Theory Problem 5. Basic Data Structuress

Moyuan Huang
Department of Computer Science
University of California, San Diego
moh008@eng.ucsd.edu

1 Computing Tree Height

1.1 Algorithm

1.1.1 Pseudo Code

Algorithm 1 **computing tree height**

Input:
$parent_1, parent_2, ..., parent_{n1}$, A sequence of nodes.
Output:
$height$

$index \leftarrow 0, root \leftarrow NULL$
for i from 1 to n-1 **do**
  if $parent_i = -1$ **then**
    $root \leftarrow nodes[index]$
  else
    $nodes[index].setParent(parent_i)$
  end if
end for
$height \leftarrow INT\_MIN$
$compute(root, height, 1)$
return $height$

Algorithm 2 compute($root$, $height$, cur)

if root.empty() **then**
  $height$ = max($height$¡ cur)
  return
end if
for child in children **do**
  compute(child, height, cur + 1)
end for
return

1.1.2 description

The algorithm first iterates through all the nodes to get their parents' index, and set the parent nodes' index to the corresponding child.(This is done by calling $setParent()$ function) If a node has no parent, then it is said to be the root node and we thus set the root node index to it.
Then we call the $compute()$ function to recursively find the maximum depth of the tree, and return.

1.2 correctness

The key to the solution is the $compute()$ function. It recursively traverse the tree by depth. Once it reaches the bottom of the tree(this is done in Alg. 2 line $1-4$), it compares the height to the global height and records the bigger one.

1.3 time complexity

Line $2-8$ runs n-1 time and has O(n) time complexity.
Line 10 runs at most n time and thus has O(n) time complexity.
together the algorithm has O(n) + O(n) = O(n) time complexity.