

COMP 550 · Midterm 2 Cheat Sheet

Chapters 4–5: dynamic programming and shortest paths · Tue Nov 3 · expect 65 min, closed book, 5 × 5 pts · Problem 1 = five 1-pt traces, Problems 2–5 = design (algorithm, correctness, running time; ≥2 likely from PS 4–5)

Writing a DP answer

Four steps. (1) **Subproblems:** "for every i, OPT[i] = ...", and which entry is returned. (2) **Recurrence** + base cases. (3) **Pseudocode:** array d, filled in an order where everything needed is ready. (4) **Running time** = #subproblems × time per subproblem (+ precomputation). **Correctness** = the optimal solution falls into one of these cases (list them, say why complete), and the formula scores each case correctly.

Pattern	OPT	Fill	Return
1 prefix	answer for A[1:i]	i = 1..n	d[n]
2 suffix	answer for A[i:n]	i = n..1	d[1]
3 LIS trick	for A[1:i], must use A[i]	i = 1..n	max/min(d)
4 interval	for A[i:j]	bottom row up	d[1][n]
5 budget	for (A[1:i], j ≤ B)	row by row	d[n][B]
6 two strings	for (A[1:i], B[1:j])	row by row	d[m][n]
tree	for subtree T(u)	children first	root
DAG	for vertex v	topological	array d

Bottom-up loop (recommended). **Top-down** = recursion + memoization (global d, return if d[n] ≠ 0). Pure recursion on Fibonacci: O(φⁿ). **Space saving:** keep only what the recurrence reads (not possible for LIS); loses parent pointers.

One-index array DPs

Max: OPT[i] = max(OPT[i−1], A[i]). **Sum:** OPT[i−1] + A[i].

Fibonacci: OPT[i−1] + OPT[i−2], base 1, 1.

```
LIS(A): d = [1]*n # O(n^2), LIS trick
for i = 2..n: for j = 1..i-1:
    if A[j] < A[i] and d[i] < 1+d[j]: d[i] = 1+d[j] (p[i] = j)
return max(d)
```

Why: best subsequence ending at A[i] is [A[i]] or extends the best one ending at some A[j] < A[i], j < i. [5,2,5,6,3,7,4,6] → d = [1,1,2,3,2,4,3,4]. **LCIS** (contiguous): d[i] = d[i−1]+1 if A[i] > A[i−1] else 1, O(n).

Jumps 1 or 2, cost A[i]: OPT[i] = min cost to land and stop on i; d[1]=A[1], d[2]=A[2], d[i] = A[i] + min(d[i−1], d[i−2]); return min(d[n−1], d[n]). **Stock:** d[j] = A[j] − min(A[1:j−1]) with a running min, return max(d), O(n). **Max-sum subarray:** d[i] = A[i] + max(0, d[i−1]), return max(d). **Non-adjacent sum:** d[i] = max(d[i−1], A[i] + d[i−2]).

Alice & Bob (take 1–3 from the left): S[i] = suffix sums; OPT[i] = S[i] − min(OPT[i+1], OPT[i+2], OPT[i+3]); OPT[i] = S[i] for i ≥ n−2; fill n..1; return OPT[1]; O(n).

[4,1,5,2,3,10,1,2] → 18.

Two-index DPs

LPS (longest palindromic subsequence): OPT[i][j] for A[i:j]; OPT[i][i] = 1, empty = 0; A[i] = A[j] ⇒ 2 + OPT[i+1][j−1], else max(OPT[i+1][j], OPT[i][j−1]); fill i = n..1, j = i..n; return d[1][n]; O(n²). acbcbda → 5.

```
Knapsack(v,w,B): d[i][0]=0; d[1][j]=v[1] if w[1]≤j else 0
for i=2..n: for j=1..B:
    d[i][j] = d[i-1][j]
    if w[i] ≤ j: d[i][j] = max(d[i][j], v[i] + d[i-1][j-w[i]])
return d[n][B] # O(nB)
```

Skip item i, or take it and solve items 1..i−1 with capacity j−w[i]. **Unbounded:** v[i] + d[i][j−w[i]], base [j/w[1]] · v[1]. **One array:** j from B down to 0 (upward ⇒ unbounded). **Subset sum:** same with true/false, O(nk). v=[4,2,3,5], w=[3,1,2,2], B=5 → 10.

```
Edit-Distance(A,B): d[i][0]=i; d[0][j]=j
for i=1..m: for j=1..n:
    δ = 0 if A[i]=B[j] else 1
    d[i][j] = min(d[i][j-1]+1, d[i-1][j]+1, d[i-1][j-1]+δ)
return d[m][n] # O(mn)
```

left = insert, up = delete, diagonal = replace/keep; match ⇒ copy diagonal. ED(star, water) = 3. **LCS:** match ⇒ diag + 1, else max(up, left), base 0. **Common substring:** match ⇒ diag + 1, else 0, return max. Min deletions to equalize = m + n − 2 · LCS.

Trees (root 1, T[u] = children, n−1 edges)

Order: **children before parents** (reverse BFS order / DFS post). Each vertex is a child once ⇒ O(n).

Max-weight independent set: in[u] = w[u] + Σ_{v∈T[u]} out[v]; out[u] = Σ max(in[v], out[v]); leaf (w[u], 0); return max(in[1], out[1]). If u in S, children out; if u out, children free; subtrees independent. One-array form: max(Σ OPT[children], w[u] + Σ OPT[grandchildren]).

Max-weight matching: out[u] = Σ OPT[v]; OPT[u] = max(out[u], max_v w(u,v) + out[v] + (out[u] − OPT[v])); leaf 0; return OPT[1]. The subtraction replaces Σ_{x≠v} OPT[x] and makes it O(n) instead of O(n²).

Vertex cover: in = w + Σ min(in, out), out = Σ in.

Downward path: w[u] + max(0, max child), return max.

Connected piece with root: w[u] + Σ max(0, OPT[v]).

Subtree sums: w[u] + Σ OPT[v]. General-graph MIS: no known polynomial algorithm (P vs NP).

Shortest paths (SSSP: $d[v]$ = distance from s)

Algorithm	Needs	Time
BFS	all lengths equal, > 0	$O(m+n)$
DAG-DP	no cycles (negative ok)	$O(m+n)$
Dijkstra	all $\ell(e) \geq 0$	$O(n^2) / O(m \log n)$
Bellman-Ford	no negative cycle	$O(mn)$

DAG-DP(G,s): $d=[\infty]*n$; $d[s]=0$; topo-sort V ; $G_R = \text{reverse}$
for v in topo order: for u in $G_R[v]$:
 $d[v] = \min(d[v], d[u] + l(u,v))$

Last edge of a shortest path is (u,v) for an in-neighbor u ;
topological order has all in-neighbors done. **Longest**: max, start $-\infty$. **Widest**: max of $\min(d[u], w(u,v))$, $d[s]=\infty$ others 0.
Count paths: sum, $d[s]=1$.

Bellman-Ford: $d[v][0]=\infty$ except $d[s][0]=0$
for $j=1..n-1$: for v : $d[v][j] = \min(d[v][j-1],$
min over u in $G_R[v]$ of $d[u][j-1] + l(u,v))$
return column $n-1$ # $O(mn)$
BF-2: $d=[\infty]$, $d[s]=0$; repeat $n-1$ times: for each (u,v) :
 $d[v] = \min(d[v], d[u]+l(u,v))$ # "relax"

$OPT[v][j]$ = shortest *walk* with $\leq j$ edges; $n-1$ edges suffice (no negative cycle $\Rightarrow$ a shortest walk is a path). BF-2 may run ahead of the table mid-way; equal at $j = 0$ and $n-1$. Extra round still improving $\Rightarrow$ negative cycle. "At most k edges" $\Rightarrow k$ columns, $O(mk)$.

Dijkstra(G,s): $d=[\infty]*n$; $d[s]=0$; $S=\emptyset$
while $|S|<n$: $u = \text{unfinished vertex with min } d[u]$; add u to S
for v in $G[u]$: $d[v] = \min(d[v], d[u]+l(u,v))$

Chosen value is final (needs $\ell \geq 0$; $(1,2):1,(1,3):3,(3,2):-5$ breaks it). Array: n scans $\times O(n) = O(n^2)$. Heap: n extract-min + $\leq m$ decrease-key at $O(\log n) = O(m \log n)$. Not Prim: MST $\neq$ shortest-path tree $(1-2-3-4$ of $1s$ vs edge $1-4$ of $2)$. Lecture trace: edges $(1,2)3 (1,4)10 (2,3)1 (2,4)4 (3,4)2 (3,5)3 (4,5)1 \rightarrow$ order $1,2,3,4,5$, $d=[0,3,4,6,7]$. Longest walk with a cycle is unbounded. Through x : $Dijkstra(s)[x] + Dijkstra(x)[t]$. One free edge: two copies of each vertex.

Traps

Return $\max(d)$ with the LIS trick, not $d[n]$. State base cases; fill in an order where needed entries exist (suffix: right to left; interval: bottom up; tree: children first; DAG: topological). Edit distance needs row/column 0 (empty prefix). Knapsack one-array must go right to left. Bellman-Ford needs "no negative cycle", Dijkstra needs " $\ell \geq 0$ ", DAG-DP needs "no cycles". Running time = #subproblems $\times$ per-subproblem; say "each list scanned once $\Rightarrow O(m+n)$ " for graphs; "each vertex is a child once $\Rightarrow O(n)$ " for trees.