree which will be elementAt(i). Each element should have type int[] and represent a tree using a sequence of numbers as described above.

The second function should return the number of trees which can be made with num_nodes nodes. We give you a trivial implementation that may be helpful in testing your code. Of course you may find it useful for other things too, and you are welcome to implement it differently as long as it gives the right answer.

See Section 6 for some helpful hints.

5 More Programming Problems

For the second part of the programming assignment you will need to implement the following functions:
int compressTree(int[] tree);
int[] decompressTree(int compressed_tree);

For the first function you need to “optimally compress” the given tree. Specifically from above you know that there are exactly \( \sum_{i=0}^{n} \text{countTrees}(i) \) number of trees with \( n \) or fewer nodes. Your compression function will map each of those trees to a number in the range \([0, \sum_{i=0}^{n} \text{countTrees}(i) - 1]\). For the decompress function you must reverse the work of your compress function. In fact we don’t care which number your compression function maps to in the given range as long as the decompress function can recover the tree. Do you see why compressing to any smaller set of numbers would make correct decompression impossible?

As an aside, there are many other desirable qualities a real-world codec often has and other notions of optimality. For instance if you know that certain objects occur more often than others, we can make the common ones small and the less common ones larger. Also many compressors forsake a little space to be faster, or to allow common operations to be efficiently applied while the object is still in compressed form. Finally many codecs, such as for audio and video, save tremendous amounts of space by allowing the decompressed version to differ slightly from the original.

6 Hints

- Recall the big idea that inductively defined things can be enumerated. Here is the informal method for doing so for binary trees. A binary tree with \( n \) nodes is built using two other binary trees whose sizes sum to \( n - 1 \). To get all possible binary trees of that type we need only generate each combination of smaller sized trees. But how to get the smaller sized trees? Very simple actually... call enumerateTrees() and it will give them to you. Of course binary trees are only one example of the much more general statement we are claiming, but you may have to tackle that problem sometime later ;)

The compression/decompression relies directly on enumerating and counting. Remember that the enumeration function only enumerates trees of a specific size, however the compression algorithm must work with trees of any size. Therefore don’t be fooled into mapping the first tree of each size onto the number 0. When we give you back 0 you won’t know which size tree it came from. However you will likely find a good method of combining the enumerations of each size into a large enumeration of all binary trees. The compression function is just the inverse of an enumeration function... map each tree onto its index in the enumeration of all binary trees.

- Don’t forget the empty tree! It is represented as an array of length 0.

- You may find it handy to write this helper function

  ```java
  private int[] concatenateTrees(int[] left, int[] right);
  ```

  The job of this function is to take two trees and renumber them to get a tree containing both as sub-trees. Of course you do not have to keep the trees in the array representation to work with them, but I recomend you consider it. Its not too tough.

- What do you think the output of this code is?
int[] a = new int[2];
int[] b = new int[2];
a[0] = 1; a[1] = 2;
b[0] = 1; b[1] = 2;
if(a == b) System.out.println("We are equal!");
else System.out.println("We aren’t");

They won’t be equal because they are two separate objects, regardless of the fact that they contain the same values. If you are trying to test arrays for equivalence make sure you don’t try that way. Perhaps a helper function?

7 A Few Programming Rules

- Do not precompute any results and hard-code them into your program – no lists of trees, no counts for different sizes, NO hard-coded answers to the test inputs. That is cheating, we’ll catch you, we’ll be angry.

- A reminder, do not look at anyone’s code, share code, write code together, write pseudo-code together, find code online, etc. This is also cheating. You may assist each other with issues which are clearly more general the current assignment such as explaining compiler errors or how the API works for Java’s Vector class. I wish I didn’t have to bring it up and I know most people still have both a good memory of the cheating policy they signed and no intention to ever cheat. However it has been the case in the past that on the first real programming assignment a few people forget the rules.

- Your program may not use an unreasonable amount of memory. The Java virtual machine has a default memory space (64 MB?), and the driver code will need a little itself (perhaps 1 MB, probably less). The inputs will be small enough that our inefficient solution code stays far under the memory limits. Nonetheless be aware that if you get OutOfMemoryErrors when running the check script on your code, you should get that fixed.

8 How to Hand In

You must write your program in Java. We have created a template file to help you get started. Copy it from the assignment directory to your home directory. The assignment directory is here:

    /afs/andrew/scs/cs/15251/assignment4

Make whatever changes you want to Assignment4.java, and then copy this file only to your handin directory:

    /afs/andrew/scs/cs/15251/assignment4/handin/<youruserid>

You should test that your program has been submitted correctly by running the “check” script, found in /afs/andrew.cmu.edu/scs/cs/15251/bin/check. The check script will simulate the grading procedure and tell you what grade you will get for that handin. PLEASE make use of this!
9 Grading

- Written Problem (20 points)
- Enumeration/Counting (50 points)
- Compression/Decompression (30 points)

If your program has been running for 5 minutes we will stop it at that point and any parts which are incomplete will receive no points. Likewise if your code has any fatal error, you will receive no points for the remaining parts. We will be providing inputs for which neither time nor memory should be of great concern. Unless the check script indicates otherwise, you need not worry.

You may hand in your assignment multiple times over the course of the 7-day grading period, but the lateness penalty will apply as usual. Every night at midnight, starting on the night the assignment is due, the electronic grading process will run automatically. The results will be available the next morning as a file in your handin directory, and your current best grade will be posted to your grades page on the Web.