Name: Grade:

Quick Test: Trees and Data Structures

Master MoSIG — Algorithms and Program Design

Florent Bouchez Tichadou — Gwenaël Delaval

October 2nd, 2015

Duration: 20 minutes.
All documents forbidden.

Reminder: given a binary tree `t`, the left child can be accessed with `t.left`, the right child with `t.right`, the value on the node at its root with `t.value`. If it exists, the parent node can be accessed with `t.parent`.

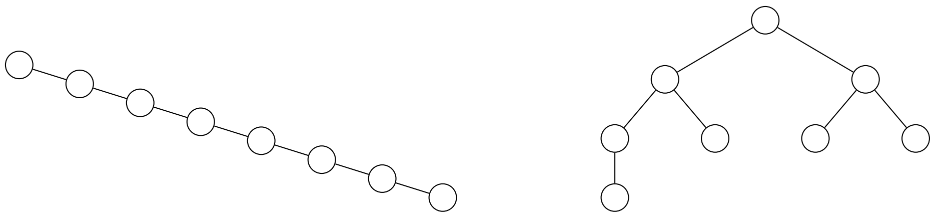
Exercise 1 (Binary Search Trees)

Reminder: a Binary Search Tree (BST) is a binary tree with the following property: each node n contains a value v and every value in the left subtree of n is less than v, and every value in the right subtree of n is greater than v.

Question 1.1 Given the number of nodes n in a binary tree, what are respectively the **minimum** and **maximum** height of the tree? Give one example in both cases for $n = 8$.

Solution to 1: The height of a tree is the length of the longest path from the root to a leaf, i.e., the number of edges on this path.

In that case, the maximum height is $n - 1$ for a chain-like tree (7 on the example on the left) and the minimum height is attained in the case of a balanced tree and is $\lfloor \log n \rfloor$ (3 on the example on the right).



Question 1.2 Write an algorithm which, given a BST and a value, finds whether this value belongs to this BST.

Solution to 1:

```
Search(v: value; t: tree)
if t = NULL then
    return False
else if v = t.value then
    return True
else if v < t.value then
    return Search(v, t.left)
else
    return Search(v, t.right)
```

Question 1.3 What is the complexity of this algorithm? Justify your answer.

Solution to 9: The algorithm starts at the root, and in the worst case (e.g., when the value does not belong to the tree) ends at a leaf, following a direct path.

The complexity is then $O(\text{height of the tree})$. If the tree is highly unbalanced it can be $O(n)$, but if the tree is balanced it is $O(\log n)$.

Exercise 2 (Paths in trees)

Question 2.1 In a binary tree, what is the **maximum** distance possible between any two nodes (i.e., the length of the path between those nodes)? What if the tree is balanced? Give an example where this distance is reached in both cases.

Solution to 2: The maximum distance is $n - 1$, when the tree is just a chain. (See left example of question 1.1).

However in general, the maximum distance is between two leaves, hence it is at most $2 \times$ height. If the tree is balanced, the maximum distance is then $O(\log n)$. More precisely, for instance with a complete tree, it is exactly $2 \times \lfloor \log n \rfloor$, see for instance such a tree with 31 nodes: $\lfloor \log 31 \rfloor = 4$, and any longest path (from a leaf to another leaf) is of length 8 (there are 5 levels).

Question 2.2 Write an algorithm which, given a BST and two values, prints the shortest path between these two values in this BST (precondition: the two values belong to the BST).

Solution to 2: There are of course multiple solutions.

A simple one would be to modify the `Search()` function defined above to that it returns a pointer to the node where we can find the value, and then use a modified version of the LCA (lowest common ancestor) algorithm that keeps the path in memory. However this algorithm will walk in the worst case four times the height of the tree, while it is possible to do it in only two at worst (not counting the time to print the path).

We suppose we are looking for a path between x and y, with $x < y$. We will be using a list for the path on which we can `Push()` at the head and `Append()` at the tail. The idea is to go down the BST from the root searching for the LCA. It will be the first node `t` such that $x \leqslant$ `t.value` $\leqslant y$. Now, we look for x on `t.left` while `Push`ing the values encountered on the path. Similarly, we look for y on `t.right` while `Append`ing the values encountered on the path. Special care is taken so that it works whenever x or y is exactly `t.value`.

When this is done, the path list contains all the values of the BST in the path from x to y. The complexity in $O(2 \times \text{height} + |\text{path}|) = O(\text{height})$.

```
Search-Path(x: value, y: value, t: tree)
if t = NULL then Abort;
if y < t.value then
    Search-Path(x, y, t.left)
else if x > t.value then
    Search-Path(x, y, t.right)
else
    /* We found the LCA */
    path ← ∅;
    if x ≠ t.value then
        path ← Search-Add(x, t.left, Push, path);
    path ← Search-Add(y, t, Append, path);
    print path;
```

```
Search-Add(v: value, t: tree, f: function, path: list)
if t = NULL then Abort;
f(t.value, path) ;    /* push or append value */
if v = t.value then
    return path
else if v < t.value then
    return Search-Add(v, t.left, f, path)
else
    return Search-Add(v, t.right, f, path)
```