Homework 3 – Due May 22nd, 2017 on Canvas at 1:20pm Pacific time

Homework Guidelines

Please make sure you read the collaboration policy, and write the following as the first line of your homework: "I have read and agree to the collaboration policy. ⟨`Your name`⟩." Your homework will not be graded if you do not write this.

Collaboration policy Collaboration on homework problems is permitted, but not encouraged. You are allowed to collaborate with *at most two students enrolled in the class*. You must mention the name of your collaborators clearly on the first page of your submission. Even if you collaborate, you are *expected to write and submit your own solution independent of others*, and your collaboration should be restricted to discussions only. Also, you should be able to explain your solution verbally to the course staff if required to do so. *Collaborating with any one not enrolled in the class, or taking help from any online resources for the homework problems is strictly forbidden.*

The Computer Science Department of UCSC has a zero tolerance policy for any incident of academic dishonesty. If cheating occurs, consequences within the context of the course may range from getting zero on a particular assignment, to failing the course. In addition, every case of academic dishonesty will be referred to the student's college Provost, who sets in motion an official disciplinary process. Cheating in any part of the course may lead to failing the course and suspension or dismissal from the university.

How to submit your solutions Each problem must be **typed** up separately (in at least an 11-point font) and submitted in the appropriate assignment box on the Canvas website as a PDF file.

This means that you will submit 4 separate files on Canvas, one for each problem!

You are strongly encouraged, but not required, to format your problem solutions in LaTeX. Template HW files and other LaTeXresources are posted on the course webpage. LaTeXis a free, open-source scientific document preparation system. Most technical publications in CS are prepared using this tool.

You might want to acquire a LaTeX manual or find a good online source for LaTeX documentation.

The top of each problem should include the following:

- your name,
- the acknowledgement to the collaboration policy,
- the names of all people you worked with on the problem (see the handout "Collaboration and Honesty Policy"), indicating for each person whether you gave help, received help or worked something out together, or "Collaborators: none" if you solved the problem completely alone.

Solution guidelines For problems that require you to provide an algorithm, you must give the following:

1. a precise description of the algorithm in English and, if helpful, pseudocode,
2. a proof of correctness,
3. an analysis of running time and space.

You may use algorithms from class as subroutines. You may also use any facts that we proved in class.

You should be as clear and concise as possible in your write-up of solutions. Understandability of your answer is as desirable as correctness, because communication of technical material is an important skill. A simple, direct analysis is worth more points than a convoluted one, both because it is simpler and less prone to error and because it is easier to read and understand. Points might be subtracted for illegible handwriting and for solutions that are too long. Incorrect solutions will get from 0 to 30% of the grade, depending on how far they are from a working solution. Correct solutions with possibly minor flaws will get 70 to 100%, depending on the flaws and clarity of the write up.

Assigned Problems

Following are the problems to be handed in, 25 points each. Maximum score for this homework is 100 points.

1. (**Graphs**, 2-page limit – your solutions should fit on two sides of 1 page).
   You are managing the creation of a large software product. You have the following information:

   (a) A set V of n small projects that are part of your software product.

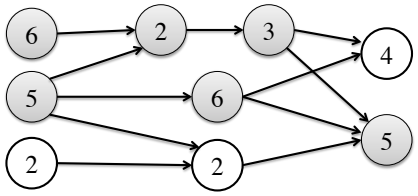
   (b) A set E of pairs of projects in V. A pair (u, v) is in E if project u must be completed before project v is started. There are no cycles in E.

   (c) For each project $u \in V$, a duration $t_u \in \mathbb{N}$. This is the amount of time the project will take from start to finish.

   You can do any number of projects in parallel, but you can't start a project before all of its predecessors (according to the list E) are completed.

   Given this information, you want to know two things: First, how soon can each of the projects be completed? Second, which projects are *critical*? We say a project is critical if increasing its duration by 1 would delay the completion of the entire product.

   For example, in the following picture, nodes represent projects, and numbers inside nodes are durations. Critical nodes are colored gray.



   Give an algorithm that takes the inputs above and returns:

   (a) For each project $u \in V$, the earliest possible completion time $c(v)$.

(b) A list of the critical projects.

Give the running time and space complexity for your algorithm. Prove that your algorithm is correct. (*Hint:* You may want to compute the completion times $c(v)$ first, then look for critical projects.)

2. (**Dynamic Programming**, 2-page limit – your solutions should fit on two sides of 1 page).

You are managing the construction of power plants along a river. As opposed to the last problem that involved construction along a river, this time around the possible sites for the power plants are given by real numbers $x_1, ..., x_n$, each of which specifies a position along the river measured in miles from its spring. Assume that the river flows in a completely straight line. Due to differences in the water flow, the position of each power plant influences its capacity for energy production. In other words, if you place a power plant at location x_i, you will produce a quantity of electricity of $r_i > 0$ MW[1].

Environmental regulations require that every pair of power plants be at least 5 miles apart. You'd like to place power plants at a subset of the sites so as to maximize total energy production, subject to this restriction. The input is given as a list of n pairs $(x_1, r_1), ..., (x_n, r_n)$ where the x_i's are sorted in increasing order.

(a) Your foreman suggests some very simple approaches to the problem, which you realize will not work well. For each of the following approaches, give an example of an input on which the approach does not find the optimal solution:

i. *Next available location:* put a power plant at $i = 1$. From then on, put a power plant at the smallest index i which is more than five miles from your most recently placed power plant.

ii. *Most profitable first*: Put a power plant at the most profitable location. From then on, place a power plant at the most profitable location not ruled out by your current power plant.

(b) Give a dynamic programming algorithm for this problem. Analyze the space and time complexity of your algorithm.

To make it easier to present your answer clearly, try to follow the steps below (as with any design process, you may have to go back and forth a bit between these steps as you work on the problem):

i. Clearly define the subproblems that you will solve recursively (note: weighted interval scheduling should be a good source of inspiration here).

ii. Give a recursive formula for the solution to a given subproblem in terms of smaller subproblems. Explain why the formula is correct.

iii. Give pseudocode for an algorithm that calculates the profit of the optimal solution. Analyze time/space and explain why the algorithm is correct (based on previous parts).

iv. Give pseudocode for an algorithm that uses the information computed in the previous part to output an optimal solution. Analyze time/space and explain why the algorithm is correct.

3. (**Dynamic Programming**, 2-page limit – your solutions should fit on two sides of 1 page).

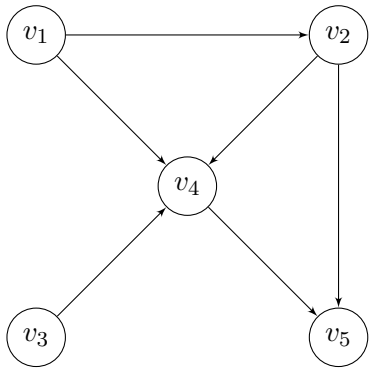
Let $G = (V, E)$ be a directed graph with nodes v_1,...,v_n. We say that G is an *ordered graph* if it has the following properties.

[1] this stands for megawatts, but don't concern yourself with the unit of measurement

(i) Edges go from a node with a lower index to a node with a higher index. In other words, every directed edge has the form (v_i, v_j) with $i < j$.

(ii) Each node with the exception of v_n has at least one edge leaving it. In other words, for every node $v_i, i = 1, 2, ..., n-1$, there is at least one edge of the form (v_i, v_j).

The length of a path is the number of edges in it. See the following example.



The correct answer for the above graph is 3: The longest bath from v_1 to v_n uses the three edges (v_1, v_2), (v_2, v_4), and (v_4, v_5). The goal in this question is to solve the following problem.

Given an ordered graph G, find the length of the longest path that begins at v_1 and ends at v_n.

(a) Show that the following algorithm does not correctly solve this problem, by giving an example of an ordered graph on which it does not return the correct answer.

```
Set a = v_1
Set L = 0
While there is an edge out of node a
    Choose the edge (a, v_j) for the smallest possible j
    Set a = v_j
    Increase L by 1
Return L as the length of the longest path
```

In your example, say both what the correct answer is and what the algorithm above finds.

(b) Give an efficient algorithm that takes an ordered graph G and returns the *length* of the longest path that begins at v_1 and ends at v_n.

4. (**Graphs**, 2-page limit – your solutions should fit on two sides of 1 page).

Suppose that you have a directed graph $G = (V, E)$ with an edge weight function w and a source vertex $s \in V$. The weights can be negative, but there are no negative weight cycles. Furthermore, assume that all edge weights are distinct (i.e. no two edges have the same weight). The single source shortest path problem is to find the shortest path distances from s to every vertex in V.

(a) Suppose that you also guaranteed that for all $v \in V$, a shortest path from s to v has increasing edge weights. Give an algorithm to find the shortest path distances from s to every vertex in V. Analyze the running time of your algorithm and explain why it is correct. For full credit, your algorithm should run in time $O(V + E \log E)$.

(b) A sequence is *bitonic* if it monotonically increases and then monotonically decreases, or if by a circular shift it monotonically increases and then monotonically decreases. For example the sequences $(1, 4, 6, 8, 3, -2)$, $(9, 2, -4, -10, -5)$, and $(1, 2, 3, 4)$ are bitonic, but $(1, 3, 12, 4, 2, 10)$ is not bitonic. Now, suppose that the sequences of edge weights along the shortest paths are no longer guaranteed to be increasing, but instead are guaranteed to be bitonic. Give a single source shortest path algorithm, explain why it is correct, and analyze its running time. For full credit, your algorithm should run in time $O(V + E \log E)$.