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

Mads Richardt

1 General Methodology and Theory

Dynamic Programming Principles

- **Subproblems & optimal substructure.** Define states so each depends only on smaller states.
- **Recurrence → evaluation order.** Choose bottom-up or memoized top-down consistent with dependencies.
- **Reconstruction.** Save decisions or backtrack by comparing neighboring states.
- **Complexity.** time ≈ #states × work/state; space ≈ #states stored.

2 Knapsack Problem (Detailed & Space-Optimized)

Problem

Given items with weights w_i and values v_i and capacity W, choose at most one of each to maximize $\sum v_i$ s.t. $\sum w_i \le W$.

2D DP Recurrence (0–1 Knapsack)

Let $OPT(i, w)$ be the best value using the first i items within capacity w:

$$OPT(i,w) = \begin{cases} OPT(i-1,w), & w < w_i, \\ \max\big(OPT(i-1,w),\ v_i + OPT(i-1, w-w_i)\big), & w \ge w_i. \end{cases}$$

Bottom-up algorithm

```
Array M[0..n][0..W];  M[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]
```

Time $O(nW)$, space $O(nW)$.

Linear-Space Optimization ($O(W)$ space)

Observation: Row i depends only on row $i-1$ ⇒ reuse one array $D[0..W]$.

Correct 1D algorithm (iterate weights *backwards*)

```
D[0..W] := 0
for i = 1..n:
  for w = W down to wi:            # descending!
    D[w] = max(D[w], vi + D[w - wi])
return D[W]
```

Why backwards? In $OPT(i,w)$ the term $OPT(i-1,w-w_i)$ must come from the *previous* row. If you iterate $w = 0..W$ (forwards), then $D[w-w_i]$ may already include item i (same pass), allowing multiple uses of the same item (unbounded knapsack). Iterating $w = W, W-1, \ldots, w_i$ preserves $D[w-w_i]$ as row $i-1$ until it is read.

Induction invariant (proof sketch). At the start of the pass for item i, $D[w] = OPT(i-1,w)$. For $w < w_i$, $D[w]$ unchanged $\Rightarrow OPT(i,w) = OPT(i-1,w)$. For $w \geq w_i$,

$$D[w] \leftarrow \max\big(OPT(i-1,w),\ v_i + OPT(i-1,w-w_i)\big) = OPT(i,w),$$

because $D[w]$ and $D[w-w_i]$ still hold *row* $i-1$ values when looping backwards.

Worked 1D Trace (tiny)

$W = 5$, items $(w,v) = (3,4), (2,3)$, $D = [0,0,0,0,0,0]$.

```
Item (3,4): w=5..3 -> D = [0,0,0,4,4,4]
Item (2,3): w=5..2 -> D = [0,0,3,4,4,7]
```

Answer $D[5] = 7$ (take both).

3 Notes from Slides and Textbook (concise)

- **Knapsack (0–1)**: pseudo-polynomial $O(nW)$ DP; space can be $O(W)$ via backward 1D update.
- **Sequence alignment**: gap penalty δ, mismatch costs α_{pq}; $O(mn)$ DP; shortest-path view on grid.
- **DP recipe**: define states, prove optimal substructure, base cases, evaluation order, reconstruction.

4 Solutions to Problem Set

1. Knapsack Table (by hand)

Items $(w_i, v_i) = (5,7), (2,6), (3,3), (2,1)$, capacity $W = 6$. Fill $M[i,w]$.

$i\backslash w$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	0	7	7
2	0	0	6	6	6	7	7
3	0	0	6	6	6	9	9
4	0	0	6	6	7	9	9

Optimal value $M[4,6] = 9$ (choose items $(2,6)$ and $(3,3)$ with total weight 5).

Linear-space note. The same instance can be solved with the 1D algorithm above using $O(W)$ space by looping $w = 6 \downarrow w_i$ for each item.

2. Sequence Alignment (APPLE vs PAPE)

Alphabet $\{A, E, L, P\}$, penalty matrix P:

	A	E	L	P
A	0	1	3	1
E	1	0	2	1
L	3	2	0	2
P	1	1	2	0

gap penalty $\delta = 2$.

Let $A[i,j]$ be min-cost to align $X[1..i]$ with $Y[1..j]$; $A[i,0]=2i$, $A[0,j]=2j$, and

$$A[i,j]=\min\big(\alpha_{x_iy_j}+A[i-1,j-1,\ \delta+A[i-1,j],\ \delta+A[i,j-1]\big).$$

For $X=$ "APPLE" ($m=5$), $Y=$ "PAPE" ($n=4$), the filled table A is:

	0	1	2	3	4
0	0	2	4	6	8
1	2	1	2	4	6
2	4	2	2	2	4
3	6	4	3	2	3
4	8	6	5	4	4
5	10	8	7	6	$\boxed{4}$

Minimum cost $=4$. One optimal alignment (via backtracking):

$$\begin{array}{ccccc} A & P & P & L & E \\ P & A & P & - & E \end{array}$$

(*Cost check:* $1+1+0+2+0=4$.)

3. Book Shop

Prices h_i, pages s_i, budget x, each book at most once.

Recurrence.

$$OPT(i,x)=\begin{cases} OPT(i-1,x), & x<h_i, \\ \max\big(OPT(i-1,x),\ s_i+OPT(i-1,x-h_i)\big), & x\ge h_i. \end{cases}$$

2D DP (baseline).

1. Initialize $M[0][x]=0$ for $x=0..X$.
2. For $i=1..n$, for $x=0..X$, apply the recurrence.
3. Answer $M[n][X]$.

Time $O(nX)$, space $O(nX)$.

Linear-space version ($O(X)$).

```
D[0..X] := 0
for i = 1..n:
  for x = X down to h[i]:
    D[x] = max(D[x], s[i] + D[x - h[i]])
return D[X]
```

Backward iteration ensures 0–1 usage (no repeated purchase of the same book).

4. Longest Palindromic Subsequence (LPS)

For string $S[1..n]$, let $L(i,j)$ be the LPS length in $S[i..j]$:

$$L(i,j)=\begin{cases} 1, & i=j, \\ 2, & i+1=j\ \wedge\ S[i]=S[j], \\ \max\big(L(i+1,j),L(i,j-1)\big), & S[i]\ne S[j], \\ 2+L(i+1,j-1), & S[i]=S[j]. \end{cases}$$

Bottom-up: fill by increasing interval length $\ell=1..n$. Return $L(1,n)$. Time $O(n^2)$, space $O(n^2)$. (*Reconstruction*: follow choices that achieved the max.)

5. Defending Zion (KT 6.8)

Given arrivals $x_1, \ldots, x_n$ and recharge function f, EMP used at time k after j idle secs kills $\min(x_k, f(j))$. Let $D[t]$ = max robots destroyed up to time t. The DP:

$$D[t] = \max\Big(D[t-1],\ \max_{1 \le j \le t} \big\{D[t-j] + \min(x_t,\ f(j))\big\}\Big), \quad D[0] = 0.$$

Evaluation: For $t = 1..n$, scan $j = 1..t$ (time $O(n^2)$). If f has special structure (e.g. concave), optimizations may apply. (*Schedule-EMP* greedy fails in general; counterexamples exist.)

Puzzle of the Week: The Blind Man

Take any 10 face-up/face-down cards as pile B; flip all of pile B. The number of face-up cards in A equals that in B (invariant: "ups in B after flip" = "ups in A initially among those moved").

5 Summary

- **Knapsack 0–1:** $OPT(i, w) = \max(OPT(i-1, w), v_i + OPT(i-1, w-w_i))$. Space-optimal 1D update must iterate w *downwards.*
- **Sequence alignment:** $A[i, j] = \min(\alpha_{x_i y_j} + A[i-1, j-1],\ \delta + A[i-1, j],\ \delta + A[i, j-1])$.
- **LPS:** interval DP; match ends or drop one end.
- **Zion:** charge-length choice j each time t: $D[t] = \max(D[t-1], \max_j\{D[t-j] + \min(x_t, f(j))\})$.
- **Complexities:** Knapsack $O(nW)$ time, $O(W)$ space; Alignment $O(mn)$; LPS $O(n^2)$; Zion $O(n^2)$.