**Problem 1:** You are running an art museum. There is a long hallway with $k$ paintings on the wall. The locations of the paintings are $l_1, ..., l_k$. These locations are real numbers, but not necessarily integers. You can place guards at locations in the hallway, and a guard can protect all paintings within $1$ unit of distance from his location. The guards can be placed at any location, not just a location where there is a painting. Design a greedy algorithm to determine the minimum number of guards needed to protect all paintings.

**Problem 2:** Suppose you are given a set of jobs $J_1, ..., J_m$ to perform. Each job $J_i$ will pay you some amount of money $p_i$ (not all jobs give the same payment). Each job $J_i$ additionally has a deadline $d_i$, and if the job is not complete before its deadline, you will not receive any of the payment for that particular job. Assume that the timeline starts at time 0. Each job takes one unit of time to perform.

For example, suppose that there are three jobs $J_1, J_2, J_3$, with payments of $1, 2, 5$ and deadlines of $3, 1, 1$. Then the best schedule is to do job $J_3$ from time 0 to time 1, and job $J_1$ from time 1 to time 2 (or time 2 to time 3). Because the timeline starts at time 0 and each job takes 1 unit of time to complete, it is not possible to do both jobs $J_2$ and $J_3$ before their deadline at time 1. Then after time 1, $J_1$ is the only job left, so we select it, and get a total payment of $6$.

Design a greedy algorithm to determine the schedule of jobs that will maximize the amount of payment that you receive. Your algorithm should output the actual schedule, not just the total payment (i.e., for each job, it should tell me when that job will start, or if it won't be scheduled at all, then it should say that).

**Problem 3:** Suppose that you are traveling on a river by canoe. There are $n$ canoe rental shops along the river. At each shop, you can pick up a canoe or drop off a canoe. The cost for renting a canoe from shop $i$ to shop $j$ is given by $c_{i,j}$. There is no relationship between the different $c$ values. You need to go from the first rental shop to the last rental shop. Design a dynamic programming algorithm to determine the minimum cost of such a trip. Hint: Let $C[i]$ represent the cheapest way to get to rental shop $i$. How would your solution change if you also have to keep track of which shops you should rent from and drop off to in order to achieve the minimum cost trip?

Hint: Define $C[i]$ to be the cheapest way to get to rental shop $i$. If you stop at shop $i$, which rental shops could you have potentially stopped at immediately before you got to shop $i$?

**Problem 4:** You have a set of coins. Each coin $i$ has value $v_i$. Your goal is to find a set of coins with total value exactly equal to $V$. You can use as many copies of each coin as you wish. Design a dynamic programming algorithm to solve this.

**Problem 5:** You are given a series of boxes. Each box $i$ has a rectangular base with width $W_i$ and length $L_i$, as well as a height $H_i$. You are stacking the boxes, subject to the following: In order to stack a box $i$ on top of a second box $j$, the width of box $i$ must be strictly less than the width of box $j$ *and* the length of box $i$ must be strictly less than the length of box $j$ (assume that you cannot rotate the boxes to turn the width into the length, or the length into the height, etc.). Your job is to make a stack of boxes with total height as large as possible. You can only use one copy of each box.

Part a: Draw and explain the DAG corresponding to this problem. Hint: What is a reasonable way to sort the boxes so that all edges go from earlier nodes to later nodes?

Part b: Describe an efficient algorithm to determine the height of the tallest possible stack.

**Extra Credit**: You are given a sequence of $n$ numbers $x_1, ..., x_n$. Create a dynamic programming algorithm to find the maximum alternating sum of a contiguous subsequence within this sequence. An alternating sum of a sequence $x_j, ..., x_k$ is defined as $x_j - x_{j+1} + x_{j+2} - x_{j+3} + xj + 4 - ... \pm x_k$. $j$ and $k$ reflect indices within the original sequence. Note that you do not have flexibility in determining which terms are added and which are subtracted: the first, third, etc. terms in the subsequence are always added, and the others are subtracted. Your algorithm does not have to output the best subsequence: just its sum.