I am revisiting the Hu-Tucker algorithm, since I messed up with the homework problem with the Hu-Tucker algorithm last time, and gave you a sequence which produced a tree with no crossing edges. This didn’t let you see some of the complexities of the algorithm in action. I also wrote up some notes on the Hu-Tucker algorithm which are on the class website, which are closer to the way I presented the algorithm in class, and give what I believe is a proof that the algorithm always produces an order preserving tree. You may want to look at them before doing 1b and 1c.

1a: Construct the Hu-Tucker tree for frequencies (in order) of

6, 5, 1, 2, 7, 5, 4, 2, 10, 9.

1b: Show that in the Hu-Tucker algorithm, if you join the two compatible nodes which give the minimum frequency for their sum, then the non-leaf nodes are created in order of ascending frequency.

Hint: Suppose not. Then we must have at some point combined nodes with frequencies $a$ and $b$ and then nodes with frequencies $c$ and $d$, where $a + b > c + d$. Thus, $c$ and $d$ could not have been compatible before $a$ and $b$ were combined. Use this to show that either $(a \leq c$ and $b \leq d)$, or $(a \leq d$ and $b \leq c)$, a contradiction.

1c: Construct a set of frequencies such that if at every point you join two compatible nodes, but you break ties inconsistently, the Hu-Tucker algorithm fails and does not construct an order-preserving code.

I promise to have notes for the Lempel-Ziv algorithm on my website by the end of the weekend.

2a: Construct the parsing that the Lempel-Ziv algorithm produces for the sequence

$ABBABBABAABBBABABBABBABABA$

for the sequence

$ABABABABABABABABABABABABABABABABABABABABAB$.

2b: For each of these sequences, can you find a parsing that produces more distinct phrases than the Lempel-Ziv algorithm does?

3: Construct a linear (i.e. matrix) single error correcting code mapping 11 bits to 15 bits.

If you would rather do (4) and (5) using MATLAB, go ahead.

4: Create a spreadsheet encoder for your code.
5: Create an automatic spreadsheet decoder for your code. This should allow you to decode any 15 bit received message and reverse any single bit error and find the sent message, if there is at most 1 error.

Hint for (5): one way to do this is:
Write down your decoding matrix $D$.
Assign a distinct number to each row of it by treating its bits as forming the bits of a number in binary. Thus 0110 would be 6, etc (this takes one instruction, and copying).
Write your “received message” as a column.
Form a new matrix whose entries are the products of the entries of $D$ with the corresponding entry of your received matrix. (Takes one instruction and copying).
Sum the columns of this mod 2 and form its number. (easy)
Identify which row of $D$ has the same number and copy the received message flipping the bit in that row. (use one if statement, copied)