REVIEW PROBLEMS: NOT TO BE TURNED IN

(25) Problem 1. We consider the problem finding the longest path in a graph. Note that we only consider simple paths where each vertex can appear at most once in the path. Also, in the following settings there can be both positive and negative arc weights, and even negative cycles in our graph.

a) Suppose we have an acyclic graph. Describe an efficient algorithm to find the longest path between every pair of vertices.

b) Now suppose our graph has cycles. Describe a slow but correct algorithm to solve the all-pairs longest path problem. Analyze the running time of your solution (don't worry if its exponential).

Suppose you had a program L which given any input graph G, returns the longest path distance matrix D where $D[i, j]$ is the length of the longest path between i and j in G.

c) Given a graph G and three vertices i, j, k in G describe how to use two calls to your program L to determine whether or not vertex j is on *every* longest path from i to k. (note that you cannot modify L, but you can call L with any graph H you like and it will return a longest path distance matrix D for H). Your solution should run in $O(n^2)$ time excluding the time taken by the call to L.

d) Describe how to use the program L to efficiently find the *shortest path* length between a given pair of vertices s and t in a graph G.

(25) Problem 2. We consider approximation algorithms for the 0-1 KNAPSACK problem: Given n items, each with a weight w_i and a value v_i, and a weight bound W. We assume that we first remove any item with weight greater than W (so all $w_i \leq W$.)

a Suppose we sort them into decreasing order by value to weight ratio, r_i, and add items until the the next one won't fit. Give a family of examples where the value of this solution, v is much less than the optimal value v^*.

b Now consider a slight variation: We run the the algorithm of part a) and use that solution or the solution that uses the single item of greatest value (call this v_{max}, whichever is better. Show that the value of this solution is at least half the optimal value.

(25) Problem 3. GRAPH-3-COLORING is a *yes/no* question, but we can phrase it as an optimization problem as follows.

Suppose we are given a graph $G = (V, E)$, and we want to color each node with one of three colors, even if we aren't necessarily able to give different colors to every pair of adjacent nodes. Rather, we say that an edge (u, v) is *satisfied* if the colors assigned to u and v are different.

Consider a 3-coloring that maximizes the number of satisfied edges, and let c^* denote this number. Give a polynomial-time algorithm that produces a 3-coloring that satisfies at least $2/3c^*$ edges. If you want, your algorithm can be randomized; in this case, the *expected* number of edges it satisfies should be at least $2/3c^*$.

(25) Problem 4. (a) For the Pattern BABBAB give the finite state machine which allows you to search for this pattern.

(b) The finite state machine you construct will find overlappin instances of the pattern (e.g. if the text starts with BABBABBAB it will find a match starting at both positions 1 and 3). Suppose we only want non-overlapping matches (so after finding a match in positions 1-6 above, we want our next match to start at position 7 or later). Describe how to modify your FSM so that it only reaches the last state when it finds a non-overlapping match.

(25) Problem 5. Given a pattern P of length n and a text T of length m we showed that we could find the best *approximate* match for P in T in O(mn) time using dynamic programming. Our scheme assumed that when matching P to a set of postions in T there was a cost of 1 for: a mismatch; matching a symbol in P to nothing; matching a symbol of T to nothing.

As usual, we also assume that $m >> n$.

a) How much space is required to find some point in the text which has a mismatch of at most k or to determine that no such place exists?

b) How much space is required if we want not only the first position in the text with a mismatch of at most k, but also how that best match occurs (i.e. which characters in the pattern are matched to which characters in the text, and which P/T characters are ignored)?

c) Suppose we modify our cost function so that a mismatch costs 4, matching a character in P to nothing costs 1, and matching a character in the text to nothing in the pattern costs 2. Describe how to modify the dynamic programming algoorithm given in class to find the cheapest match between part of the text and the pattern. Briefly justify your algorithm and give its running time (Hint: if you think carefully you can simplify things a bit).