Algorithms and Data Structures 2
Exam Notes
Week 2: Dynamic Programming 1

Mads Richardt

1. General Methodology and Theory

Dynamic Programming Principles

- Break problems into *overlapping subproblems* with *optimal substructure*.
- Use a recurrence relation to express solution of a large problem in terms of smaller ones.
- Implement either:
  1. **Top-down with memoization**: recursive + cache.
  2. **Bottom-up iteration**: fill table in order of subproblem dependencies.
- Running time = (number of subproblems) × (time per subproblem).
- Typical steps:
  1. Identify subproblems.
  2. Derive recurrence.
  3. Prove correctness (usually by induction).
  4. Analyze time/space.
  5. Reconstruct solution if needed.

Mathematical Tools

- Geometric sums: $\sum_{i=0}^{k} r^i = \frac{r^{k+1}-1}{r-1}$.
- Harmonic number: $H_n \approx \ln n + \gamma$.
- Logarithm rules: $\log_a b = \frac{\ln b}{\ln a}$.
- Recurrence solving (for DP analysis):

$$T(n) = T(n-1) + O(1) \implies O(n)$$
$$T(n) = T(n-1) + T(n-2) + O(1) \implies O(\varphi^n),\ \varphi = \tfrac{1+\sqrt{5}}{2}.$$

2. Notes from Slides and Textbook

Weighted Interval Scheduling (KT 6.1)

- Input: jobs $j = 1, \dots, n$, each with (s_j, f_j, v_j).
- Sort by finish time: $f_1 \le f_2 \le \cdots \le f_n$.
- Define $p(j)$ = largest $i < j$ with $f_i \le s_j$ (compatible).

- Recurrence:

$$OPT(j) = \begin{cases} 0 & j = 0, \\ \max\{v_j + OPT(p(j)),\ OPT(j-1)\} & j \geq 1. \end{cases}$$

- Running time: $O(n \log n)$ (sort + binary search for $p(j)$).

Job Planning (KT 6.2)

- Weekly jobs: revenue ℓ_i (low-stress) or h_i (high-stress).
- Constraint: if week i is high-stress, then week $i-1$ must be none.
- Recurrence:

$$OPT(i) = \max\{\ell_i + OPT(i-1),\ h_i + OPT(i-2)\}.$$

- Base: $OPT(0) = 0$, $OPT(1) = \max(\ell_1, h_1)$.

Office Switching (KT 6.4)

- Costs: N_i (NY), S_i (SF), moving cost M.
- State: $OPT(i, NY)$ = min cost up to month i, ending in NY.
- Recurrence:

$$OPT(i, NY) = N_i + \min(OPT(i-1, NY),\ OPT(i-1, SF) + M),$$
$$OPT(i, SF) = S_i + \min(OPT(i-1, SF),\ OPT(i-1, NY) + M).$$

- Answer: $\min(OPT(n, NY), OPT(n, SF))$.

Grid Paths with Traps

- Grid $n \times n$, traps forbidden, moves: right or down.
- Let $P(i, j)$ = number of paths to (i, j).
- Recurrence:

$$P(i,j) = \begin{cases} 0 & \text{if trap at } (i,j), \\ 1 & \text{if } (i,j) = (1,1), \\ P(i-1,j) + P(i,j-1) & \text{otherwise.} \end{cases}$$

- Answer: $P(n, n)$.

Discrete Fréchet Distance

- Paths: $p_1, \ldots, p_n$, $q_1, \ldots, q_m$.
- Distance: $d(p, q)$ Euclidean.
- Define $L(i, j)$ = leash length needed to match $p_1 \ldots p_i$ with $q_1 \ldots q_j$.
- Recurrence:

$$L(i,j) = \max\big(d(p_i, q_j), \min\{L(i-1,j), L(i-1,j-1), L(i,j-1)\}\big).$$

- Base: $L(1, 1) = d(p_1, q_1)$.
- Answer: $L(n, m)$.

3. Solutions to Problem Set

Exercise 1: Weighted Interval Scheduling

Recursive with Memoization:

```
M[0..n] = empty
Compute-Opt(j):
  if j == 0: return 0
  if M[j] != empty: return M[j]
  M[j] = max(v[j] + Compute-Opt(p[j]), Compute-Opt(j-1))
  return M[j]
```

Iterative:

```
M[0] = 0
for j = 1..n:
  M[j] = max(v[j] + M[p[j]], M[j-1])
return M[n]
```

Exercise 2: Grid Paths

```
Paths[1..n][1..n]
for i = 1..n:
  for j = 1..n:
    if trap(i,j): Paths[i][j] = 0
    else if (i,j) == (1,1): Paths[i][j] = 1
    else: Paths[i][j] = Paths[i-1][j] + Paths[i][j-1]
return Paths[n][n]
```

Running time $O(n^2)$.

Exercise 3: Job Planning

```
OPT[0] = 0
OPT[1] = max(l1, h1)
for i = 2..n:
  OPT[i] = max(l[i] + OPT[i-1], h[i] + OPT[i-2])
return OPT[n]
```

Exercise 4: Office Switching

```
OPT_NY[1] = N1
OPT_SF[1] = S1
for i = 2..n:
  OPT_NY[i] = N[i] + min(OPT_NY[i-1], OPT_SF[i-1] + M)
  OPT_SF[i] = S[i] + min(OPT_SF[i-1], OPT_NY[i-1] + M)
return min(OPT_NY[n], OPT_SF[n])
```

Exercise 5: Discrete Fréchet Distance

```
for i = 1..n:
  for j = 1..m:
    if i == 1 and j == 1:
      L[i][j] = d(p1,q1)
    else if i == 1:
      L[i][j] = max(d(pi,qj), L[i][j-1])
```

```
    else if j == 1:
      L[i][j] = max(d(pi,qj), L[i-1][j])
    else:
      L[i][j] = max(d(pi,qj), min(L[i-1][j], L[i-1][j-1], L[i][j-1]))
return L[n][m]
```

Running time $O(nm)$, space $O(nm)$.

4. Summary

- Weighted Interval Scheduling: $OPT(j) = \max(v_j + OPT(p(j)), OPT(j-1))$.
- Job Planning: $OPT(i) = \max(\ell_i + OPT(i-1),\ h_i + OPT(i-2))$.
- Office Switching: $OPT(i, city) = \text{cost} + \min(\dots)$.
- Grid Paths: $P(i,j) = P(i-1,j) + P(i,j-1)$ (skip traps).
- Discrete Fréchet Distance: $L(i,j) = \max(d(p_i, q_j), \min\{\dots\})$.