Practice Problem Set: Dynamic Programming

Problem 1. The counting sums problem is to count the number of ways a number can be written as the sum of two or more positive integers. For example, we can write 6 as the sum of two or more positive integers in the following ways

$$\begin{aligned}
5+1&=6\\
4+2&=6\\
4+1+1&=6\\
3+3&=6\\
3+2+1&=6\\
3+1+1+1&=6\\
2+2+2&=6\\
2+1+1+2&=6\\
2+1+1+1+1&=6\\
1+1+1+1+1+1&=6
\end{aligned}$$

Using a dynamic programming approach, write an algorithm to determine the number of ways 100 can be written as the sum of two or more positive integers.

Problem 2. Consider the triangle below constructed such that the n^{th} row contains n numbers. Suppose, starting at the top of the triangle, we have to move to adjacent numbers in the row below such that the sum of the selected numbers is maximum. For example, in the following triangle, the maximum total from top to bottom is 23 given by the path of numbers shown in red, i.e. $3+7+4+9=23$

3
7 4
2 4 6
8 5 9 3

Note that there are a total of 2^{n-1} paths (where n is the number of rows in the triangle) from the top to the bottom of the triangle. Therefore, it is not feasible to find the maximum sum that can obtained by the allowed movement when n is large. Devise a dynamic programming solution to find the maximum possible sum and analyze the running time of your solution.

Problem 3. The zombie apocalypse has occurred, and you and a group of friends have managed to survive for the moment. You are currently bunkered in a government safe-house that has an electric fence around it, with sensors attached which indicate when zombies are approaching. At minute i, your sensors indicate that x_i zombies are approaching. If they cross the fence, they will attack the safe-house. But as they are right on the fence, you have the choice of 'zapping' the zombies by discharging the capacitors attached to the fence. Unfortunately, the capacitors need time to recharge i.e. if you let the fence charge for j minutes, you will have enough charge to zap d_j zombies.

Example: Suppose $(x_1,x_2,x_3,x_4) = (1, 10, 10, 1)$ and $(d_1,d_2,d_3,d_4) = (1, 2, 4, 8)$. The best solution would be to zap at times 3 and 4, since this would zap a total of 5 zombies.

1. Construct an instance of the problem for which the following "greedy" algorithm does not produce an optimal solution. Intuitively, the greedy algorithm figures out how many minutes (j) are needed to zap the zombies in the last time slot. It zaps during that last time slot, and then accounts for the j minutes required to recharge for that last time slot, and recursively considers the best solution for the smaller problem of size $n - j$.

Algorithm 1 :

function BadZap($(x_1, \cdots, x_n)$,$(d_1, \cdots, d_n)$)
  Compute the smallest j such that $d_j \geq x_n$. Set $j = n$ if no such j exists
  Zap fence at time n.
  if $n > j$ **then**
    return $\min(d_j, x_n)$ + BadZap($(x_1, ..., x_{n-j}), (d_1, ..., d_{n-j})$)
  else
    return $\min(d_j, x_n)$

2. Given two arrays $(x_1, \cdots, x_n)$ and $(d_1, \cdots, d_n)$, devise an algorithm which zaps the most zombies and produces the optimal times to zap the zombies. Analyze the runtime of your algorithm.

Problem 4. Aiman uses her cellphone for m_i minutes in a month i. She can use the *rob-you-as-you-go* plan and pay r_i for each minute (hence her cost for a month i is $m_i r_i$). She can also sign up for the *rob-you-in-bulk* plan in which she pays a flat monthly rate of c, but she must also agree to a 6-month contract by which she must continue the plan for 6 months. To simplify, assume the 6-month contract ends after 6 months and does not extend on a month-to-month basis as many cellphone plans do. Given the minutes $(m_1, \cdots, m_n)$ and cost parameters $(r_1, ...r_n)$ for n months and the fixed cost c, Aiman wishes to find the cheapest way to provide herself with the cellphone service.

1. Describe a strategy for solving the problem by formulating a relation of the solution to the problem of size n to the solutions of problems of smaller sizes.

2. Devise a polynomial-time algorithm of the form **CheapestService**$((m_1, \cdots, m_n),(r_1, \cdots, r_n),c)$ which computes the minimum cost for Aiman (the actual decisions of which plan to use for each month are not needed). Analyze the running time of your algorithm.

Problem 5. Consider a directed graph $G = (V, E), V = \{v_1, v_2, \cdots, v_n\}$. The following two properties make G an *ordered graph*:

(i) Each directed edge (v_i, v_j) is from a node with lower index to a node with higher index i.e. $\forall (v_i, v_j) \in E, i < j$

(ii) Each node except v_n has out degree atleast 1, i.e. $\forall\ v_i, i = 1, 2, \cdots, n-1$, there is at least one edge of the form (v_i, v_j)

The length of a path is the number of edges it contains. Give an efficient algorithm using dynamic programming which finds the length of the longest path P in an ordered graph G, such that P begins at v_1 and ends at v_n. Analyze the correctness and running time of your algorithm.

Problem 6. Recall the coin change problem: Coins of a set of denominations (values) are available to a cashier who needs to provide change for a given amount V , such that the number of coins returned in exchange for V is minimum. We assume an infinite supply of each denomination. Also recall that the greedy approach does not produce the optimal result for *any* given set of denominations. Devise a dynamic programming algorithm which, given a set of denominations $D = \{d_1, \cdots, d_k\}$ and value V, produces the optimal result for the coin change problem. Briefly argue about the optimal substructure property of the problem.

Problem 7. Recall the definitions of a substring of a string. The longest common substring problem is as follows. Given two strings, find the longest common substring of the two strings. For example, if A="abcdefgyu" and B="bcdtyu", then the longest common substring of A and B is "bcd" of length 3. Devise a dynamic programming solution to find the longest common substring of two given strings. [*Hint:* Find the length of the longest common suffix for all substrings of both strings and store these lengths in a table.]

Problem 8. Recall the definitions of a subsequence of a sequence. The longest common subsequence problem is as follows. Given two sequences, find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous. For example, "abc", "abg", "bdf", "aeg", "acefg", etc. are subsequences of "abcdefg". Briefly argue about the optimal substructure and overlapping subproblems of this problem.

1. Devise a dynamic programming solution to find the length of longest common subsequence of two given sequences S and T. [*Hint:* Look at the length of the longest common subsequence of a prefix of S of length i and a prefix of T of length j, running over all pairs of prefixes, and consider the two cases: $S[i] = T[j]$ and $S[i] \neq T[j]$.]

2. Describe how you would find the actual longest common subsequence using the algorithm you devised in the previous part of this question.