Sample Solution for CS3323 Fall 2006 Assignment 3 (40 marks)

Due Monday, Oct. 30, by 5pm.

- Assignments should be handed in by placing them in the CS3323 bin on E level of Gillin Hall.

---

1. **(10 marks)** Given a new element e_1 and an element e_2 stored in a singly linked list, design an algorithm to insert e_1 before e_2 (if e_2 is not in the list, insert e_1 at the tail). Give the pseudo code of your algorithm and analyze its time complexity.

   Solution:

```
Algorithm InsertBefore (S, e1, e2 ):
        Input: A singly linked list S, two elements e1 and e2
        Output: S with e1 before e2
        v ← new Node()
        v.setElement(e1)
        previous ← null
        current ← S.head
        while (current != null) and (current.getElement()!= e2)
                previous ← current
                current ← current.getNext()
        if (current.getElement() = e2 )
          if (previous = null)
            S.head ← v
            v.next ← current
          else
            previous.next ← v
            v.next ← current
        else
          if (current = null)
            S.head ← v
            v.next ← null
          else
            v.next ← null
            previous.next ← v
```

S.size ← S.size +1

2. **(5 marks)** What is the output from the following sequence of priority queue ADT methods: insert(5,A), insert(4,B), insert(7,I),insert(1,D), removeMin(), insert(3,J), insert(6,L), removeMin(), removeMin(), insert(8,G), removeMin(), insert(2,H), removeMin(), removeMin().

   Solution: $(1, D)$, $(3, J)$, $(4, B)$, $(5, A)$, $(2, H)$, $(6, L)$.

3. (a) **(5 marks)** Illustrate the execution of the selection-sort algorithm implemented by a Priority Queue on the following input sequence: $(22, 15, 36, 44, 10, 3, 9, 13, 29, 25)$.
   (b) **(5 marks)** Illustrate the execution of the insertion-sort algorithm implemented by a Priority Queue on the input sequence given in part

   Solution:

   (a)

	Sequence S	**Priority Queue P**
Input	$(22, 15, 36, 44, 10, 3, 9, 13, 29, 25)$	$()$
1.a	$(15, 36, 44, 10, 3, 9, 13, 29, 25)$	(22)
1.b	$(36, 44, 10, 3, 9, 13, 29, 25)$	$(22, 15)$
1.c	$(44, 10, 3, 9, 13, 29, 25)$	$(22, 15, 36)$
1.d	$(10, 3, 9, 13, 29, 25)$	$(22, 15, 36, 44)$
1.e	$(3, 9, 13, 29, 25)$	$(22, 15, 36, 44, 10)$
1.f	$(9, 13, 29, 25)$	$(22, 15, 36, 44, 10, 3)$
1.g	$(13, 29, 25)$	$(22, 15, 36, 44, 10, 3, 9)$
1.h	$(29, 25)$	$(22, 15, 36, 44, 10, 3, 9, 13)$
1.i	(25)	$(22, 15, 36, 44, 10, 3, 9, 13, 29)$
1.j	$()$	$(22, 15, 36, 44, 10, 3, 9, 13, 29, 25)$
2.a	(3)	$(22, 15, 36, 44, 10, 9, 13, 29, 25)$
2.b	$(3, 9)$	$(22, 15, 36, 44, 10, 13, 29, 25)$
2.c	$(3, 9, 10)$	$(22, 15, 36, 44, 13, 29, 25)$
2.d	$(3, 9, 10, 13)$	$(22, 15, 36, 44, 29, 25)$
2.e	$(3, 9, 10, 13, 15)$	$(22, 36, 44, 29, 25)$
2.f	$(3, 9, 10, 13, 15, 22)$	$(36, 44, 29, 25)$
2.g	$(3, 9, 10, 13, 15, 22, 25)$	$(36, 44, 29)$
2.h	$(3, 9, 10, 13, 15, 22, 25, 29)$	$(36, 44)$
2.i	$(3, 9, 10, 13, 15, 22, 25, 29, 36)$	(44)
2.j	$(3, 9, 10, 13, 15, 22, 25, 29, 36, 44)$	$()$

(b)

	Sequence S	**Priority Queue P**
Input	$(22, 15, 36, 44, 10, 3, 9, 13, 29, 25)$	$()$
1.a	$(15, 36, 44, 10, 3, 9, 13, 29, 25)$	(22)
1.b	$(36, 44, 10, 3, 9, 13, 29, 25)$	$(15, 22)$
1.c	$(44, 10, 3, 9, 13, 29, 25)$	$(15, 22, 36)$
1.d	$(10, 3, 9, 13, 29, 25)$	$(15, 22, 36, 44)$
1.e	$(3, 9, 13, 29, 25)$	$(10, 15, 22, 36, 44)$
1.f	$(9, 13, 29, 25)$	$(3, 10, 15, 22, 36, 44)$
1.g	$(13, 29, 25)$	$(3, 9, 10, 15, 22, 36, 44)$
1.h	$(29, 25)$	$(3, 9, 10, 13, 15, 22, 36, 44)$
1.i	(25)	$(3, 9, 10, 13, 15, 22, 29, 36, 44)$
1.j	$()$	$(3, 9, 10, 13, 15, 22, 25, 29, 36, 44)$
2.a	(3)	$(9, 10, 13, 15, 22, 25, 29, 36, 44)$
2.b	$(3, 9)$	$(10, 13, 15, 22, 25, 29, 36, 44)$
2.c	$(3, 9, 10)$	$(13, 15, 22, 25, 29, 36, 44)$
2.d	$(3, 9, 10, 13)$	$(15, 22, 25, 29, 36, 44)$
2.e	$(3, 9, 10, 13, 15)$	$(22, 25, 29, 36, 44)$
2.f	$(3, 9, 10, 13, 15, 22)$	$(25, 29, 36, 44)$
2.g	$(3, 9, 10, 13, 15, 22, 25)$	$(29, 36, 44)$
2.h	$(3, 9, 10, 13, 15, 22, 25, 29)$	$(36, 44)$
2.i	$(3, 9, 10, 13, 15, 22, 25, 29, 36)$	(44)
2.j	$(3, 9, 10, 13, 15, 22, 25, 29, 36, 44)$	$()$

4. **(5 marks)** Draw a single binary tree T such that

   - each internal node of T stores a single character
   - a *preorder* traversal of T yields EXAMFUN; and
   - a *inorder* traversal of T yields MAFXUEN

   Solution:

5. **(10 marks)** Give an $O(n)$-time algorithm for computing the depth of each node of a tree T, where n is the number of nodes of T. Assume the existence of methods setDepth(v,d) and getDepth(v) that run in $O(1)$ time.

   Solution: Depth of a node v is equal to depth of the parent of v incremented by one. Therefore, our algorithm will mimic preorder traversal algorithm (each parent has to

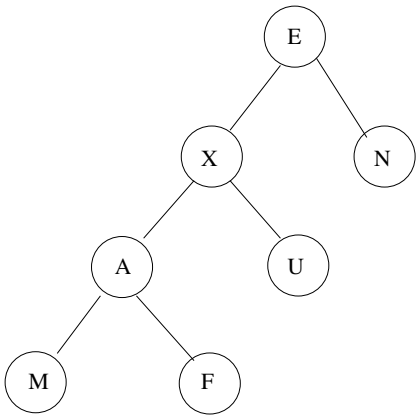

Figure 1: The solution tree for Question 4.

be "processed" before its children). To compute depth of each node of T, the following algorithm should be called with T and $T.root()$.

Algorithm ComputeDepth (T, v):
Input: tree T; v is a node of T
Output: depth of each node in the subtree of T rooted at v
if (T.isRoot(v)) **then**
  setDepth(v, 0)
else
  setDepth(v, 1 + getDepth(T.parent(v)))
children $\leftarrow$ T.children(v)
while (*children*.hasNext()) **do**
    child $\leftarrow$ *children*.next()
    ComputeDepth(T, *child*)

We assume that children(v) runs in $O(c_v)$ worst-case time where c_v is the number of children of v. Then every line of the **if** statement takes $O(1)$. Assignment to *children* takes $O(c_v)$ time, and the **while** loop gets executed c_v times also, and recursively computes depths of all the nodes in the subtree rooted at v. Excluding these recursive calls, ComputeDepth takes $O(c_v)$ time.

How much time does ComputeDepth(T, T.root()) take? Note that for each node v of T, we get to call ComputeDepth(T,v) exactly once (since each node has at most one parent and we start out with the root of T). Therefore the time it takes to execute ComputeDepth(T, T.root()) is equal to the combined time it takes to execute non recursive part of ComputeDepth on each node of v. This is equal to $\sum_{v \in T} O(c_v) = O(\sum_{v \in T} c_v)$. So, this is $O(n)$.

An alternative solution for this problem involves an auxiliary data structure that can perform $O(1)$ time insertions and deletions (for example, stacks and queues would qualify). The following is the algorithm that uses a stack to compute depth. To make the following algorithm use a queue, we have to start out with an initially empty queue and replace all push and pop methods with enqueue and dequeue methods correspondingly.

Algorithm ComputeDepth(T)
Input: tree T
Output: depth of each node in the subtree of T
current $\leftarrow$ T.root()
$S \leftarrow$ an empty Stack
S.push(*current*)
while (! S.isEmpty()) **do**
  current $\leftarrow$ S.pop()
  if (T.isRoot(*current*)) **then**
    setDepth(*current*, 0)
  else
    setDepth(v, 1 + getDepth(T.parent(v)))
  children $\leftarrow$ T.children(v)
  while (*children*.hasNext()) **do**
    S.push(*children*.next())

The reason we can use both a queue and a stack here is because it is irrelevant in which order we process the children of each node, as long as we process the node itself before any of its children. We ensure this by exracting a node from and _then_ inserting its children into whatever auxiliary data structure we are using.

Note that each node gets inserted and removed exactly once. The number of insertions bounds from above the number of times the outer **while** loop gets executed and the number of removals bounds the number of times the number of iterations of the inner **while** loop. Everything else is simple operations and $O(1)$ methods, so this algorithm is also $O(n)$.