Data Structures and Algorithms – 2018

Assignment 1 – 0% of Continuous Assessment Mark

Deadline: 5pm Monday 29th January, via Canvas

The University’s Code of Practice on Assessment and Feedback says “Formative feedback should be provided on the first piece of work of a particular type in a programme/module”. For that reason, this first assignment will be marked (with appropriate additional feedback) as usual, but the mark will not contribute towards the final mark awarded for this module.

Question 1 (10 marks)

You need to insert the numbers 2, 4, 3, 7, one at a time in that order into an initially empty queue.

Represent that process using the standard constructors push and EmptyQueue.

\[
\text{push (7, push (3, push (4, push (2, EmptyQueue))))}
\]

Show, in the standard two-cell notation, the resulting queue.

\[
\begin{array}{c}
\text{2} \\
\text{4} \\
\text{3} \\
\text{7}
\end{array}
\]

What is the result of the operation top on that queue?

2

What is the result of the operation pop on the original queue you created?

\[
\begin{array}{c}
\text{4} \\
\text{3} \\
\text{7}
\end{array}
\]

What is the result of the operation pop followed by pop followed by top on the original queue you created?

3

Question 2 (18 marks)

In the lecture notes (Section 3.2) a recursive procedure \texttt{last(L)} was defined that returns the
last item in the given list L. By making the simplest possible modification to that procedure, create a recursive procedure \texttt{secondlast(L)} that returns the second to last item in a given list L.

\begin{verbatim}
secondlast(L) {
    if ( isEmpty(L) )
        error('Error: Empty list in procedure secondlast')
    elseif ( isEmpty(rest(L)) )
        error('Error: Short list in procedure secondlast')
    elseif ( isEmpty(rest(rest(L)) )
        return first(L)
    else return secondlast(rest(L))
}
\end{verbatim}

What is the time complexity of your algorithm?

Linear in n, or O(n), where n is the length of the list.

Now carry out a more general modification of the \texttt{last(L)} procedure to give a recursive procedure \texttt{getItem(i,L)} that returns the \textit{i}th item in the given list \texttt{L}, where \textit{i} is an integer greater than zero. [Hint: See Lecture Notes Section 6.8.]

\begin{verbatim}
getItem(i,L) {
    if ( isEmpty(L) )
        error('Error: List is too short.')
    elseif ( i == 1 )
        return first(L)
    else return getItem(i-1,rest(L))
}
\end{verbatim}

\textbf{Question 3} (10 marks)

Often one needs to check whether two given lists are the equal, i.e. contain the same items in the same order. Write a recursive procedure \texttt{equalList(L1,L2)} that returns \texttt{true} if the two given lists \texttt{L1} and \texttt{L2} are the same, and \texttt{false} if they are not. The only procedures it may call are the standard primitive list operators \texttt{first}, \texttt{rest} and \texttt{isEmpty}.

\begin{verbatim}
equalList(L1,L2) {
    if ( isEmpty(L1) and isEmpty(L2) )
        return true
    elseif ( isEmpty(L1) or isEmpty(L2) )
        return false
    elseif ( first(L1) != first(L2) )
        return false
    else return equalList(rest(L1),rest(L2))
}
\end{verbatim}

What is the time complexity of your algorithm?

Linear in \textit{n}, or \texttt{O(n)}, where \textit{n} is the length of the shortest list.
Question 4 (16 marks)

A quadtree was defined in the lectures in terms of primitive constructors baseQT(value) and makeQT(luqt,ruqt,llqt,rlqt), selectors lu(qt), ll(qt), ru(qt) and rl(qt), and condition isValue(qt). Suppose a gray-scale picture is represented by such a quadtree with values in the range 0…255, for example:

![Quadtree example](image)

Write a procedure flip(qt), that uses the above primitive quadtree operators, to flip the picture about the vertical line through its centre.

```
flip(qt) {
  if ( isValue(qt) )
    return qt
  else return makeQT(flip(ru(qt)),flip(lu(qt)),
    flip(rl(qt)),flip(ll(qt)) )
}
```

Write another procedure avevalue(qt), that uses the above primitive quadtree operators, to return the average gray-scale value across the whole picture.

```
avevalue(qt) {
  if (isValue(qt) )
    return qt
  else return (avevalue(lu(qt)) + avevalue(ru(qt))
    + avevalue(ll(qt)) + avevalue(rl(qt)))/4
}
```

Question 5 (12 marks)

It is often important to know whether two given binary trees are the identical. Write a recursive procedure equalBinTree(bt1,bt2) which returns true if the given binary trees bt1 and bt2 are the same, and false otherwise. You can assume that you have access to the standard primitive binary tree procedures root(bt), left(bt), right(bt) and isempty(bt). [Hint: Remember that you can only directly test the equality of numbers, e.g. node values.]
equalBinTree(t1,t2) {
    if ( isEmpty(t1) and isEmpty(t2) )
        return true
    elseif ( isEmpty(t1) or isEmpty(t2) )
        return false
    else return ( (root(t1) == root(t2) ) and
                  equalBinTree(left(t1),left(t2)) and
                  equalBinTree(right(t1),right(t2)) )
}

What is the time complexity of your algorithm?

Linear in n, or O(n), where n is the number of nodes in the smallest tree.

Question 6 (16 marks)

Suppose you have access to the primitive binary tree procedures root(bt), left(bt), right(bt) and isempty(bt). Write a procedure isLeaf(bt) using them that returns true if the binary tree bt is a leaf node, and false if it is not.

isLeaf(bt) {
    if ( isempty(bt) )
        return false
    else return ( isempty(left(bt)) and isempty(right(bt)) )
}

Then write a recursive procedure numLeaves(bt) that returns the number of leaves in the given binary tree bt. It is only allowed to call the above primitive binary tree procedures and your isLeaf(bt) procedure.

numLeaves(bt) {
    if( isempty(bt) )
        return 0
    elseif ( isLeaf(bt) )
        return 1
    else return (numLeaves(left(bt)) + numLeaves(right(bt)))
}

Question 7 (18 marks)

How many different orderings of the four numbers \{1, 2, 3, 4\} are there?

4! = 24

By considering all those possible orderings, draw all possible binary search trees of size four with nodes labeled by the four numbers \{1, 2, 3, 4\}. After discarding any duplicate trees, how many different binary search trees of size four are there?

Out of the 24, there are 14 different trees:
For each different tree, state its height, how many leaf nodes it has, and whether it is perfectly balanced.

Height: 3, 3, 2, 3; 2, 2, 2; 3, 3, 2, 3.
Leafs: 1, 1, 2, 2; 1, 2, 2; 1, 1, 2, 1.
Balanced: N, N, N, N; Y, Y, Y; N, N, N, N.