Colorado CSCI 5454: Algorithms
Homework 2

Instructor: Bo Waggoner
Due: September 12, 2019 at 11:59pm
Turn in electronically via Gradescope.

Problem 1 (8 points)

A *tree* is an undirected graph that (1) is connected, meaning every vertex is reachable from every other vertex, and (2) has no simple cycles. Trees are broadly used in data structures, as we will see in this problem.

Usually, when given a tree, we are also given a *root* r, one of the vertices of the tree. Consider any non-root vertex w and the path from r to w. The second-to-last vertex on this path is the *parent* of w, and w is a *child* of v. A vertex with no children is a *leaf*.

For parts (a) and (b), you do not need to prove correctness or running time of your algorithms (but it should be clear from your algorithm and the course material).

Part a (2 points) The *level* of a node is its distance from the root. Give an $O(n+m)$ time algorithm to output an ordering of the vertices by level, i.e. the root, followed by all level-1 vertices, followed by all level-2 vertices, ...

Part b (2 points) Give a linear-time $O(n+m)$ algorithm to output an ordering of the vertices where each parent appears immediately after all of its children.

Part c (4 points) Suppose that initially, each vertex v has a number $c[v]$ stored in array c. Give a recursive $O(n+m)$ time algorithm to output an array d, where $d[v]$ is the maximum number in the subtree rooted at v. (This is the tree consisting of v as the root, all of v's children, their children, and so on.) For this part, prove correctness and running time of your algorithm. *Hint for correctness: use induction, with the leaves of the tree as the base case.*

Problem 2 (4 points)

Dasgupta, Papadimitriou, Vazirani Problem 4.8 (Prof. F. Lake problem).

Problem 3 (12 points)

The following question was asked by a friend of mine and his young child:

How many rainbow paths are there (red to purple)?

A step in the path must move from one peg to another peg below (directly below or diagonally adjacent). For example, for each orange peg, there happen to be exactly two ways to reach it from a red peg.

In this question, we will design an efficient algorithm to solve this problem, first as pictured, then in general graphs.

Part a (2 points) Suppose we had a board with n colors (so it is n rows high). Argue briefly that for some peg configurations, the number of paths can be very large, for example $\Omega(2^n)$. We conclude that counting the paths one-by-one is not practical.

Part b (1 point) In the image, for each of the three yellow pegs (left to right), how many red $\rightarrow$ orange $\rightarrow$ yellow paths are there ending at that peg?

Part c (1 point) In the image, how many red $\rightarrow$ orange $\rightarrow$ yellow $\rightarrow$ green paths are there ending at the leftmost green peg? *Hint: don't count them all; use your answer to Part b!*

Part d (4 points) Given a pad of post-it notes and a pen, describe a strategy for computing the number of rainbow paths. Your algorithm should be asymptotically much more efficient that counting the paths one by one. Briefly argue correctness. *Hint: start by posting a "1" on every red peg, then move on to the orange ones...*

Part e (4 points) We are given a directed acyclic graph (DAG) $G = (V, E)$. We are given a set of *start* vertices $S \subseteq V$ and a set of *end* vertices $T \subseteq V$. Give an efficient algorithm to output the number of paths that start in S and end in T. You should argue correctness and running time. You may assume the graph is represented either as an adjacency list or an adjacency matrix.[1]

Problem 4 (10 points)

This is a programming assignment. Download the undirected, unweighted Texas road network from here: `https://snap.stanford.edu/data/roadNet-TX.html` The format is as follows: the first four lines of the text file are comments and can be ignored. The remaining lines each describe an edge. Each of these lines contains two integers separated by whitespace, representing the two vertices of the edge. The vertex names range from zero to 1393382, inclusive.

Write code in any programming language and use it to solve the following problems. Attach your source code to the end of the assignment (PDF format is fine).

Part a (2 points) Read in and store the graph, e.g. as an adjacency list.[2] What is the largest degree of any vertex? What is the average degree?

Part b (4 points) (Theory problem.) Describe an algorithm that, given an undirected, unweighted graph, a vertex v, and an integer d, counts how many vertices are at distance d or less from v. The time and space complexity should only depend on the final answer, not on n or m. (Example: on input v and $d = 7$, suppose only 20 vertices are at distance 7 or less from v. Then the algorithm should only execute on the order of 20 iterations of any loop, even if n and m are over one million.)

Part c (4 points) Implement your algorithm and run it on the Texas road network, picking 10 random initial vertices. For each, report its index (name) and how many vertices are at distance: 5 or less; 10 or less; and 50 or less. (Can your code handle 500?)

[1] You may also assume the word size in this RAM model is large enough to hold the total number of paths in the graph.

[2] An adjacency matrix is *not* recommended, unless you have 15 TB free disk space and a lot of free time!