Dynamic Programming (DP) - Lecture Notes

Dynamic programming is an optimization technique used to solve complex problems by breaking them down into simpler sub-problems. It stores the result of these sub-problems to avoid computing them multiple times.

Memoization vs. Tabulation

These are the two main approaches to implementing DP algorithms.

Memoization (Top-Down)	Tabulation (Bottom-up)
Uses recursion.	Uses iteration (Loops).

Feature	Memoization	Tabulation
Direction	Top-Down	Bottom-up
Implementation	Recursive	Iterative
Storage	Hash Map or Array	Array or Matrix

The Knapsack Problem

Given a set of items, each with a weight and a value, determine which items to include in a collection so that the total weight is less than or equal to a given limit (W) and the total value is as large as possible.

> Constraint: You cannot break items (0 or 1).

DP State: dp[i][w] = Maximum value achievable using the first i items with a capacity of w.

Recurrence Relation:

$$dp[i][w] = \max(val[i] + dp[i-1][w - wt[i]],\ dp[i-1][w])$$

C Code

```
#include <stdio.h>
 int max( int a, int b) { return (a>b) ? a : b; }

 // w: capacity, wt[]: weights, val[]: values, n: number of
// items.
 int knapsack (int W, int wt[], int val[], int n) {
        int dp[n+1][w+1];
        for (int i=0; i <= n; i++) {
            for (int w=0; w <= W; w++) {
                if (i == 0 || w==0)
                    dp[i][w] = 0; //Base case
```

```
            else if (wt[i-1] <= w)
                dp[i][w] = max(val[i-1] + dp[i-1][w-wt[i-1]],
                                                      dp[i-1][w]);
            else
                dp[i][w] = dp[i-1][w];

        }
    }
    return dp[n][w];
}
```

The Coin Change Problem

There are two variations:

1. Min Coins: Minimum number of coins to make a value.
2. Number of ways: Total ways to form a value.

Variation: Minimum Coins

Problem: Given an amount V and set of coin denominations, find the minimum number of coins needed.

DP State: dp[v] minimum coins needed to make value v.

Recurrence Relation:

$$dp[v] = \min(dp[v],\ dp[v - coin_j] + 1)$$

Segmented Least Squares

This is an optimization problem often encountered in statistical analysis and curve fitting.

Problem: You have n points in a plane $(x_1, y_1), \ldots\ldots,$ (x_n, y_n). You want to fit piecewise linear functions (segments) to these points.

Trade off

Accuracy: We want to minimize the error. (SSE) of the lines.
Parsimony: We want to minimize the number of lines used.

$$\boxed{\text{Cost Function : } E + c \times L}$$

E : sum of squared errors.
c : Penalty cost for each segment.
L : number of segments.

$$\boxed{OPT(j) = \min_{1 \leq i \leq j} \left(SSE(i,j) + C + OPT(i-1)\right)}$$

$SSE(i,j)$: The least squares error for a single line fitted through points i to j.
C : The penalty cost for creating a new segment.
$OPT(i-1)$: The optimal solution found for the previous points.