CS 330 - Dynamic Programming Problems

Procedure

We want to find the choices that maximize or minimize some quantity. Here's how we could approach this with dynamic programming

1. First write a recursive solution for the max (or min) quantity.
   (a) For each initial choice you can make, compute the cost of making that choice plus the (recursively calculated) best way of making all other choices.
   (b) Return the maximum (or minimum) of all the possible choices.
2. Then, it is usually more efficient to write a non-recursive version:
   (a) Compute "base cases" and write answers into an array.
   (b) Have a loop compute higher values in terms of already computed values in the array
   (c) The code should look very similar to the recursive version.
3. If needed, modify code to keep track of the choices made as well as the quantities they yield.

Problems

We will start by focusing on writing a top-down recursive solution first.

1. Suppose you have a knapsack that can hold a weight of up to W. You are robbing a store, and in front of you are items $1 \ldots n$ with values v_i and weights w_i. You want to pick the items to put in your knapsack so that you maximize the total value of what's in the knapsack.
   (a) A greedy approach is to take the most valuable item that will fit in your knapsack first, then the next most valuable item, etc. Find an example where this doesn't yield an optimal solution.
   (b) Another form of the greedy approach is to first take the most valuable item by weight. That is, you compute the price per pound of each object, then take the item with highest price-per-pound, etc. Suppose $W = 50$ and you have three items with $v_1 = 60, v_2 = 100, v_3 = 120$ and $w_1 = 10, w_2 = 20, w_3 = 30$. Does this greedy approach work for this example?
   (c) Let's consider whether the last item, n,should go in the knapsack or not. Clearly it won't go in if its weight is too large. If it does go into the knapsack, what recursive subproblem will we need to solve?
   (d) If item n does not go into the knapsack, what recursive subproblem do we need to solve?
   (e) How would we test whether n goes into the knapsack or not?
   (f) Complete a *recursive* function `knapsackRecursive(int W, int[] wt, int[] val,int n)` in the code provided. This function should compute the maximal possible total value that can fit into a knapsack of capacity W when the items have values $val[i]$ and weights $wt[i]$

Now we will develop a bottom-up solution.

2. Write a bottom-up solution for the knapsack problem. We will create an array such that `K[i][w]` will contain the optimal solution for when we have i items and we can carry weight w. We will use nested loops to fill in the values in this array. Instead of doing a recursive call as before, we will simply use a value that we previously assigned in the array. Complete the new version and test it out.

We now want to develop a full solution.

4. Modify your previous solution to keep track of the optimal choices being used. You will want to create another array to store that information.