# Homework #5
# Introduction to Algorithms/Algorithms 1
# 600.363/463
# Spring 2013

**Due on:** Tuesday, March 4th, 5pm
**Late submissions:** will NOT be accepted
**Format:** Please start each problem on a new page.
**Where to submit:** On blackboard, under student assessment
Please type your answers; handwritten assignments will not be accepted.
To get full credit, your answers must be explained clearly,
with enough details and rigorous proofs.

March 5, 2014

## 1 Problem 1 (20 points)

Suppose we have a complete binary tree $T = (V, E)$ with $n$ nodes. Since $T$ is complete, we must have $n = 2^d - 1$ for some non-negative integer $d$. Suppose we have a function $f : V \mapsto \mathbb{R}$, and $f(u) \neq f(v)$ for all $u \neq v$, $u, v \in V$. A node $u \in V$ is *golden* if $f(u)$ is such that $f(u) > f(v)$ for all $v \in V$ for which $(u, v) \in E$. That is, node $u$ is golden if the value assigned to it by function $f$ is strictly greater than the value of any neighboring node. Given any node $u \in V$, we can find out the value of $f(u)$ by *querying* $u$. Devise an algorithm that finds a golden node in $T$ using $O(\log n)$ queries. Prove the correctness of your algorithm, and prove that it does indeed require $O(\log n)$ queries.

Answer: First of all, in any such tree as the one we are given, at least one golden node exists, since the function $f$ assigns distinct values to each of the nodes in $T$ (and any set of distinct numbers must have a global maximum). The following algorithm will suffice to find some such node. Note that we have overloaded notation slightly. The function $\mathrm{FindGoldenNode}$ when called on a node (rather than on a tree) means to call the function on the subtree rooted at that node.

$t \leftarrow \text{ROOTNODE}(T)$
**if** $\text{IS-LEAF-NODE}(t)$ **then**
  **return** $t$
**end if**
$l \leftarrow \text{LEFTCHILD}(t)$
$r \leftarrow \text{RIGHTCHILD}(t)$
**if** $f(t) > f(l)$ **and** $f(t) > f(r)$ **then**
  **return** $t$
**else**
  $u \leftarrow \arg\max_{s \in \{l,r\}} f(s)$
  **return** $\text{FindGoldenNode}(u)$
**end if**

Runtime: at each iteration, the algorithm either terminates and returns a node or makes a recursive call on a node at the next-lowest depth of the tree. Since $T$ is assumed to be a complete tree on $n = 2^d - 1$ nodes, it has $O(\log n)$ depth, and thus the algorithm requires at most $O(\log n)$ operations, since the algorithm terminates when it reaches a leaf.

Correctness: it will suffice to show that any node returned by the algorithm is a golden node. Suppose node $t \in T$ is returned by the algorithm. If this is the case, it is because either (a) $t$ is a leaf node or (b) both children of $t$, call them $u$ and $v$, are such that $f(t) > f(u)$ and $f(t) > f(v)$. If (a), then we need only show that the parent of $t$, call it $p$, satisfies $f(t) > f(p)$. This follows from construction of the algorithm. If it had been the case that $f(p) > f(t)$, then depending on the value at the other child of $p$, call it $t'$, we would have either returned $p$ (if $f(p) > f(t')$) or returned $t'$ (if $f(t) < f(p) < f(t')$). If (b) holds, then let us first note that the fact that we descended to node $t$ indicates that $t$'s parent, call it $p$, if it existed all, was such that $f(p) < f(t)$, since otherwise we would have either returned $p$ or descended to its other child. Thus, $t$ takes a larger value under $f$ than both of its children (by construction of the algorithm) and a larger value than its parent (if that parent exists). Thus, $t$ takes a larger value under $f$ than any of its neighbors, and satisfies the definition of a golden node.

## 2 Problem 2 (20 points)

Devise an algorithm that takes an undirected graph $G = (V, E)$ encoded as an adjacency list and returns `True` if a graph $G$ is a tree and `False` if not. Prove the correctness of your algorithm and prove its runtime. Your algorithm should run in $O(|V| + |E|)$ time for full credit. You may assume that the input graph $G$ does not contain any double edges and does not contain any loops– that is, there is at most one edge between any pair of nodes $u$ and $v$, and there are no edges of the form $(u, u)$.

Answer: a minor adaptation of DFS will suffice. We perform depth-first search starting at any node. When we follow edge $(u, v)$ out of $u$ during the search, if $v$ has already been visited but not yet removed from the stack (i.e., $v$ is grey) we return `False`, because we just found a cycle. If the stack becomes empty before we've visited all the nodes in the graph, then the graph is disconnected, so we must return `False`. If we visit all nodes without returning `False`, return `True`.

Correctness: Correctness follows from the definition of DFS search and from the parentheses theorem. We will try to visit a grey node during DFS if and only if we have not finished processing that node, i.e., the node we are currently at is a descendant of the grey node. But this happens precisely when there is a path from the grey node to the node we are currently at, and an edge from the node we are currently at to the grey node, i.e., a cycle. Our algorithm returns `False` if there are nodes not reachable from the node we started at– this is correct, since a tree cannot be disconnected. Note that it would also have sufficed to count the number of edges– if $|E| \geq n$, then we can't possibly have a tree.

Runtime: depth-first search requires $O(|V| + |E|)$ time. Our modifications to DFS added no more than constant work to any step of the algorithm, so this runtime is maintained.

## Optional exercises

Solve the following problems and exercises from CLRS: 22.2-5, 22.2-8, 22.4-5, 22.5-4, 22-4, 22-1.