CS3510-A: Design and Analysis of Algorithms, Spring 2021

Homework-2

January 26, 2021

DUE DATE: Tuesday, February 2, 1l:59pm

Note-1: Your homework solutions should be electronically formatted as a single PDF document that you will upload on Gradescope. If you have to include some handwritten parts, please make sure that they are very clearly written and that you include them as high resolution images.

Note-2: Please think twice before you copy a solution from another student or resource (book, web site, etc). It is not worth the risk and embarrassment.

Note-3: You need to **explain/justify** your answers. Do not expect full credit if you just state the correct answer.

Note-4: You will get 2 extra points if you submit electronically typed solutions instead of hand-written.

Problem-1 (30 points)

You are planning the seating arrangement for a wedding. You are given the list of guests, V , and a lookup-table T where $T[v]$ for $v \in V$ is a list of guests that v knows. If u knows v, then v also knows u. You are required to arrange the seating such that any guest at a dining table knows every other guest sitting at the same dining table either directly or through some other guests sitting at the same dining table. For example, if x knows y and y knows z and z knows w, then x,y,z,w can sit at the same dining table. Describe an efficient algorithm that takes as input the set V and lookup-table T and returns the minimum number of dining tables needed to achieve a valid seating. You can assume a check in the lookup-table is an $O(1)$ operation.

Note 1: Please provide a detailed description of your solution in plain English or in pseudocode. Also, explain the time complexity and correctness of the prescribed algorithm.

Note 2: You are also allowed to use BFS and DFS as blackbox algorithms (you do not need to explain how it works if you use it) so long as you detail the inputs and outputs to the algorithms as well as any modifications you make.

Problem-1 Response:

Pseudocode:

Data: V: list of guests (nodes)
T: table of guests that each guest knows

Initialization:
S = set of all unexplored nodes (starts with all nodes in V); tables = 0;

while *S !=* ∅ **do** *v = guest* ∈ *S*
BFS(v) (run BFS starting from v); remove all guests found through BFS from v; tables += 1;
end return *tables*

Efficiency:

n: number of nodes in the graph m: number of edges in the graph

Runtime: T(n, m) = n * (T(BFS(n, m))) Algorithm will run BFS at once per node.

Big-O: O(n + m)
This algorithm will have the same worst case run-time as BFS (because this algorithm is just repeatedly running BFS until there are no unexplored nodes).

Problem-2 (35 points)

Problem-2 Response:

1) Run a BFS traversal over all nodes from *v*. Keep track of distances from *v* during the traversal. Also keep track of parent nodes for each node (nodes that are connected to the current node, and all equally closer to the *v* node).

2) All possible paths can be determined by traversing the parent nodes (tracked from step 1), starting from the *w* node. This can be done by recursively exploring all the possible paths from *w* to *v* along the parent nodes.

Justification:
All the paths found will have the same length, by nature of the parent node tracking method. No node's parents will be any further from *v*. If *d* is the distance from *v* to a given node *n*, then all parents of *n* will have a distance of $d - 1$. Therefore, all paths along the parent tree will be equal to the shortest possible distance from *v* to *w*.

Runtime:

n: number of nodes

m: number of edges

$T(n,m) = n + 2m$ (same as BFS)
$= O(n + m)$

Problem-3 (35 points)

There's a natural intuition that two nodes that are far apart in a communication network—separated by many hops – have a more tenuous connection than two nodes that are close together. There are a number of algorithmic results that are based to some extent on different ways of making this notion precise. Here's one that involves the susceptibility of paths to the deletion of nodes.

Suppose there exists a communication network $N = (V,E)$ with n nodes (or "hubs"). N contains two hubs s and t such that the distance between s and t is strictly greater than $n/2$. Show that there must exist some hub v (v 6= s,t) such that deleting v from the network destroys all $s-t$ circuits (paths). In other words, the communication network obtained by deleting v contains no path from s to t. Give a linear time algorithm to find such a hub v. Justify the correctness and running time of your algorithm.

Note 1: Please provide a detailed description of your solution in plain English or in pseudocode.

Note 2: You are also allowed to use BFS and DFS as blackbox algorithms (you do not need to explain how it works if you use it) so long as you detail the inputs and outputs to the algorithms as well as any modifications you make.

Problem-3 Response:

1) Pick a random point x.
2) Set A to be the furthest point in the graph from x (uses BFS).
3) Set B to be the furthest point in the graph from A (uses BFS).
4) Identify all paths from A-B (uses BFS).
5) Return the node(s) that appears in all paths from A-B.

Justification:

If two nodes s and t are the farthest two in a graph, then any one node that can prevent them from being connected would have to be a hub node (removing it would result in at least one of the two nodes being disconnected form the rest of the graph). If any node appears in all possible paths between s and t, then removing that node would result in all paths between s and t being destroyed. If there were no such node, then the graph has no hubs.

Runtime:

n: number of nodes

m: number of edges

$T(n, m) = 3 * (n + 2m)$ *(BFS is run 3 times)*

$= O(n + m)$