STAT 534 Homework 2

Out Sunday April 14, 2019

Due Monday April 22, 2019 (noon)

mmp@stat.washington.edu

Problem 1 (after is Exercise 5-10, page 118 in Kernighan & Ritchie)
a. Write in pseudocode the program `expr`, which evaluates a "reverse Polish" expression entered as a command line, where each operator or operand is a separate command line argument. For example,

`expr 3 4 +`	evaluates $3 + 4 = 7$
`expr 3 4 + 2 *`	evaluates $(3 + 4) \times 2 = 14$
`expr 2 3 4 + *`	evaluates $2 \times (3 + 4) = 14$

In general, the syntax *expression1 expression2 operanor* translates in the usual mathematical notation to *(expression1) operand (expression2)*. An *expression* is either a *number* or *expression1 expression2 operator*. A valid expression is one that can be represented by a tree with operators at the internal nodes and numbers at the leafs. For example, `expr 2 3 4 + * 3 6 - * = ( (2 * ( 3 + 4 )) * ( 3- 6 )` is represented by the tree

```
         *
        / \
      *     -
     / \   / \
    2   + 3   6
       / \
      3   4
```

This problem has an elegant solution using a stack. Hence, in your pseudocode, use the stack functions

The reverse Polish form with the stack implementation is a convenient form for arranging an arithmetic expression so that it is ready to be evaluated by the central processing unit of a computer when a program is run.

Require that all command line arguments are floating point numbers or the operands $+, -, *, :$. Check for any other possible exceptions and take appropriate actions (e.g print an error message).

b. Implement in python the stack functions Stack-Empty, Pop, Push using the python `list`. Then implement the `exec` function above.

c. Execute

`expr 5 2 7 4 * + /`	0.166667
`expr 2 2 - 2 * 2 2 /`	1
`expr 1.5 10 4 5 / * +`	9.5
`expr 1 2 3 4 + - *`	-5
`expr 5 6 7 8 - * +`	-1

and print each results as a single number, on a separate line. All code must be in the same file; call your file `hw2-expr.py`.

Problem 2 – Search trees

a. 12.2-1 NOT GRADED Suppose we have numbers 1 to 1000 in a binary search tree and want to search for the number 363. Which of the following sequences could NOT be the sequence of nodes examined?

1. 2,252, 401, 398, 330, 344, 397, 363.
2. 935, 278, 347, 621, 299, 392, 358, 363.
3. 2, 399, 387, 219, 266, 382, 381, 278, 363.

b. 12.2–4 Professor Bunyan thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k ends in a leaf. Consider three sets: A, the keys to the left of the search path; B the keys on the search path; C the keys to the right of the search path. Prof. Bunyan claims that any three keys $a \in A, b \in B, c \in C$ must satisfy $a \leq b \leq c$. Give a counterexample to the professor's claim.

c. 12.3–2 NOT GRADED Suppose that a binary search tree is constructed by repeatedly inserting distinct values into the tree. Show that the number of nodes examined in searching for a value is one plus the number of nodes examined when the value was first inserted in the tree.

Problem 3 – Binary search trees with equal keys

a. Assume that $n > 1$ items with identical keys k are inserted into an empty tree. What is the asymptotic performance of TREE-INSERT in this case? Give a short proof (by e.g. induction).

b. Suppose that the operation of **a.** has just been performed, and that now we are requested to delete *any* node with key k. What is the asymptotic performance of this variation of TREE-DELETE?

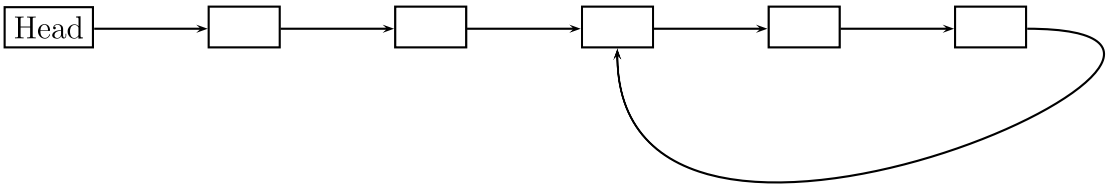
c. We propose to improve TREE-INSERT by testing before line 5 whether or not $key[z] = key[x]$ and by testing before line 11 whether or not $key[z] = key[y]$. If equality holds, we implement the following strategy. Keep a boolean flag b at each node, and set x (respectively y) to either $left[x]$ or $right[x]$ based on the value $b[x]$, which is switched during insertion every time it is checked. Suppose we insert 7 nodes with equal keys in an empty tree. Draw the resulting tree.

d. Find the asymptotic performance of inserting $2^m - 1$ nodes with equal keys in an empty tree, using the strategy in **c.**

e. Instead of the strategy in **c.** we propose now to maintain a *doubly linked* list at each node x, and insert in it any node with key equal to $key[x]$. What is the asymptotic performance of inserting n nodes with equal keys in an empty tree using this strategy?

Problem 4 – Loopy linked lists

A singly list that has a loop looks like this



(If the list doesn't have a loop then the forward pointer of the last element is NULL.)

a. Can a singly linked list have more than one loop? Motivate your answer.

b. Can a doubly linked list have loops? Motivate your answer.

c. Write in pseudocode an algorithm that determines whether a singly linked list has a loop. The algorithm should run in $\mathcal{O}(n)$ time and constant space, where n is the number of elements in the list.

Hint: start at the beginning of the list with two pointers that traverse the list moving at different rates.

d. For extra credit. Write an algorithm that also determines the number of elements in the list.

Assume that the lists in this problem are unsorted.

In both **c** and **d**, start each algorithm with a short description of the idea of the algorithm and the meaning of the variables. Example (not to be taken as a hint for the solution!!): "SUCC: I first find the minimum element of the list L and store its address in x, ..."