1. **(2 points each question) True or false?** Circle the correct answer. No explanation required. No points will be subtracted for incorrect answers, so guess all you want.

   F In an undirected graph, the shortest path between two nodes always lies on some minimum spanning tree.

   T If all edges of the graph have different weights then the highest weight spanning tree is unique.

   T In Huffman coding, the item with the second-lowest probability is always at the leaf that is furthest from the root.

   F In Huffman coding, the item with the highest probability is always at a leaf that is the child of the root.

   F If the frequencies of the letters a, b, c are equal, then $a = 01$, $b = 10$, $c = 11$ is a valid Huffman code.

   F Adding an edge to a directed graph can decrease the number of strongly connected components by at most 1.

   T Adding an edge to a directed graph cannot increase the number of strongly connected components.

   T If a Horn formula has no clause with a single literal, then it is necessarily satisfiable.

   T The product-cost of a path from s to t is the product of the weights on the edges of the path. There is an efficient algorithm to compute the minimum product-cost path from s to t.

   F The term dynamic programming refers to the early days when programming languages were still very primitive, and this was the most sophisticated way of programming a computer.

2. *A set of cities V is connected by a network of roads $G(V, E)$. The length of road $e \in E$ is l_e. There is a proposal to add one new road to this network, and there is a list E' of pairs of cities between which the new road can be built. Each such potential road e' has an associated length. As a lobbyist for for city s, you wish to determine the road $e' \in E$ that would result in the maximum decrease in the driving distance between s and a particular city t. Give an efficient algorithm for solving this problem, and analyze its running time as a function of $|V|, |E|$ and $|E'|$.*

   We use Dijkstra from s to any node x and again Dijkstra from t to any node y. This will give us the shortest distances from s ($d_s(x)$) and t ($d_t(y)$). Then we iterate over all $e' = (x, y) \in E'$ and find the edge that minimizes $d_s(x) + l_{e'} + d_t(y)$. The total running time is Dijkstra plus $O(|E'|)$, which is $O(|E| \lg V + |E'|)$

   Any solution that had a slower running time was worth 10 points (e.g. if you run Dijkstra for every edge in E', the running time is $O(|E||E'| \lg V)$).

3. *The set cover problem is easy if the sets have the following special structure: $U = \{1, 2, \ldots, n\}$, and you are given a collection of m subsets $S \subseteq U$, where each set S in the collection is contiguous. i.e. $S = \{i : j \leq i \leq k\}$ for some $1 \leq j, k \leq n$. Give an efficient algorithm for* contiguous set cover*. Prove the correctness of your algorithm and*

analyze its running time. (Extra credit if your algorithm runs in $O(m+n)$ time)

The algorithm is to repeatedly cover the lowest numbered uncovered element with the set that covers the most elements.

Correctness follows inductively. We consider a point where the leftmost uncovered element is i. Let T_i be the set of sets that cover i. An optimal cover must include one of these sets and must cover all the remaining elements that are in none of the sets in T. The algorithm produces has size 1 plus the size of the optimal cover of the remaining elements not in i. The base case is the fact that covering no elements has cost 0. (Note this induction is using the induction hypothesis on the larger numbered elements.)

Without care, the above algorithm runs in $O(nm)$ time.

The algorithm can be implemented in linear time by bucketing the subsets in order of left endpoint. After choosing a set that covers an element i, scan the buckets up to the next uncovered element i', choosing the set in these buckets that covers the largest numbered element. (If no set covers i', there is no set cover.)

We accepted dynamic programming solutions of several kinds as well. Some ran in $O(n^2m)$ time.

4. *You are given n integers in the range $[1, m]$ and a target T. Find a subset of the n integers whose sum is as close to T as possible without exceeding it. Give an efficient dynamic programming algorithm for this problem. Remember to specify the order in which you fill in the entries of your table.*

This problem reduces to knapsack without repetition. Assume the integers you are given are $a_1, ..., a_n$. The capacity of the knapsack is T and there are n items- item i has value a_i and weight a_i. The running time is $O(nT)$, and the dynamic programming algorithm is the same as the algorithm for knapsack without repetition (page 168 in the textbook).

Now suppose that you wish to partition your integers into two subsets such that the sum of the two subsets are as nearly equal as possible. i.e. $|S_1 - S_2|$ is as small as possible. How would you modify the above algorithm to achieve this?

Let $S = \sum_i a_i$. Change the target T to $\frac{S}{2}$. Let the subset that gives you the optimal solution be S_1, and all other integers will form S_2. If the optimal value with capacity $\frac{S}{2}$ is k, then $|S_1 - S_2| = S - 2k$.