AAD Problem Set - 2

Dynamic Programming

September 2025

Introduction

This problem set is ungraded and meant purely for practice. The problems have been shuffled and do not include topic tags, so that you can approach them without a bias. If you encounter any doubts, feel free to reach out during office hours. In case of typos or unclear statements, please let us know.
In the exam, you might be asked to "give an algorithm" to solve a problem. Your write-up should take the form of a short essay. Start by defining the problem you are solving and stating what your results are. Then provide:

- A description of the algorithm in English and, if helpful, pseudo-code.
- A proof (or proof sketch) for the correctness of the algorithm.
- An analysis of the running time.

We will give full credit only for correct solutions that are described clearly.

How to Write a Proper DP Solution

When presenting a dynamic programming solution, there are a few points that should always be stated **explicitly outside of the pseudocode**. This helps avoid ambiguity and makes the logic behind the solution transparent.

1. **State:** Clearly define what $dp[x]$ represents. Without this, the recurrence is meaningless.
2. **Base Case(s):** Write down the initial conditions for the DP. These are the anchors that ensure your recurrence has something to build upon.
3. **Transition:** Provide the recurrence relation, and explain why it avoids circular dependencies (remember: DP essentially relies on solving problems over a DAG).
4. **Target:** Specify what you are actually trying to compute. In some problems, the final answer is $dp[n]$, but in others it could be something like $dp[1] + dp[2] + \cdots + dp[n]$.

Finally, it goes without saying that every DP solution should include:

- A **proof of correctness** (why the states and transitions are sufficient to cover **ALL** valid cases).
- A clear statement of the **time and space complexity**.

Knapsack DP

Problem 1

Design an algorithm to count the number of ways to construct sum n by throwing a dice one or more times. Each throw produces an outcome between 1 and 6.

Problem 2

Consider a money system consisting of n coins. Each coin has a positive integer value. Design an algorithm `COIN-COMBINATIONS` to calculate the number of distinct ways you can produce a money sum x using the available coins. For example, if the coins are $\{2, 3, 5\}$ and the desired sum is 9, there are 8 ways:

$$2+2+5$$
$$2+5+2$$
$$5+2+2$$
$$3+3+3$$
$$2+2+2+3$$
$$2+2+3+2$$
$$2+3+2+2$$
$$3+2+2+2$$

Problem 3

Consider a money system consisting of n coins. Each coin has a positive integer value. Design an algorithm `ORDERED-COMBINATIONS` to calculate the number of distinct *ordered* ways you can produce a money sum x using the available coins.

Problem 4

For example, if the coins are $\{2, 3, 5\}$ and the desired sum is 9, there are 3 ways:

$$2+2+5$$
$$3+3+3$$
$$2+2+2+3$$

Problem 5

Consider a money system consisting of n coins. Each coin has a positive integer value. Your task is to design an algorithm `MINIMIZE-COMBINATIONS` which produces a sum of money x using the available coins in such a way that the number of coins is minimal. For example, if the coins are $\{1, 5, 7\}$ and the desired sum is 11, an optimal solution is $5 + 5 + 1$ which requires 3 coins.

Problem 6

There are N items, numbered $1, 2, \ldots, N$. For each i ($1 \leq i \leq N$), item i has a weight of w_i and a value of v_i. Alice has decided to choose some of the N items and carry them home in a knapsack. The capacity of the knapsack is W, which means that the sum of the weights of items taken must be at most W.

Design an algorithm that finds the maximum possible sum of the values of items that Alice takes home.

Problem 7

You have n coins with certain values. Your task is to design `MONEY-SUMS` which finds all money sums you can create using these coins.

Miscellaneous DP

Problem 8

Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is {5, 10, 3, 12, 5, 50, 6} using MCM(Matrix Chain Multiplication) DP.

Problem 9

Show that a full parenthesization of an n-element expression has exactly $n - 1$ pairs of parentheses.

Problem 10

Design an algorithm called as `PRINT-OPTIMAL-PARENS` for constructing an optimal solution of matrix-chain product.

Problem 11

Describe the subproblem graph for matrix-chain multiplication with an input chain of length n. How many vertices does it have? How many edges does it have, and which edges are they?

Problem 12

Give an $O(n)$-time dynamic-programming algorithm to compute the n^{th} Fibonacci number. Draw the subproblem graph. How many vertices and edges are in the graph?

Problem 13

Consider a variant of the matrix-chain multiplication problem in which the goal is to parenthesize the sequence of matrices so as to maximize, rather than minimize the number of scalar multiplications. Does this problem exhibit optimal substructure?

Problem 14

Design an algorithm called as `PRINT-LIS` which reconstructs the LIS of a given sequence.

Problem 15

Biological applications often need to compare the DNA of two (or more) different organisms. A strand of DNA consists of a string of molecules called bases, where the possible bases are adenine, guanine, cytosine, and thymine. Representing each of these bases by its initial letter, we can express a strand of DNA as a string over the finite set {A, G, C, T}. One reason to compare two strands of DNA is to determine how "similar" the two strands are, as some measure of how closely related the two organisms are. For example, the DNA of one organism may be S_1= `ACCGGTCGAGTGCGCGGAAGCCGGCCGAA`, and the DNA of another organism may be S_2= `GTCGTTCGGAATGCCGTTGCTCTGTAAA`. Hence the similarity or the longest-common-subsequence of two sequences becomes an interesting study. Design an algorithm `LCS-Length` which computes the longest common subsequence of two sequences. Formally state it's proof of correctness and time complexity for a full credit.

Problem 16

Design algorithm `PRINT-LCS` which returns the any one of the computed LCS (Notice that there might be multiple, so we can accept any one of those such strings). For example let S_1 = `axyb` and S_2= `abyxb` the `PRINT-LCS(`S_1`, `S_2`)` gives us `axb`. Either `axb` or `ayb` will be accepted too.

Problem 17

Give an $O(nlgn)$-time algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers.

Problem 18

Suppose that we are given a directed acyclic graph $G = (V, E)$ with real valued edge weights and two distinguished vertices s and t. Describe a dynamic programming approach for

finding a longest weighted simple path from s to t. What does the subproblem graph look like? What is the efficiency of your algorithm? (Hint: Class example)

Problem 19

A *palindrome* is a nonempty string over some alphabet that reads the same forward and backward. Examples of palindromes are all strings of length 1, `civic`, `racecar`, and `aibohphobia` (fear of palindromes).

Give an efficient algorithm to find the longest palindrome that is a subsequence of a given input string. For example, given the input `character`, your algorithm should return `carac`. What is the running time of your algorithm?

Problem 20

In the *euclidean traveling-salesman problem*, we are given a set of n points in the plane, and we wish to find the shortest closed tour that connects all n points. Figure 1(a) shows the solution to a 7-point problem. The general problem is NP-hard, and its solution is therefore believed to require more than polynomial time.

J. L. Bentley has suggested that we simplify the problem by restricting our attention to *bitonic tours*, that is, tours that start at the leftmost point, go strictly rightward to the rightmost point, and then go strictly leftward back to the starting point. Figure 1(b) shows the shortest bitonic tour of the same 7 points. In this case, a polynomial-time algorithm is possible.

Describe an $O(n^2)$-time algorithm for determining an optimal bitonic tour. You may assume that no two points have the same x-coordinate and that all operations on real numbers take unit time. (*Hint:* Scan left to right, maintaining optimal possibilities for the two parts of the tour.)

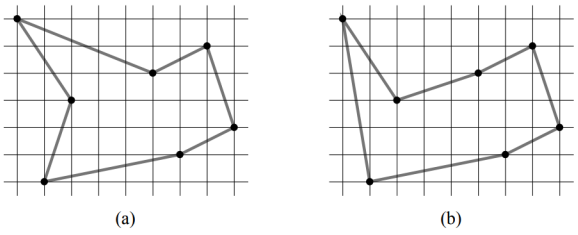

Figure 1: Seven points in the plane, shown on a unit grid. (a) The shortest closed tour, with length approximately 24.89 and tour is not bitonic. (b) The shortest bitonic tour for the same set of points. Its length is approximately 25.58

Problem 21

Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer. In other words, is it possible to make change v using coins

of each denomination **atmost** once? Design an algorithm and analyze it's time complexity and proof of correctness.

Suppose that the available coins are in the denominations that are powers of c, i.e., the denominations are $c^0, c^1, ..., c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

Problem 22

Given integers n and k, along with $p_1, \ldots, p_n \in [0, 1]$, you want to determine the probability of obtaining exactly k heads when n biased coins are tossed independently at random, where p_i is the probability that the ith coin comes up heads. Give an $O(n^2)$ algorithm for this task.[1]

Problem 23

Is it possible to make change for v using atmost k coins, of denominations given by set $\{x_1$, x_2, ... $x_n\}$ assuming infinite supply of each coin.

Problem 24

Suppose two teams, A and B, are playing a match to see who is the first to win n games (for some particular n). We can suppose that A and B are equally competent, so each has a 50% chance of winning any particular game. Suppose they have already played $i + j$ games, of which A has won i and B has won j. Give an efficient algorithm to compute the probability that A will go on to win the match.

For example, if $i = n - 1$ and $j = n - 3$ then the probability that A will win the match is 7/8, since it must win any of the next three games.

Problem 25

Often two DNA sequences are significantly different, but contain regions that are very similar and are highly conserved. Design an algorithm that takes as input two strings $x[1 \ldots n]$ and $y[1 \ldots m]$ and a scoring matrix δ , and outputs substrings x' and y' of x and y, respectively, that have the highest-scoring alignment over all pairs of such substrings. Your algorithm should take time $O(mn)$.

Problem 26

Not a DP problem: Prove that gcd of F_n and F_{n-1} is 1 where F_n is n^{th} term in fibonacci sequence, given by $F_0 = 1$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$

[1] Assume you can multiply and add two numbers in $[0, 1]$ in $O(1)$ time.

More problems adding soon