CS 330, Fall 2020, Midterm Review Problems
October 22, 2020

Problem 1 Graph adjacency list.

1. Consider the DAG $G(V, E)$. As we have learned, one way of finding the topological order of its nodes is to repeatedly remove a node with (currently) zero indegree. (We proved in class that such a node always exists.) Discuss the efficient implementation of this algorithm; specifically how to keep track of the indegrees of nodes and efficiently find the 0 degree nodes. For this you maintain the adjacency list A and an additional array D of length n, where index i corresponds to node i. What should you store in D?

2. You are given the adjacency list of the *undirected* graph $G(V, E)$. The *square* of G is the graph $G^2(V, E^2)$, where there is an edge from node u to v, if either there is a (u, v) edge in E or there is a length-2 path from u to v. How long does it take to compute G^2 from the adjacency list?

Problem 2 Shortest paths. Suppose you have access to a blackbox implementation of Dijkstra's algorithm. That is, the algorithm A takes as input a directed, weighted graph $G(V, E, w)$ and a source node s. It returns the distance of each node from s as well as the shortest paths tree.

This algorithm is called a blackbox because you have no knowledge of the internal workings of this algorithm. (Similar to when you're using an off-the-shelf software package.) In your solution you shouldn't think about how it works, only the fact that you have access to its output.

1. Describe how to use A to run BFS on a directed, *unweighted* graph $G'(V', E')$.

2. The second time you are given a directed, *weighted* graph $G(V, E, w)$. the weight on the edges correspond to time. In addition, each node v has a delay $d(v)$ for which amount of time the traveller has to stay in that node. How would you use A to find the shortest duration time paths from s to the other nodes, when idle times are also considered for the total duration of a paths?

Problem 3 MSTs.

Let $G(V, E, w)$ be an undirected weighted graph. Let T_1 be an MST of G. Suppose that after removing the edges in T_1 from G, the remaining graph is still connected. LET T_2 be an MST in the remaining graph. Argue whether the following statements are true.

1. For any cut C in G the second lightest edge in C is in T_2.

2. For any cycle C, the heaviest edge in C is neither in T_1 nor in T_2.

Problem 4 Divide and Conquer

Consider the following modified version of mergeSort, 3-wayMergeSort. The goal is to sort a list of n numbers. But instead of dividing the list in two, we divide the list in to three equal parts. We sort each of the three subarrays (by recursively calling 3-way MergeSort), then merge the three arrays.

1. describe in short how to do the 3-way merge. Note, that you can find the minimum of 3 items by performing 2 comparisons. Write the recurrence relation for the 3-way mergeSort. Compute its asymptotic running time.

2. describe the k-way merge. Again, you can find the minimum of k items with $k - 1$ comparisons. Write the recurrence relation and compute the asymptotic running time for the k-way mergeSort. Note, that k is part of the input, not a constant!