CS2020: Data Structures and Algorithms (Accelerated)

Problems 1–3

For Week10

1 Lecture 18 - Algorithms on DAG/Basic DP

1.1 Verifying Your Understanding of Various Algorithms on DAG

Let's start this DG with something light. Your DG leader will draw one **medium-sized random weighted connected DAG** of about 10-15 vertices (the number of vertices cannot be too small, otherwise this exercise is too trivial).

Then, three random students will be selected to perform SS Shortest Paths, SS Longest Paths on DAG, and Counting Paths on DAG, respectively, as shown in Lecture18. For each of these task, you may want to use the easier/more visual approach, i.e. find any valid topological order of the DAG first – $O(V + E)$, then perform the requested task also in linear time – $O(V + E)$.

1.2 Verifying Your Understanding of LIS Problem/Solution

During Lecture18, Steven linked the classical Longest Increasing Subsequence (LIS) to the SS Longest Paths on DAG problem. Your DG leader will write one **random sequence of integers** of about 10-15 integers. Then a random student is selected to draw the *implicit LIS DAG* of this sequence and then find the LIS using either via graph way (reduction to the longest paths on DAG) or via Dynamic Programming (see slide 40 of Lecture18).

1.3 On Why the SS Longest Paths on General Graph is Hard

During Lecture18, Steven said that the SS Longest Paths (SSLP) on general graph is a hard problem (or technically, it is an NP-complete problem).

Isn't SSLP is just the reverse of SSSP? Isn't maximizing a positive value is just the same as minimizing a negative value? So, if the input graph to the longest path problem is G, then we can build another graph H which is the clone of graph G but with all its edge weights negated. Then, isn't the shortest simple path on the graph H is the longest simple path on G? As now the edge weights can be negative, can't we just use $O(VE)$ Bellman Ford's algorithm to solve this SSLP (which is reduced to SSSP)?

Discuss the problem of this idea.

1.4 Stopping the Algorithm for SSSP/SSLP on DAG one Iteration Earlier

During Lecture18, Steven asked whether the edge relax/stretch operations can be stopped *after* relaxing/stretching the *second last* vertex in the topological order? If you think it is always OK, prove it. If you think there is a counter example, show it.

1.5 The $O(n \log k)$ solution of LIS – not examinable

The $O(n^2)$ Longest Increasing Subsequence (LIS) solution shown in Lecture18 is not the best algorithm known for solving LIS. There is an $O(n \log k)$ solution where k is the length of the LIS. You can obviously browse the Internet for the well-known solution. However, if it is still unclear, you can discuss this 'greedy' + 'binary search' solution in DG.

2 Lecture 13-17 Material - Another Graph Modeling Exercise :)

Last week, you were presented with three UVa online judge problems: one SSSP problem on unweighted graph with some pruned/unused vertices (UVa 10653), one MST problem which turns out just about finding edges not eventually selected in the MST - these edges are called the *chord* edges (UVa 11747), and one finding connected component/floodfill problem with some simple sorting (UVa 10336).

This time, you are presented with another two UVa online judge problems. These two are harder problems than last week and likely the standard for CS2020 final exam. Your task is to read them and choose the best graph DS/algorithm to solve them using your knowledge from Lecture 13-17, Quiz 2, plus Recitation on Week08-10. That is, these two problems are solvable without DP technique. These modeling exercises will help you deal with the story-like questions in final exam.

1. UVa 10801 - Lift Hopping
   http://uva.onlinejudge.org/external/108/10801.html
2. UVa 11635 - Hotel Booking
   http://uva.onlinejudge.org/external/116/11635.html

After reading these problems, two students will answer these questions (same as last week).

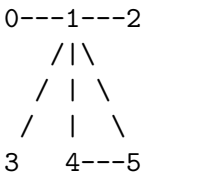
1. What is the graph (the vertices, edges, weighted/unweighted, directed/undirected, etc)?
2. What is the graph problem?
3. What should be the best graph data structure and algorithm to solve these problems? State their space/time complexity! Argue why you choose those DS/algorithm.
4. Try to sketch a *pseudo code* that implements the chosen data structure and algorithm. If possible, write a working Java code during the DG :).

3 The Weakest Link Problem - A Variant :O

In Quiz 2, there is one question about the weakest link. Now this question is also about the weakest link, but on another scenario.

We now define a weak edge as an edge e in the (originally connected) graph G that will cause graph G to be disconnected if edge e is removed. Your task is to find *all* weak edges in G.

Example: There are three weak edges: (0, 1), (1, 2), and (1, 3) in the graph G below.



Can you solve this problem 'naively' in $O(V^2 + VE)$? How?

Can you solve this problem faster, e.g. in $O(V + E)$ (to deal with a large input graph)? How?

4 Discussion on Current Problem Set (PS9)

If you still have time, you can spend some time during DG8 to discuss the solutions for problems posed in PS9 (the three short DP-related problems). Note that you are allowed to discuss! You are only prohibited from coding the solution together/copying the solution. Write down the list of collaborators in your solution.