Question 1. (20 points) Let G be an n-vertex directed graph that has the following properties:

- For every vertex v, the number of other vertices that appear on the adjacency list of v is exactly the same as the number of times that v appears in the adjacency lists of other vertices (in other words, for every vertex v the number of directed edges that have v as head is equal to the number of directed edges that have v as tail).
- The undirected version of G (the undirected graph obtained from G by ignoring edge directions) is connected.

Does G have to be strongly connected, or can it be otherwise? Justify your answer by providing a proof if your answer is "Yes", giving a counterexample if your answer is "No".

Question 2. (30 points) Suppose you are given a set S of n points $(x_1, y_1), \ldots, (x_n, y_n)$ where the x_is and y_is are distinct (i.e., no two are equal). We use p_i as a shorthand for the point (x_i, y_i). A point p_i is said to *dominate* another point p_j if $x_j < x_i$ and $y_j < y_i$. Two points are *comparable* if one of them dominates the other, and are *incomparable* if neither of them dominates the other. For example, the point $(9.2, 3.3)$ dominates the point $(7.1, 1.2)$, but the two points $(9.2, 3.3)$ and $(4.5, 6.8)$ are incomparable. Let α be the number of points of a largest subset of S in which all the points are pairwise comparable. Let β be the number of points of a largest subset of S in which all the points are pairwise incomparable.

1. Give an $O(n \log n)$ time algorithm for computing α by making use of the longest increasing subsequence (LIS) algorithm we covered in class.
2. Repeat the above for computing β.
3. Prove that $\max\{\alpha, \beta\} \geq \sqrt{n}$. (*Hint:* Use the pigeonhole principle.)

Question 3. (20 points) Let T be a (not necessarily complete or balanced) n-leaf binary tree, of height h, whose leaves initially contain n data items $d_1, \ldots, d_n$ (not in sorted order); the ith leftmost leaf initially contains d_i. Assume that h is much smaller than n. We would like to support the following operations, in $O(h)$ time per operation.

1. $Increment(i, j, x)$ where $1 \leq i < j \leq n$: Adds (in the sense of arithmetic addition) x to the value of the item d_k associated with the kth leftmost leaf, for all k such that $i < k < j$.
2. $Decrement(i, j, x)$ where $1 \leq i < j \leq n$: Subtracts x from the value of the item d_k associated with the kth leftmost leaf, for all k such that $i < k < j$.
3. $Value(i)$: Returns the current value d_i associated with the ith leftmost leaf. Even though such an operation can be done in constant time in the initial tree T (i.e., before there have been any *Increment* or *Decrement* operation), by indexing into the ith leaf and reading the d_i value in it, this will no longer be a constant-time operation after there have been many *Increment* and *Decrement* operations.

Note that $Increment(i, j, x)$ and $Decrement(i, j, x)$ do not return anything: Their only effect is on later $Value(i)$ operations (which are the only operations that return a value to the outside world). This is why, even though the number of values affected by these two operations could be proportional to n, it is possible to process them in $O(h)$ time. This is done by storing at each node v a field $\delta(v)$ that is initially zero if v is not a leaf. If v is a leaf then $\delta(v)$ is initialized to be the item stored at that leaf. We assume an array is available whose ith entry points to the ith leaf (so that the ith leaf can be accessed in constant time).

- Explain in detail how $Increment(i, j, x)$ and $Decrement(i, j, x)$ are implemented, in $O(h)$ time, so as to maintain the following invatriant: "$Value(i)$ equals the sum of all the $\delta(v)$ values on the path between the root and the ith leftmost leaf".

Note that the above implies that a query $Value(i)$ is performed in $O(h)$, time by adding all the $\delta(v)$ values on the path between the root and the ith leaf.

Question 4. (30 points) In this problem we consider the exact pattern matching problem when the alphabet consists of the 5 symbols $\{a, b, c, d, \#\}$ where the special symbol # matches any symbol (including itself). For example, if $T = ab\#aca\#ab\#a$ and $P = da\#ac$ then P occurs starting at position 3 in T. Give an $O(n \log n)$ time algorithm for determining whether a pattern P of length n occurs in a text T of length $2n$, assuming that # symbols can occur (possibly $O(n)$ times) in both T and P.

Hint. Use convolution, and beware of double-counting.

Date due: November 21, 2013