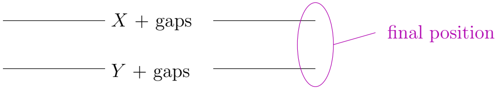
Dynamic Programming

An Algorithm for Sequence Alignment

Algorithms: Design and Analysis, Part II

The Subproblems

(1) x_m & y_n, (2) x_m & gap, (3) y_n & gap



Optimal substructure: Let $X' = X - x_m$, $Y' = Y - y_n$.

If case (1) holds, then induced alignment of X' & Y' is optimal.
If case (2) holds, then induced alignment of X' & Y is optimal.
If case (3) holds, then induced alignment of X & Y' is optimal.

Relevant subproblems: Have the form (X_i, Y_i) where
X_i = 1st i letters of X
Y_j = 1st j letters of Y
[Since only peel off letters from the right ends of the strings]

The Recurrence

Notation: P_{ij} = penalty of optimal alignment of X_i & Y_j.

Recurrence: For all $i = 1, \ldots, m$ and $j = 1, \ldots, n$:

$$P_{ij} = \min \left\{ \begin{array}{ll} (1) & \alpha_{x_i y_j} + P_{i-1,j-1} \\ (2) & \alpha_{\text{gap}} + P_{i-1,j} \\ (3) & \alpha_{\text{gap}} + P_{i,j-1} \end{array} \right\}$$

Correctness: Optimal solution is one of these 3 candidates, and recurrence selects the best of these.

Base Cases

Question: What is the value of $P_{i,0}$ and $P_{0,i}$?

A) 0
B) $\boxed{i \cdot \alpha_{\text{gap}}}$
C) $+\infty$
D) Undefined

The Algorithm

A = 2-D array.

$A[i,0] = A[0,i] = i \cdot \alpha_{\text{gap}}, \forall i \geq 0$

For $i = 1$ to m

For $j = 1$ to n

$$A[i,j] = \min \begin{Bmatrix} (1) & \text{A[i-1,j-1]} + \alpha_{x_i y_j} \\ (2) & \text{A[i-1,j]} + \alpha_{\text{gap}} \\ (3) & \text{A[i,j-1]} + \alpha_{\text{gap}} \end{Bmatrix}$$

All available for $O(1)$-time lookup!

Correctness: [i.e., $A[i,j] = P_{ij}, \forall i,j \geq 0$] Follows from induction + correctness of recurrence.

Running time: $O(mn)$ [$\Theta(1)$ work for each of $\Theta(mn)$ subproblems]

Reconstructing a Solution

- Trace back through filled-in table A, starting $A[m, n]$
- When you reach subproblem $A[i, j]$:
  - If $A[i, j]$ filled using case (1), match x_i & y_j and go to $A[i-1, j-1]$
  - If $A[i, j]$ filled using case (2), match x_i with a gap and go to $A[i-1, j]$
  - If $A[i, j]$ filled using case (3), match y_j with a gap and go to $A[i, j-1]$

[If $i = 0$ or $j = 0$, match remaining substring with gaps]

Running time is only $O(m + n)$!