CSCE 586 HW 3

Marvin Newlin

1 November 2018

1. **Chapter 5 Problem 2:** Recall the problem of finding the number of inversions. As in the text, we are given a sequence of n numbers $a_1, \ldots, a_n$, which we assume are all distinct, and we define an inversion to be a pair $i < j$ such that$a_i > a_j$.
   We motivated the problem of counting inversions as a good measure of how different two orderings are. However, one might feel that this measure is too sensitive. Let's call a pair a significant inversion if $i < j$ and $a_i > 2a_j$. Give an $O(n \log n)$ algorithm to count the number of significant inversions between two orderings.
   Solution:

   Description: We can just modify the merge-and-count part of the original sort-and-count algorithm from the text that allows us to count the number of inversions (Kleingberg, Tardos 224).This is the modification we make:

   Change the check of $b_i < a_j$ to $2b_j < a_i$. This count of significant inversions is then what is returned.

   Runtime Analysis: Our modification takes the same amount of time as the original check so the overall algorithm runtime is not modified. Therefore it runs in $O(n \log n)$ time in all cases.

   Proof of Correctness: The original algorithm counts the number of inversions so our algorithm with the modification returns the counted number of significant inversions and it only increments the *Count* variable when $a_i > 2b_j$. Thus, since the original Sort-and-Count algorithm is correct and ours is a modification, it works as well.

   Difficulty: 4, Time Spent: 30 min

2. **Chapter 5 Problem 6:** Consider an n-node complete binary tree T, where $n = 2^d - 1$ for some d. Each node v of T is labeled with a real number x_v. You may assume that the real numbers labeling the nodes are

all distinct. A node v of T is a local minimum if the label x_v is less than the label x_w for all nodes w that are joined to v by an edge.

You are given such a complete binary tree T, but the labeling is only specified in the following implicit way: for each node v, you can determine the value x_v by probing the node v. Show how to find a local minimum of T using only $O(\log n)$ probes to the nodes of T.

Solution: **Description:**

```
Notation: r is the root node of Tree T
          c_l is the left child of node r
          c_r is the right child of node r
          val(r) indicates the value of the node r

Algorithm FindLocalMin(r)
Input: root node r of tree T
Output: Node whose value is a local minimum
     IF r has no child nodes
          return r
     ELSE IF val(r) > val(c_l)
          return FindLocalMin(c_l)
     ELSE IF val(r) > val(c_r)
          return FindLocalMin(c_l)
     ELSE
          return r
END
```

Proof of Correctness

Termination: This algorithm is guaranteed to terminate because eventually if we continue to recurse, we will reach a leaf node with no children that algorithm will return the leaf node.

Local Minimum: Suppose our algorithm doesn't find a local minimum. Then we returned a node v with value x_v such that $x_v > x_w$ for some w connected to v. But since we returned x_v then there are 2 cases:
Case 1: v was a leaf node. If this is the case then to have reached v its value was less than its parent so it is a local minimum so this case produces a contradiction.
Case 2: x_v is greater than the value of one of it's children (since its not a

local minimum). But we returned the value of node v so in the algorithm the check of the child node values returned that x_v was less than the value of both child nodes. Therefore v is a local minimum. Thus we have a contradiction.

Since both cases produce a contradiction, then v is indeed a local minimum.

Runtime Analysis: In the best case, the tree we're given has been sorted such that the root node is less than both child nodes, so then we only have to probe the root node and its children to find a local min. In this case then, the runtime is $\Theta(1)$.

In the average and worst case, we must recurse to some level of the tree T. Since T is a complete binary tree with $n = 2^d - 1$ nodes then its height is given by $d = \log(n)$. Our algorithm always probes $O(d)$ nodes because we have to probe the root and its children until we find a local minimum so we always recurse to some fraction of the depth d (or all the way down in the worst case). Thus, our algorithm runs in $\Theta(\log n)$ time in the average and worst cases.

Difficulty 3, Time Spent: 20 min

3. **Chapter 5 Problem 7:** Suppose now that you're given an $n \times n$ grid graph G. (An $n \times n$ grid graph is just the adjacency graph of an $n \times n$ chessboard. To be completely precise, it is a graph whose node set is the set of all ordered pairs of natural numbers (i, j), where $1 \leq i \leq n$ and $1 \leq j \leq n$; the nodes (i, j) and (k, l) are joined by an edge if and only if $|i - k| + |j - l| = 1$.)
We use some of the terminology of the previous question. Again, each node v is labeled by a real number x_v; you may assume that all these labels are distinct. Show how to find a local minimum of G using only $O(n)$ probes to the nodes of G. (Note that G has n^2 nodes.)
Solution:
Description:

```
Notation: B is the boundary of graph G
          C is the union of middle row and column of graph G

Algorithm FindnxnLocalMin(G)
Input: n x n grid graph G
Output: Node whose value is a local minimum
     IF G has 1 node
          return the node
     v ← node whose value is a local min of B ∪ C
     IF v is a local min of all its neighbors (includes all of G)
          return v
     ELSE
          v' ← neighbor node of v with x_v' < x_v
          G' ← quadrant of G containing v' (including B ∪ C)
          return FindnxnLocalMin(G')
END
```

Proof of Correctness
Termination: The algorithm will eventually terminate once the graph is contained of one node. Since the size of the graph is halved at each turn, it will eventually become a one node graph if it continues recursing.

Local Minimum: If we find a minimum of the union of the border and center sections then checking it's neighbor nodes we find that it is indeed

a local minimum then return a true local minimum. If we recurse into the quadrant containing v' then it too must contain a local minimum since either v' is that local minimum or it neighbors another node in the quadrant that is less than it. Regardless, the node is guaranteed to be in that quadrant.

Runtime Analysis: Let $T(n)$ be the number of nodes probed by our algorithm. At each iteration, we the recurrence relation $T(n) = O(n) + T(\frac{n}{2})$. The $O(n)$ comes from searching the single columns and rows of the border and center and then when we recurse into the quadrant we are then searching a graph that is $\frac{n}{2} \times \frac{n}{2}$. Thus, by the master theorem, the runtime of our algorithm is $O(n)$ (Kleingberg, Tardos 219).

Difficulty 7, Time Spent: 60 min

4. **References**

Kleinberg, Jon, and Tardos Eva. Algorithm Design. 1st ed., Pearson, 2006. Pp 224, 219.