CIS 502 - Algorithms

Fall 2014 Midterm 1 Solutions

Problem 1:(20 Points) You are given a binary tree which satisfies the (min) heap property: the parent is smaller than both the children. It is not a search tree and need not be balanced. The number of elements in the heap is n. At each node, you can only see the value at the node and follow the left or right pointers to the respective children.

You want to know if the k^{th} smallest element in the heap is less than the value y. Give a worst case $O(k)$ time algorithm. The algorithm cannot depend on n. Your algorithm only needs to output yes/no. You **do not** need to output that element.

You need to give reasons why your answer is correct.

Solution: The task reduces to **counting** the number of nodes with value less than y and deciding if that count is more than k (yes answer to the question) or less than k (no answer to the question).

We use DFS/BFS - except that (1) we do not explore nodes with value larger or equal to y (since all descendants of that node will also have values larger or equal to y). and (2) we stop the DFS/BFS after we have explored k nodes (using a global counter). The running time of BFS/DFS is $O(m' + n')$ but here $n' \leq k$ since we stop the process as soon as we discover k nodes. Also $m' = n' - 1 \leq k - 1$ since the underlying graph is a tree. Thus the algorithm runs in time $O(k)$.

Problem 2:(30 Points) You have an undirected graph with weights $w_{e'}$ on edges e'. All weights $w_{e'} \geq 0$. You have already computed an MST T for the graph. You are now told that the weight of some single edge e should actually be w'_e (but still ≥ 0). For each of the four cases below give the most efficient (as low running time as possible) algorithm for computing the MST under the new weights. Give a **short** argument for correctness for each.

(a) (10 Points) $w'_e < w_e$ and $e \in T$.

(b) (5 Points) $w'_e < w_e$ and $e \notin T$.

(c) (10 Points) $w'_e > w_e$ and $e \in T$.

(d) (5 Points) $w'_e > w_e$ and $e \notin T$.

Solution: **For part (a):** the tree does not change. Because the edge remains the minimum edge across the cut corresponding to the cut in the Kruskal's algorithm when this edge was added. A crisper reason: (suppose not) if some other tree T' became the new minimum, what was that tree doing in the original MST computation? If T' does not have e, we have a contradiction immediately. If T' contained e, then T and T' changed by the same amount and why is T' the new minimum now? Contradiction.

For part (b): We add the edge e to T and find the unique cycle (using favorite algorithm, say DFS). The running time is $O(n)$ because a tree has $n - 1$ edges. We now delete the maximum edge along this cycle and that is the new tree. Follows from the cycle property.

For part (c): We delete the edge e from T and find the two (connected) components $S, V \setminus S$ of the tree T. This is $O(n)$. We now find the minimum edge across this cut in time $O(m)$ time. Based on Kruskal's algorithm, this is the new MST.

For part (d): The MST does not change. Some bunch of other trees became more expensive – why should the MST change?

Problem 3: (25 Points) A subsequence is defined to be **palindromic** if it is the same when it is read left-to-right or right-to-left. A sequence can have many palindromic subsequences. For example

$$a, b, a, f, a, b, a, c, d, e, d, c$$

Has several such subsequences $aba, aa, baab, cdedc$, etc. Give an efficient (as low running time as possible) algorithm to find the **longest** palindromic subsequence of a sequence $a_1, a_2, \ldots, a_n$. For full credit give an $O(n^2)$ time algorithm.

A subsequence need not be a substring (contiguous) as the example shows.

Solution: A palindromic subsequence is a subsequence of both $X = a_1, a_2, \ldots, a_n$ and $X^R = a_n, a_{n-1}, \ldots, a_1$. We can find the longest common subsequence by the following dynamic program **(this is necessary, because this is an exercise in the book and does not have a provided solution)**.

Let $LCS[i, j]$ be the (length of the) longest common subsequence of $X_1, \ldots, X_i$ and $Y_1, \ldots, Y_j$.

$$LCS[i,j] = \max \begin{cases} LCS[i-1,j] \\ LCS[i,j-1] \\ 1 + LCS[i-1,j-1] & \text{(provided } X_i = Y_j\text{, otherwise this option does not exist)} \end{cases}$$

$LCS[i, 0] = LCS[0, j] = 0$ for all i, j. Final answer is $LCS[|X|, |Y|]$. The size of the table is $O(n^2)$ and each entry is updated in $O(1)$ time.

A simpler and more direct solution is to let $PAL[i, j]$ be the (length of the) longest palindromic sequence.

$$PAL[i,j] = \max \begin{cases} PAL[i+1,j] \\ PAL[i,j-1] \\ 2 + PAL[i+1,j-1] & \text{(provided } X_i = X_j\text{, otherwise this option does not exist)} \end{cases}$$

$PAL[i, j] = 0$ for $i > j$ and $PAL[i, i] = 1$ for all i. Final answer is $PAL[1, n]$.