CS340 Winter 1996
Ming Li

Assignment #2, Due Date: Feb. 9

1. (2 marks) The goal of this problem is to help you understand the dynamic programming technique used to compute the optimal binary search trees. Simulate, by hand, this algorithm to compute the optimal binary search tree with the following data:

   Keys: $x_1 < x_2 < x_3 < x_4 < x_5 < x_6$.

   Let

   $$\beta_i = \text{Prob}(\text{access } x_i),\ \ i = 1, 2, \ldots, 6.$$

   $$\alpha_i = \text{Prob}(\text{access } x,\ \ x_i < x < x_{i+1}),\ \ i = 0, 1, \ldots, 6,$$

   (assuming $x_0 = -\infty$ and $x_7 = \infty$). $\beta_6 = 0.5$, $\beta_i = 0.1$ for $i = 1, 2, 3, 4, 5$. $\alpha_i = 0$ for $i = 0, 1, \ldots, 6$.

   You should compute the matrices C and R with the entry of C_{ij} being the optimal cost from x_{i+1} to x_j and R_{ij} being the optimal root for this range, and draw the optimal search tree.

2. (Programming question, 8 marks) In this problem you will be using trees to represent arithmetic expressions and to perform some algebraic manipulations on them. Expressions consist of variables (with a single letter name) and constants (unsigned integer numbers).

   To avoid problems parsing the input expressions, we will assume that they are written in reverse polish notation (RPN). This is the postfix notation where the operator comes *after* the operands. For instance, the expression `(a+b)*(1-x)` would be represented as `a b + 1 x - *`. The operands allowed are `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), `^` (power). In infix notation the unary minus sign is the same as the binary one, but in RPN, to avoid ambiguity, we will use an underscore (`_`) to represent the unary minus. Thus, the expression `a*(-b)` is represented as `a b _ *`. To simplify reading these expressions, we will assume that they are written one token per line.

   Of course, this is not the notation we are most familiar with, so we have provided two commands (that you will find in the `bin` directory of the `cs340` account). The command `polish '`*expression*`'` receives an expression in standard infix notation as its argument, and writes the equivalent RPN expression in its standard output. The companion command `infix` reads an expression in RPN from its standard input and prints an equivalent infix expression in its standard output. If you would like to use your home computer, you can download the programs called polish and infix (they are written in PERL) in the bin directory.

   You are to write a program that acts as a filter, reading an expression in RPN, and outputting a *simplified* version of the expression, also in RPN. These simplifications should include the following ones: computation of constant sub-expressions, and detection of addition of zero, subtraction of or from zero, multiplication by zero, multiplication or division by one, and raising to the power zero or one. (Expressions in need of this kind of simplification are often generated as a result of simple-minded symbolic differentiation.) For example, the output of

   `polish '(a*1+b*0)^(3-2)-2*6' |` *yourprogram* `| infix` should be `a-12`.

You should organize your program in the following way:

(a) Read the input formula and build a tree representation of it. Use a stack to store the subtrees you are working on (actually, you store pointers to them).

(b) Program the simplification of a formula as a recursive function that receives a tree as input, and returns an equivalent simplified tree. This function is recursive because you will have to simplify the subtrees before being able to simplify the whole tree.

(c) Finally, output the simplified formula by traversing the tree.

Run your program on the expression given in the preceding example, and also on `'(-2+6-4)*(a+b)-(c+1*d)'` and `'(x-y)^((1-1)*z)+(-1)^3'`. For convenience, you can assume that all input numbers are single digit numbers. You can add your own examples to illustrate other simplifications that you have implemented. Include also examples of expressions with simplifications that your program does not handle, and discuss how hard it would be to extend it to cover those cases.