Handout: Dynamic Programming

Tools

Common subproblems

Finding the right subproblem takes creativity and experimentation. But there are a few standard choices that seem to arise repeatedly in dynamic programming.

i. The input is $x_1, x_2, \ldots, x_n$ and a subproblem is $x_1, x_2, \ldots, x_i$.

$$\boxed{x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6} \quad x_7 \quad x_8 \quad x_9 \quad x_{10}$$

The number of subproblems is therefore linear.

ii. The input is $x_1, \ldots, x_n$, and $y_1, \ldots, y_m$. A subproblem is $x_1, \ldots, x_i$ and $y_1, \ldots, y_j$.

$$\boxed{x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6} \quad x_7 \quad x_8 \quad x_9 \quad x_{10}$$

$$\boxed{y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5} \quad y_6 \quad y_7 \quad y_8$$

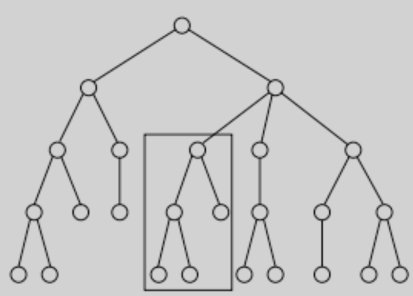
The number of subproblems is $O(mn)$.

iii. The input is $x_1, \ldots, x_n$ and a subproblem is $x_i, x_{i+1}, \ldots, x_j$.

$$x_1 \quad x_2 \quad \boxed{x_3 \quad x_4 \quad x_5 \quad x_6} \quad x_7 \quad x_8 \quad x_9 \quad x_{10}$$

The number of subproblems is $O(n^2)$.

iv. The input is a rooted tree. A subproblem is a rooted subtree.



If the tree has n nodes, how many subproblems are there?

(textbook, chapter 6)

Examples

Fibonacci. 1D table.
B.C. F(1) = F(2) = 1.
Recurrence F(n) = F(n-2) + F(n-1)
O(n) time.

Decomposition into Dictionary Words (covered in class, slides). 1D table.
B.C. T(0) = 0.
Recurrence: $T(k) = OR_{j = 1 \text{ to } k}$ (T(j-1) AND dict(x[j..k]))
$O(n^2)$ time.

Pascal's Triangle. 2D table.
B.C. C(n,0) = 1
Recurrence C(n,k) = C(n-1,k) + C(n-1,k-1)
$O(n^2)$ time.

Longest Common Subsequence. 2D table.
B.C. $T_{i,0} = T_{0,j} = 0$
Recurrence $T_{ij} = \max \{T_{i-1,j-1} + 1 \text{ (if } x_i = y_j) , T_{i-1,j} , T_{i,j-1} \}$
$O(n^2)$ time.

Bellman-Ford (SSSP) (relax #edges k in SP from source). 2D table.
B.C. $l_i^{(1)} = d_{0j}\ \forall\ j = 1, ..., n-1$
Recurrence $l_j^{(k)} = \min \{ l_j^{(k-1)} , \min_{i=1 \text{ to } n} (l_i^{(k-1)} + d_{ij}) \}$
// k advanced O(V) times from 1 to n-1; min inside { } requires O(E) time over all vertices j
O(VE) time (p. 117 in textbook).

Naive APSP (relax #edges m). 3D table.
B.C. $d_{ij}^{(0)} = 0$ if i = j; = ∞ otherwise
Recurrence $d_{ij}^{(m)} = \min_k (d_{ik}^{(m-1)} + w_{kj})$
// m → n-1 passes; d_{ij} → n^2 table entries ; $\min_k$ → O(n)
$O(n^4)$ time.

Floyd APSP (relax vertices allowed in paths, analogous to Dijkstra)
// Terminology: $c_{ij}^{(m)}$ = i-j shortest path cost with intermediate vertices $\in \{v_1, v_2, \ldots, v_m\}$
B.C. $c_{ij}^{(0)} = w_{ij}$ // no intermediate vertices allowed
Recurrence $c_{ij}^{(m)} = \min \{ c_{ij}^{(m-1)} , c_{im}^{(m-1)} + c_{mj}^{(m-1)} \}$
// m → O(n) passes ; c_{ij} → n^2 table entries ; min{} is O(1) operation
$O(n^3)$ time.

DP. Handout adapted from Prof. Impagliazzo (on class website).