4005-800 Algorithms

Homework 3

Christopher Wood

April 6, 2012

PROBLEM 1. *CLRS 22.1-1*

Solution.

Given an adjacency list representation of a *directed* graph, the only way to determine the adjacent vertices of each vertex $v \in V$ is to traverse the entire adjacency list of v. Using this fact, we can easily determine the time complexity of computing out-degree and in-degree of every vertex as follows.

1. To compute the out-degree for a single vertex $u \in V$, we must count the total amount of vertices contained within the adjacency list of u. To do this, we must traverse the the adjacency list for u, which amounts to traversing all outgoing edges starting from u as well. Therefore, in order to compute the out-degree of every vertex in a directed graph, we must repeat this procedure for every vertex, which means that we will traverse over every vertex and every edge in the graph. Thus, the time complexity is to compute the out-degree of every vertex is $\Theta(V + E)$.

2. To compute the in-degree for a single vertex $uinV$, we must inspect all adjacency lists for every vertex $v \in V$ to determine if u is adjacent to v. Only after a complete traversal of the entire adjacency list representation can we be certain that we have examined all possible edges leading to u, and thus can compute the in-degree. A naive approach to extend this to all vertices would be to repeat this search procedure V times, amounting in a time complexity of $O(V(V + E))$. However, if we use an auxiliary data structure to keep track of the in-degree of every vertex $u \in V$, we need only perform this adjacency list traversal once, incrementing the in-degree of each vertex v that is visited in the traversal. Therefore, just as with the out-degree calculation, the time complexity is simply $\Theta(V + E)$.

PROBLEM 2-a. *CLRS 22.2-2*

Solution.

After running the breadth-first search on the undirected graph shown below (INSERT IMAGE), using vertex u as the source, we arrive at the following values for d and π.

TODO: copy from paper and insert images of before/after graphs

$$d = TODO$$
$$\pi = TODO$$

PROBLEM 2-b. *CLRS 22.2-3*

Solution.

The purpose of the gray colors in BFS is to indicate vertices that have been discovered in the traversal. Without this color, we are guaranteed that the only vertices *added* to the queue Q are those that are white upon discovery. Furthermore, the only way to avoid being added to the queue Q is for the vertex under consideration to be colored black. Therefore, without the intermediate gray color, it is possible for a vertex v to be added to Q multiple times before its adjacency list is completely traversed and it is assigned a color black. We now show that these duplicate entries in Q do not change the behavior or result of the BFS procedure.

Let v be a vertex in that was just added to Q for the first time. If at a later point in time (after v was dequeued from Q) v is re-added to Q, where $v, v_1, v_2, ..., v_i, v$ is a sequence of vertices in Q, we know the following must be true:

1. v will be colored black by the time the second instance of v is dequeued from Q. This is because the BFS routine will enqueue all neighbors of v before continuing to the next vertex to dequeue and process in a breadth-first manner, at which point v would be colored black.

2. All vertices $v_1, v_2, ..., v_i$ that exist in Q will also be colored black because the BFS routine must have dequeued all of them, enqueued their white-colored adjacent vertices, and then finished by coloring all of them black.

Therefore, using these facts we know that upon the next visit to v (i.e. after it is dequeued the next time), all of the vertices adjacent to v that were contained within $\{v_1, v_2, ..., v_i\}$ will not be added to Q again because they are colored black. Furthermore, using this same argument, any additional adjacant vertices $\{u : u \notin \{v_1, v_2, ..., v_i\}\}$ that are enqueued into Q because they are not colored black must be duplicates.

Finally, since after every traversal of the adjacency list of a vertex v we are guaranteed to color the vertex black we know that any duplicate vertices that appear in Q will not have any impact on the results of the BFS traversal. Furthermore, since the removal of the gray color does not change the breadth-first behavior of the traversal (because it does not modify the behavior of Q), we can conclude that the removal of the gray color altogether does not change the behavior of BFS; it simply poses the risk of a longer time complexity.

PROBLEM 2-c. *CLRS 22.2-5*

Solution.

In the correctness proof provided in the textbook, it is shown that $u.d = \delta(s, u)$, meaning that $u.d$ is always equal to the length of the shortest path between s and u upon termination of the BFS routine. Furthermore, the proof goes on to show that the BFS routine will always produce the shortest path lengths between a start vertex s and all other vertices $u \in V$ for any graph G. Finally, since the order of vertices in the adjacency list representation of a graph G does not have any effect on the topolgy of G (i.e. the actual edges that exist in the graph), we can now conclude that the value $u.d$ assigned to a vertex u is independent of the order in which the vertices appear in each adjacency list for G.

TODO: show image of different paths/trees based on neighbor ordering (from paper drawing)

PROBLEM 3. *CLRS 22.3-7*

Solution.

The code for the DFS algorithm that uses a stack for its depth-first traversal is shown below:

ALGORITHM 1: Stack-Based DFS

1: TODO: copy from textbook here

PROBLEM 4-a. $T(n) = aT(n-1) + bn$

Solution.

PROBLEM 4-b. $T(n) = aT(n-1) + bn\ log(n)$

Solution.

PROBLEM 4-c. $T(n) = aT(n-1) + bn^c$

Solution.

PROBLEM 4-d. $T(n) = aT(n/2) + bn^c$

Solution.

PROBLEM 5.

Solution.

TODO: discuss the maximum routine that requires NO comparisons!
max(a,b) = (a+b)/2 + —(a-b)/2—