2020 Semester 1
COMP20007 Design of Algorithms
School of Computing and Information Systems
The University of Melbourne

Assignment 1 Solutions

Problem 1

```
function EvaluateExpression(Input String S[0, ..., n − 1])
    Values ← NewStack()
    Operators ← NewStack()

    for i ← 0 to n − 1 do

        if S[i] is a digit then
            // A digit should either be the first element in S or follow an operator or a "("
            if i > 0 and S[i − 1] a digit or ")" then
                return NotWellFormed

            val ← S[i] interpreted as an integer
            Push(Values, val)

        else if S[i] is one of "+", "-", "*", "/" or "(" then
            Push(Operators, S[i])

        else if S[i] is ")" then
            // Now we find the value of this bracketed expression and push it to the value stack
            if length of Values < 2 or length of Operators < 2 then
                return NotWellFormed

            // Since y is the top of the stack the operator should be applied in the order x op y
            y ← Pop(Values)
            x ← Pop(Values)
            op ← Pop(Operators)
            if op not one of "+", "-", "*", "/" then
                return NotWellFormed
            result ← x op y
            Push(Values, result)
            if Pop(Operators) is not "(" then
                return NotWellFormed

        else
            // We've found a character which is not an operator or digit
            return NotWellFormed

    if length of Values ≠ 1 or length of Operators ≠ 0 then
        return NotWellFormed
    return Pop(Values)
```

Problem 3b

function IsSingleRunPossible
  $G \leftarrow$ read graph into adjacency lists
  $n \leftarrow$ number of nodes in G
  run DFS from 0 keeping track of the popping order
  if less than n nodes explored **then**
    return False
  topological order $\leftarrow$ Reverse(*popping order*)
  return IsPath(*topological order*)

// An array of nodes is a path if for each $i \in \{0, \ldots, n-2\}$ there is an edge between $A[i]$ and $A[i+1]$
function IsPath(graph G, array of nodes $A[0, \ldots, n-1]$)
  for $i \leftarrow 0$ to $n-2$ **do**
    if $A[i+1]$ not in $A[i]$s adjacency list **then**
      return False
  return True

Explanation

First, we know that if we can't reach all nodes in a DFS from node 0 then there must be unreachable nodes from the top of the mountain and no run can trim all trees.

If a single run is possible in the graph then it will be a path starting at 0 which goes downhill and visits every node.

Let $u_0, u_1, \ldots, u_{n-1}$ be this path. Then this path must be a topological ordering since if it was not then there would be an edge (u_i, u_j) with $j < i$, which would mean there is a cycle in the graph (contradicting the fact that it is a DAG). Additionally this topological ordering must be unique, as exchanging any two of the nodes would introduce an edge going right to left in the order.

So if the graph has a single run we will find it by finding a topological ordering, and this ordering will be a path (in the sense that successive nodes will have an edge between them). If we find a topological ordering which doesn't satisfy this property then there is no single run possible.

Time Complexity

- Reading a graph into an adjacency list takes $O(n)$ time to create the lists and $O(1)$ time per edge to insert the node into the adjacency list. So $O(n+m)$ to read the graph.
- Running a depth-first search involves exploring each node at most once and running a for loop over the out edges from each node. When we explore each node we do a constant amount of work, as well as looping over each of its out-edges (to explore unvisited nodes), so the time complexity of the DFS is,

  $$(O(1) + \deg(0)) + (O(1) + \deg(1)) + \cdots + (O(1) + \deg(n-1)) = O(n+m)$$

  Here deg is the out-degree of each node. Note that use a doubly-linked list to keep track of popping order, so each insertion is also $O(1)$ time.
- Reversing the popping order will take $O(n)$ time.
- Checking whether an ordering is a path requires checking all the out-edges for each node in the path. Similarly to the depth first search this takes $O(n+m)$ time.

So the algorithm runs in $O(n+m)$ time as required.