Homework Assignment #6 Dynamic Programming

1. Modify MEMOIZED-CUT-ROD (as given in slides and textbook CLR) to return not only the maximum revenue, but also the actual cutting that achieves the maximum revenue (as an array of integers specifying the length of rods that should be cut out). Show the content of the tables (arrays) at the end of the algorithm for the following input:

\[ p[0..10] = 0, 1, 5, 8, 9, 10, 17, 17, 20, 24, 30 \]

and \( n = 16 \).
2. (Knapsack with repetition) Read Section 6.4 of textbook DPV about the Knapsack problem with repetition and its solution.

(a) What type of implementation is used in the algorithm given in page 172 (or page 167 of the print book): top-down with memoization or bottom-up? Hint: the notation \( K(w) \) used in the algorithm should NOT be interpreted as a function call, instead it should be treated as \( K[w] \), i.e., accessing table element using index \( w \).

(b) Rewrite the algorithm using recursive function without memoization.

(c) Comment on how this version of Knapsack problem is similar to rod cutting problem. For example, you can list that the given rod length \( n \) corresponds to the total weight the burglar can carry, \( W \). Please identify all similar points between the two, and how the algorithms are also similar.
3. Dynamic Programming algorithm for LCS.

- Follow the code below to determine an LCS (Longest Common Subsequence) of <1, 0, 0, 1, 0, 1, 0, 1, 0> and <0, 1, 0, 1, 1, 0, 1, 1, 0>. Please show the 2-D table c and b (as copied below from the textbook Introduction to Algorithms, by T. Cormen, C.E. Leiserson, R. L. Rivest) as being used in the pseudocode LCS-Length. Note the 2-D table b is set so that b[i, j] points to the table entry corresponding to the optimal subproblem solution chosen when computing c[i, j].

```
LCS-LENGTH(X, Y)
1   m = X.length
2   n = Y.length
3   let b[1..m, 1..n] and c[0..m, 0..n] be new tables
4   for i = 1 to m
5       c[i, 0] = 0
6   for j = 0 to n
7       c[0, j] = 0
8   for i = 1 to m
9       for j = 1 to n
10          if x_i == y_j
11              c[i, j] = c[i-1, j-1] + 1
12              b[i, j] = "\n"
13          elseif c[i-1, j] >= c[i, j-1]
14              c[i, j] = c[i-1, j]
15              b[i, j] = "|"
16          else c[i, j] = c[i, j-1]
17              b[i, j] = "<-
"18   return c and b
```

- (optional) Give pseudocode to reconstruct an LCS from the completed c table and the original sequences X = <x_1, x_2, ..., x_m>, and Y = <y_1, y_2, ..., y_n> in O(m + n) time, without using the b table.
4. Read page 169 (or page 165 for the print book) of textbook DPV, which describes four common methods to define subproblems. This is crucial for coming up a dynamic programming solution to a problem. Answer the following questions:

(a) For the above Memoized-Cut-Rod problem, which of the four common approaches (if any) is used to define the subproblem?

(b) In the LCS algorithm, which of the four methods is used to define subproblems?
5. Choose one from the following two problems (which we have studied earlier in the class), and design a dynamic programming algorithm for the problem. Please pay attention to the following requirement for your pseudocode:

- Define the original problem as a function that takes parameters, and return some results.
- Define the subproblems to the original problem (referring to Problem 4 above for the some general ways to do this).
- Write recursive formula that relates a problem’s solution to solutions of smaller subproblems.
- Finally write out pseudocode for the algorithm (using top-down memoization or bottom-up).

(a) (Maximum Subarray Problem) Input: an integer list \( A[left...right] \) where each element is a positive or negative integer,

Functionality: find out the sublist of \( A \) that has the largest sum among all sublists of \( A \).

Output: return the sum and the indices \( i \) and \( j \), such that sublist \( A[i...j] \) has the largest sum.

Example: \( A[0...8] = \{1, -30, 20, 20, -3, -2, 18, -30, 5\} \)

Solution: \( i = 2, j = 6 \), i.e., the maximal sublist is \( A[2...6] = \{20, 20, -3, -2, 18\} \). No other subarray has a larger sum.

(b) Suppose a list \( P[1...n] \) gives the daily stock price of a certain company for a duration of \( n \) days, we want to find an optimal day \( d_1 \) to buy the stock and then a later day \( d_2 \) to sell the stock, so that the profit is maximized (i.e., \( P[d_2] - P[d_1] \) is maximized. For example, if \( P[1...6] = 20, 10, 30, 50, 5, 14 \)

Then we should buy in at day 2, and sell at day 4 to earn profit of 50 – 10 per share. (Simply buying at the lowest point and selling at the highest point will not necessarily work, as one have to buy before sell.)