CSCI-GA.1170-001/002 Fundamental Algorithms November 25, 2014

Solutions to Problem 1 of Homework 10 (12 (+4) points)

Name: Keeyon Ebrahimi *Due: Wednesday, November 26*

(a) (3 points) Assume directed graph G is acyclic. Show that G has at least one vertex v having no outgoing edges.

Solution: Imagine G as a graph that has all vertexes with outgoing edges, and that after traversing down all Vertexes except for one, we do not have a cycle. We now have to analyze the last non visited Vertex, which we will label as L. Now, if every node has an outgoing edge, then node L will also have an outgoing edge. Because all nodes other than L have already been visited, and L has an outgoing edge, then no matter what, L will have an outgoing node to a node that has already been visited, therefore creating a cyclic Graph. This shows that in order to have an acyclic graph, we need to have a Vertex with no outgoing edges. □

(b) (5 points) Consider the following greedy algorithm for topological sort of a directed graph G: "Find a vertex v with no outgoing edges. If no such v exists, output 'cyclic'. Else put v as the last vertex in the topological sort, remove v from G (by also removing all incoming edges to v), and recurse on the remaining graph G' on $(n-1)$ vertices". If this algorithm is correct, prove it, else give a counter-example.

Solution: Few things to consider

- If we have a vertex with no outgoing edges at any time, we will have an cyclic graph, as shown by $PartA$

- By taking a vertex completely out of an acyclic graph, we will never all of a sudden create a cyclic graph.

- If there is any acyclic section of G, when all of G is acyclic.

  With the above statements, we know that the very first removal will not create an cyclic graph out of an acyclic graph, and that it tests for the graph being cyclic or not correctly. By removing this node that has no outgoing edges, we will not be disrupting the tree structure that G has. After removing the vertex with no edges, we now run this algorithm on the newly created tree because if there is any section of G that is acyclic, then the whole graph becomes acyclic. We keep running this test until we remove all Vertexes, which we now know correctly answers if there is any acyclic section or not.

□

(c) (4 (+4) points) It is easy to implement the above algorithm in time $O(mn)$. Show how to implement it in time $O(n^2)$. For **extra credit**, do it in time $O(m + n)$.

Solution: We will be running a modified Topological sort using a DFS.
After the DFS Topological sort, elements with the lowest end time should have no children if we are dealing with a cyclic G. So we run a DFS and sort in reverse finish time order and place them each into a stack. This gives us G in topological sort in running time $O(m + n)$. Now we pop from the stack, giving us the element with the smallest end time, which we will label as L and we see if this child has any children. If it does have children, then we know that we are dealing with an cyclic G, for in an acyclic G, the vertex with the smallest end time will have no children. If L does not have any children, we remove this L Vertex from G, and also remove all incoming edges. This process will happen in constant time because we are storing G in a $2x2$ array.

We have now accurately implemented the initial step of the algorithm in time $O(m + n)$. After a successful removal of L, we once again pop from our stack, giving us the new Vertex with the least ending time. With this vertex we run our constant time checks and repeat.

To make this happen, we need to run our topological sort and place these all into a stack, which will take $O(m+2n)$. This is because the topological sort takes $O(m+n)$ and placing in the stack takes time $O(n)$. We then pop from the stack, make checks on children and update the problem base on this info, which only takes constant time.
Our running time is now $O(m + 2n)$, and we are also going to pop from the stack $0(n)$ times, meaning our total running time for this algorithm will be $O(m + 3n)$, which is equivalent to $O(m + n)$ □

CSCI-GA.1170-001/002 Fundamental Algorithms November 25, 2014

Solutions to Problem 2 of Homework 10 (6 points)

Name: Keeyon Ebrahimi *Due: Wednesday, November 26*

Recall, MST finds a spanning sub-tree T of the original graph minimizing the sum of edge weights in T: $\sum_{e \in T} w(e)$. Consider a related problem MST$'$ which attempts to find a spanning sub-tree T' of the original graph minimizing the maximum edge weight in T': $\sum_{e \in T'} w(e)$. Show that the solution T to MST is also an optimal solution T' to MST$'$, and vice versa.

Solution: ***************** INSERT PROBLEM 2 SOLUTION HERE *************** $\square$

CSCI-GA.1170-001/002 Fundamental Algorithms November 25, 2014

Solutions to Problem 3 of Homework 10 (10 points)

Name: Keeyon Ebrahimi *Due: Wednesday, November 26*

(a) (4 points) Assume that all edge weights of an undirected graph G are equal to the same number w. Design the fastest algorithm you can to compute the MST of G. Argue the correctness of the algorithm and state its run-time. Is it faster than the standard $O(m + n \log n)$ run-time of Prim?

Solution: ***************** INSERT PROBLEM 3a SOLUTION HERE ***************
$\square$

(b) (6 points) Now assume the all the edge weights are equal to w, except for a single edge $e' = (u', v')$ whose weight is w' (note, w' might be either larger or smaller than w). Show how to modify your solution in part (a) to compute the MST of G. What is the running time of your algorithm and how does it compare to the run-time you obtained in part (a) (or standard Prim)?

Solution: ***************** INSERT PROBLEM 3b SOLUTION HERE ***************
$\square$

CSCI-GA.1170-001/002 Fundamental Algorithms November 25, 2014

Solutions to Problem 4 of Homework 10 (16 points)

Name: Keeyon Ebrahimi *Due: Wednesday, November 26*

Assume all edge weights in G are integers from 1 to w.

(a) (8 points) Show how to modify Prim's algorithm to achieve running time $O(m + nw)$. Hence, if $w = O(1)$, you get optimal time $O(m + n)$.

Solution: ***************** INSERT PROBLEM 4a SOLUTION HERE *************** □

(b) (4 points) Now assume $w = n$, so that the previous solution in part (a) is no longer faster than standard. Show how to modify Kruscal's algorithm instead of Prim's, so that it now takes time $O(m + n \log n)$, instead of $O(m \log n)$.

Solution: ***************** INSERT PROBLEM 4b SOLUTION HERE *************** □

(c) (4 points) What is the largest w for which you can still maintain the $O(m + n \log n)$ run-time in part b? In particular, can you tolerate $w = n^2$? $w = n^3$?

Solution: ***************** INSERT PROBLEM 4c SOLUTION HERE *************** □