# Algorithms and Data Structures 2
# Exam Notes
# Week 3: Dynamic Programming II

Mads Richardt

## 1 General Methodology and Theory

### 1.1 Dynamic Programming Principles

- **Optimal substructure:** Large problems can be solved by combining optimal solutions to smaller subproblems.
- **Overlapping subproblems:** Same subproblems appear repeatedly; store results in a table.
- **Bottom-up vs Top-down:**
  - Bottom-up: Fill a table iteratively.
  - Top-down: Recursion with memoization.

### 1.2 Knapsack and Subset Sum

**Problem:** Given items with weight $w_i$ and value $v_i$, find maximum total value with total weight $\leq W$.

**Recurrence:**

$$OPT(i,w) = \begin{cases} OPT(i-1,w) & \text{if } w < w_i \\ \max\{OPT(i-1,w),\, v_i + OPT(i-1,w-w_i)\} & \text{otherwise} \end{cases}$$

**Time:** $O(nW)$ (pseudo-polynomial). **Space:** $O(nW)$, reducible to $O(W)$.

**Pseudocode:**

```
Knapsack(n,W):
  M[0..n][0..W] = 0
  for i=1..n:
    for w=0..W:
      if w < wi:
        M[i][w] = M[i-1][w]
      else:
        M[i][w] = max(M[i-1][w], vi + M[i-1][w-wi])
  return M[n][W]
```

### 1.3 Sequence Alignment

**Problem:** Given strings $X, Y$, gap penalty $\delta$, mismatch cost $\alpha_{pq}$, compute minimum-cost alignment.

**Recurrence:**

$$OPT(i,j) = \min\left(\alpha_{x_i,y_j} + OPT(i-1,j-1),\, \delta + OPT(i-1,j),\, \delta + OPT(i,j-1)\right)$$

with $OPT(i,0) = i\delta$, $OPT(0,j) = j\delta$.

**Pseudocode:**

```
Alignment(X,Y):
  A[0..m][0..n]
  for i=0..m: A[i][0] = i
  for j=0..n: A[0][j] = j
  for i=1..m:
    for j=1..n:
      A[i][j] = min( [xi,yj] + A[i-1][j-1],
                      + A[i-1][j],
                      + A[i][j-1])
  return A[m][n]
```

### 1.4 Longest Palindromic Subsequence (LPS)

**Idea:** Compute LCS between string $S$ and its reverse $S^R$.

**Alternative recurrence (direct):**

$$LPS(i,j) = \begin{cases} 1 & \text{if } i = j \\ 2 + LPS(i+1, j-1) & \text{if } s_i = s_j \\ \max(LPS(i+1,j),\, LPS(i,j-1)) & \text{otherwise} \end{cases}$$

Time $O(n^2)$, space $O(n^2)$.

## 2 Notes from Slides and Textbook

### 2.1 Knapsack (KT 6.4)

- Items have weights $w_i$ and values $v_i$.
- Dynamic programming table $M[i][w]$ holds best value for first $i$ items and capacity $w$.
- Complexity $O(nW)$.

### 2.2 Sequence Alignment (KT 6.6)

- Align strings using gaps ($\delta$) and mismatches ($\alpha_{pq}$).
- Optimal alignment can be recovered by backtracking through DP table.
- Running time $O(mn)$, space $O(mn)$.

## 3 Solutions to Problem Set

### 3.1 Problem 1: Knapsack

Items: (5,7), (2,6), (3,3), (2,1). Capacity $W = 6$.

Fill DP table ($i$ = items considered, $w$ = capacity):

$$OPT(4,6) = 9 \quad \text{(by choosing items (2,6) and (3,3))}$$

### 3.2 Problem 2: Sequence Alignment

Strings: APPLE, PAPE. Gap penalty $\delta = 2$. Penalty matrix given.

Compute DP table using recurrence. Backtrack to get alignment:

APPLEPAPE-

with minimum cost found in $OPT(5,4)$.

### 3.3 Problem 3: Book Shop

Map prices $h_i$ to weights, pages $s_i$ to values. Use knapsack DP:

$$OPT(i, w) = \max(OPT(i-1, w), s_i + OPT(i-1, w-h_i))$$

**Time:** $O(nx)$, **Space:** $O(nx)$ reducible to $O(x)$ with 1D array.

### 3.4 Problem 4: Longest Palindromic Subsequence

Use LCS($S, S^R$). **Recurrence:** see Section 1.3. **Time:** $O(n^2)$. **Answer:** Algorithm returns both length and subsequence via backtracking.

### 3.5 Problem 5: Defending Zion (Exercise 6.8)

Input: arrival sequence $x_1, \ldots, x_n$, recharge function $f(j)$.

**Recurrence:**

$$OPT(k) = \max_{1 \le j \le k} \big(OPT(k-j) + \min(x_k, f(j))\big)$$

Base case: $OPT(0) = 0$. Time $O(n^2)$, possible optimizations depend on $f$.

## 4 Puzzle of the Week: The Blind Man

**Problem:** 52 cards, 10 face-up. Divide into two piles with equal number of face-up cards.

**Solution:** Take any 10 cards for pile A. Let pile B be the rest. Flip all cards in pile A.

- Suppose pile A initially had $k$ face-up cards. Then pile B has $10-k$.
- After flipping pile A, it has $10-k$ face-up cards.
- Thus both piles have $10-k$ face-up cards.

## 5 Summary

### Knapsack

$$OPT(i, w) = \max(OPT(i-1, w),\ v_i + OPT(i-1, w-w_i))$$

Time $O(nW)$, Space $O(W)$.

### Sequence Alignment

$$OPT(i, j) = \min(\alpha_{x_i, y_j} + OPT(i-1, j-1),\ \delta + OPT(i-1, j),\ \delta + OPT(i, j-1))$$

Time $O(mn)$, Space $O(mn)$.

### Longest Palindromic Subsequence

$$LPS(i, j) = \begin{cases} 1 & i = j \\ 2 + LPS(i+1, j-1) & s_i = s_j \\ \max(LPS(i+1, j), LPS(i, j-1)) & \text{otherwise} \end{cases}$$

### Defending Zion

$$OPT(k) = \max_{1 \le j \le k} \big(OPT(k-j) + \min(x_k, f(j))\big)$$

### Puzzle

Flip chosen 10 cards $\Rightarrow$ equal face-up cards in both piles.