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

Mads Richardt

1 General Methodology and Theory

Dynamic Programming Principles

- **Optimal substructure:** Break problem into overlapping subproblems.
- **Recurrence relation:** Express solution to larger subproblem in terms of smaller subproblems.
- **Evaluation order:** Fill DP table bottom-up or use memoized recursion (top-down).
- **Time/space complexity:** Usually proportional to number of subproblems.

Subset Sum and Knapsack

Problem: Given n items with weight w_i and value v_i, and capacity W, maximize

$$\sum_{i \in S} v_i \quad \text{s.t.} \quad \sum_{i \in S} w_i \leq W.$$

Recurrence:

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

Algorithm (bottom-up):

```
Array M[0..n][0..W]
Initialize M[0][w] = 0 for all w
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]
```

Complexity: $O(nW)$ time, $O(nW)$ space (can be optimized to $O(W)$).

Sequence Alignment

Problem: Given strings $X = x_1 \dots x_m$ and $Y = y_1 \dots y_n$, gap penalty δ, mismatch costs α_{pq}, find minimum-cost alignment.

Recurrence:

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

Algorithm:

```
Array A[0..m][0..n]
A[i][0] = i*, 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]
```

Complexity: $O(mn)$ time, $O(mn)$ space. Backtracking reconstructs alignment.

Longest Palindromic Subsequence

Let $S[1..n]$ be input string. Define $L(i,j)$ = length of longest palindromic subsequence in $S[i..j]$.

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

Algorithm builds $n \times n$ table.

Supporting Math Tools

- Max/min operators, recurrence solving.
- Pseudo-polynomial runtime: $O(nW)$ depends on W.
- Backtracking in DP tables for solution reconstruction.

2 Notes from Slides and Textbook

- Knapsack: dynamic programming with recurrence (6.11), $O(nW)$ time.
- Sequence alignment: recurrence (6.16), $O(mn)$ time and space, shortest-path graph interpretation.
- DP design steps:
  1. Formulate recursively.
  2. Define subproblems.
  3. Identify base cases.
  4. Choose memoization structure and evaluation order.

3 Solutions to Problem Set

Exercise 1: Knapsack

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

We fill DP table $M[i,w]$. Optimal solution: pick items (2,6) and (3,3), weight = 5, value = 9.

Exercise 2: Sequence Alignment

Strings: APPLE vs PAPE, gap penalty $\delta = 2$, mismatch costs as given.

DP table constructed; minimal alignment cost = 3. Alignment:

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

Exercise 3: Book Shop

Recurrence:

$$OPT(i,x) = \max\big(OPT(i-1,x),\ s_i + OPT(i-1,x-h_i)\big).$$

Algorithm:

1. Input: n, budget x, arrays h_i, s_i.
2. Initialize $D[0..x] = 0$.
3. For each book i: for $j = x$ down to h_i: $D[j] = \max(D[j], s_i + D[j-h_i])$.
4. Answer $= D[x]$.

Time: $O(nx)$, **Space:** $O(x)$.

Exercise 4: Longest Palindrome Subsequence

Recurrence: See general section. Build DP table of size $n \times n$. **Pseudocode:**

1. Initialize $L[i,i] = 1$.
2. For subseq length $\ell = 2..n$, fill table using recurrence.
3. Answer: $L[1,n]$.

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

Exercise 5: Defending Zion (KT 6.8)

Let $D[t]$ = max robots destroyed up to time t. Recurrence:

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

Compute in $O(n^2)$. Possible optimizations depending on $f(\cdot)$.

Puzzle: The Blind Man

Divide 52 cards into two piles: take any 10 cards, flip them over. Each pile then has same number of face-up cards.

4 Summary

- **Knapsack recurrence:** $OPT(i,w) = \max(OPT(i-1,w), v_i + OPT(i-1,w-w_i))$.
- **Sequence alignment recurrence:** $OPT(i,j) = \min(\alpha_{x_i y_j} + OPT(i-1,j-1), \delta + OPT(i-1,j), \delta + OPT(i,j-1))$.
- **Longest palindrome subsequence:** $L(i,j) = 2 + L(i+1,j-1)$ if $S[i] = S[j]$, else $\max(L(i+1,j), L(i,j-1))$.
- **Defending Zion:** $D[t] = \max(D[t-1], \max_{j \le t}\{D[t-j] + \min(x_t, f(j))\})$.
- **Complexities:** Knapsack $O(nW)$, Sequence alignment $O(mn)$, LPS $O(n^2)$, Zion $O(n^2)$.