Homework 3

Due 09/25/2024

September 19, 2024

1. Give pseudocode for an algorithm that accepts a stack stk of size n and an integer k between 0 and n that modifies stk so that its top k elements are reversed, using only a queue for storage. (You may not declare any variables other than a single queue.)

2. Give pseudocode for an algorithm that accepts a stack stk of size n and integers i and j such that $0 \leq i \leq j \leq n$ and modifies stk so that all of its entries between indexes i and j (including j but excluding i) are reversed, using only a queue for storage.

   Note that if $i = 0$, the resulting stack will have the top j entries in reverse order. Also, if $j = i$ or $j = i + 1$, the stack should retain all elements in the same order.

   Hint: you may call your solution to problem 1 as a subroutine.

3. Give pseudocode for an algorithm that accepts a stack stk of size n and integers i and j such that $1 \leq i \leq j \leq n$ and modifies stk so that indexes i and j (counting from the top) are swapped, using only a queue for storage.

   Hint: you may call your solution to problem 2 as a subroutine.

4. Give pseudocode for a post-order tree traversal for an m-ary tree stored in firstChild–nextSibling form.

5. Answer the following questions about a modification to a BST so that it can return the min value in $\Theta(1)$ time.

   (a) What additional data members would you store in the BST? Your answer should indicate what these data members represent and how they can be used to identify the min in $\Theta(1)$.

   (b) How would you modify insertion so that your data members are updated appropriately? Pseudocode for a basic BST insertion has been given below. Your modification should exhibit the same time complexity.

   Your answer may be an English-language description of how you would modify this pseudocode, or it may be the modified pseudocode.

```
   Input: ins: new data value to insert
   Input: node: current BST node (originally root)
   Input: BST.Insert
 1 if ins ≤ node.value then
 2 |   if node.left = nil then
 3 |   |   node.left = Node(ins)
 4 |   |   node.left.parent = node
 5 |   else
 6 |   |   BST.Insert(ins, node.left)
 7 |   end
 8 else
 9 |   if node.right = nil then
10 |   |   node.right = Node(ins)
11 |   |   node.right.parent = node
12 |   else
13 |   |   BST.Insert(ins, node.right)
14 |   end
15 end
```

(c) How would you modify deletion? Your modification should exhibit the same time complexity as ordinary deletion.

```
   Input: victim: BST node to be deleted
   Input: BST.Delete
 1 Let children be the number of non-nil children of victim
 2 if children = 0 then
 3 |   if victim.parent ≠ nil then
 4 |   |   Set victim.parent's matching child pointer to nil
 5 |   end
 6 |   delete victim
 7 else if children = 1 then
 8 |   Let child be the child of victim
 9 |   if victim.parent ≠ nil then
10 |   |   Set victim.parent's matching child pointer to child
11 |   end
12 |   child.parent = victim.parent
13 |   delete victim
14 else
15 |   lhsMax = victim.left
16 |   while lhsMax.right ≠ nil do
17 |   |   lhsMax = lhsMax.right
18 |   end
19 |   Swap victim.data and lhsMax.data
20 |   BST.Delete(lhsMax)
21 end
```