Notes for Tutorial 9

6.042 - April 11 and April 14, 2003

The topics this week are:

1. Drill on recursive definitions.
2. Recursive definition of rooted full binary trees.
3. State machine proof that the "9-puzzle" is not solvable.

1 Drill on recursive definitions

The first attached page has some easy recursive definition problems. In lecture, students saw how to recursively define a set S of mathematical objects: trees, strings, etc. A common objective after you've done that is to recursively define a function f whose domain is S. For each part of the definition of S, there should be a corresponding part of the definition of f. They haven't seen how to do this, but can probably figure it out on the fly.

2 Recursive definition of rooted full binary trees.

Now we'll do a more complicated recursive definition, define some ancillary functions, and use structural induction to prove theorems.

First, *informally* define rooted, full binary trees. (Let's just call them "binary trees" hereafter, to save paper and oxygen. Point out that there are numerous micro-variants on binary trees, if you like.) Draw a few examples and describe them: they're trees, one vertex is called the root, each vertex has either two children or zero, etc. Answer specific questions, but giving a careful definition yourself defeats the purpose of the exercise, which is to formalize an intuitive notion! They should end up with something roughly like this:

Definition 1 *The set of binary trees is defined recursively as follows.*

1. *A graph containing a single vertex v and no edges is a binary tree with root v.*
2. *If $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ are binary trees with roots v_1 and v_2 and v is another vertex, then the graph*

$$T = (V_1 \cup V_2 \cup \{v\}, E_1 \cup E_2 \cup \{(v, v_1), (v, v_2)\}$$

is a binary tree with root v, provided V_1, V_2, and $\{v\}$ are disjoint sets.

If their initial definition is informal, you might ask how to make it more rigorous. (Prof. Leiserson noted that the definition of free trees given in lecture was informal.) In any case, make sure that (1) their binary trees have roots, and (2) that appropriate vertex sets are required to be disjoint. To wrap this part up, you might also draw a binary tree and ask how it is constructed by applying the rules in their definition.

Point out that binary trees come up everywhere in computer science: searching, sorting, data compression, etc. The problems are all straightforward.

2.1 Huffman Encoding

You might want to point out that prefix-free codes are used in a data compression technique called *Huffman encoding*. The idea is to map English letters to binary strings in a prefix-free code. In particular, we associate *short* binary strings with *common letters*. For example, if our alphabet had only three letters and a is used a lot, we could use the association:

$$\begin{aligned} a &= 0 \\ b &= 10 \\ c &= 11 \end{aligned}$$

Now you translate a message by mapping each letter to its binary string and concatenating them:

$$abacab = 0\ 10\ 0\ 11\ 0\ 10$$

In practice, instead of every letter taking, say, 8 bits, the common letters take just 2 or 3 bits and the uncommon letters might take 10 or 12. So *most of the time* we save bits at the cost of *rarely* losing a few bits— this is a big win! The prefix-free aspect is important so that we can uniquely decode, e.g.

$$\begin{aligned} 1100100100001010 &= 11\ 0\ 0\ 10\ 0\ 10\ 0\ 0\ 0\ 10\ 10 \\ &= caababaaabb \end{aligned}$$

2.1.1 Extra

If they chow through this stuff, ask them to prove the Kraft inequality. Suppose each vertex at depth d has *weight* $1/2^d$. Prove that the sum of the weight of the leaves in a binary tree is at most 1. (This isn't very hard, either.) This shows that a binary tree can not have too many leaves close to the root, or— in terms of Huffman codes— not too many letters can get short codes.

3 The 9-Puzzle

The students need one concept to get them started on this problem. Suppose I have a list containing the numbers 1 through 9. An *inversion* is a pair of numbers not in increasing order. For example:

1 8 2 5 3 4 6 9 7

This has the following inversions: (8, 2), (8, 5), (8, 3), (8, 4), (8, 6), (8, 7), (5, 3), (5, 4), and (9, 7). You may also need to define parity, which refers to whether a number is even or odd. 0 is even. The questions are straightforward. (The probability one uses a bijection: swap the first two elements.) If you have extra time, ask them to prove that you can get from any position with an even number of permutations back to the starting state.

Give recursive definitions for the following sets and functions:

1. The set A of all finite-length binary strings.
2. The set B of all finite-length binary strings that do not contain three consecutive zeros.
3. The function $z : B \to \mathbb{N}$ that maps each binary string in $x \in B$ to the number of zeros contained in x.
4. The function $d : B \to \mathbb{N}$ that maps each binary string in $x \in B$ to the number of ones contained in x minus the number of zeros contained in x.

The following problems concern *rooted full binary trees*, which we refer to hereafter simply as "binary trees" for brevity.

1. Give a recursive definition of the set U of binary trees.

2. Recursively define a function L mapping each binary tree in U to the set of *leaves* of that tree; that is, the set of vertices of degree 0 or 1.

3. Use structural induction to prove that the number of edges in a binary tree is twice the number of leaves minus two $(2L(T) - 2)$.

4. Recursively define a function d_T that maps each vertex in a binary tree T to its *depth*, which is the length of the shortest path from that vertex to the root.

5. Recursively define a function P_T that maps each vertex at depth d in a binary tree T to a distinct d-digit binary number. The root should be mapped to the empty string, all vertices to the left to strings starting with 0, all vertices to the right to strings starting with 1, etc.

6. Use structural induction to prove that for every binary tree T, the function P_T maps the leaves to a set of *prefix-free* binary strings; no binary string at a leaf is a prefix of the binary string at another leaf.

Suppose that the numbers 1 to 9 are listed in some order.

1. How does swapping two adjacent numbers affect the parity of the total number of inversions?
2. How does swapping two non-adjacent numbers affect the parity of the total number of inversions?
3. Suppose that we pick an ordering of the numbers 1 to 9 uniformly at random. What is the probability that it has an even number of inversions?

The *9-Puzzle* consists of a numbers 1 through 9 arranged in a square grid:

$$\begin{array}{ccc} 8 & 9 & 5 \\ 7 & 6 & 2 \\ 4 & 1 & 3 \end{array} \qquad\qquad \begin{array}{ccc} 6 & 5 & 9 \\ 8 & 7 & 1 \\ 2 & 4 & 3 \end{array}$$

The goal is to rearrange the numbers into the *winning configuration*:

$$\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array}$$

The only allowed operations are row rotations and column rotations. Rotating a row moves all entries to the right one place, except the rightmost one, which returns to the left side. For example, rotating 1 2 3 gives 3 1 2. Similarly, rotating a column moves all entries down one place, except for the bottom one, which returns to the top.

1. Show how to win the game from the first example configuration.
2. Model this problem as a state machine.
3. Use the Invariant Theorem to prove that the game can not be won from the second example configuration.