Dynamic Programming Recipe

Step 1: Devise simple recursive algorithm
- Flavor: make “first choice”, then recursively solve subproblem

Step 2: Write recurrence for optimal value

Step 3: Design bottom-up iterative algorithm
- Weighted interval scheduling: first-choice is binary
- Rod-cutting: first choice has \( n \) options
- Subset Sum: need to “add a variable” (one more dimension)
- Now: similarity between sequences

Sequence Alignment

Example. TAIL vs TALE

For two strings \( X = x_1x_2 \ldots x_m \), \( Y = y_1y_2 \ldots y_n \), an alignment \( M \) is a matching between \( \{1, \ldots, m\} \) and \( \{1, \ldots, n\} \).

\( M \) is valid if
- **Matching.** Each element appears in at most one pair in \( M \).
- **No crossings.** If \((i,j),(k,\ell) \in M\) and \( i < k \), then \( j < \ell \).

Cost of \( M \):
- **Gap penalty.** For each unmatched character, you pay \( \delta \).
- **Alignment cost.** For a match \((i,j)\), you pay \( C(x_i, y_j) \).

\[
\text{cost}(M) = \delta(m + n - 2|M|) + \sum_{(i,j) \in M} C(x_i, y_j).
\]

Problem. Given strings \( X, Y \) gap-penalty \( \delta \) and cost matrix \( C \), find valid alignment of minimal cost.

Example 1. TAIL vs TALE, \( \delta = 0.5 \), \( C(x, y) = 1[x \neq y] \).
Example 2. TAIL vs TALE, \( \delta = 10 \), \( C(x, y) = 1[x \neq y] \).
Example Recap

**Example 1.** TAIL vs TALE, $\delta = 0.5$, $C(x,y) = 1[x \neq y]$.

TA-IL not matched (gap)
TA-LE not matched (gap)

**Example 2.** TAIL vs TALE, $\delta = 10$, $C(x,y) = 1[x \neq y]$.

TAIL
TALE

Applications

**Genomics**
- Biologists use genetic similarity to determine evolutionary relationships.
- Genetic similarity = cost of aligning DNA sequences

**Spell-checkers, diff program, search engines.**
- "preffered": (0) proffered (1) preferred (2) referred ...
Toward an Algorithm

Let $O$ be optimal alignment. Is pair $(m,n)$ matched in $O$?

- If $(m,n) \in O$ we can align $x_1x_2...x_{m-1}$ with $y_1y_2...y_{n-1}$.
- If $(m,n) \notin O$ then either $x_m$ or $y_n$ must be unmatched (by no crossing).

Value $\text{OPT}(m,n)$ of optimal alignment is one of:

- $C(x_m,y_n) + \text{OPT}(m-1,n-1)$, If $(m,n)$ matched
- $\delta + \text{OPT}(m-1,n)$, If $m$ unmatched
- $\delta + \text{OPT}(m,n-1)$, If $n$ unmatched

Recurrence

Let $\text{OPT}(i,j)$ be optimal alignment cost of $x_1x_2...x_i$ and $y_1y_2...y_j$.

\[
\text{OPT}(i,j) = \min \begin{cases} 
C(x_i,y_j) + \text{OPT}(i-1,j-1) \\
\delta + \text{OPT}(i-1,j) \\
\delta + \text{OPT}(i,j-1)
\end{cases}
\]

And $(i,j)$ is in optimal alignment $\iff$ first term is the minimum.

Base case?

- $\text{OPT}(0,j) = j\delta$ align $X = \emptyset$ to $Y = y_1...y_j$
- $\text{OPT}(i,0) = i\delta$ similar

Sequence Alignment Pseudocode

```pseudo
def align(X, Y):
    Initialize $M[0..m, 0..n] = \text{null}$
    $M[i, 0] = i\delta$, $M[0, j] = j\delta$ for all $i,j$
    for $j = 1, \ldots, n$ do
        for $i = 1, \ldots, m$ do
            $v_1 = C(x_i, y_j) + M[i-1,j-1]$
            $v_2 = \delta + M[i-1,j]$
            $v_3 = \delta + M[i,j-1]$
            $M[i,j] \leftarrow \min\{v_1, v_2, v_3\}$
    end for $i$
end for $j$
```

- Blue = recurrence, rest = DP “boilerplate”
- Running time? $\Theta(mn)$
- Example. TALE and TAIL, $\delta = 1$, $C(x,y) = 2 \cdot 1[x \neq y]$.

Sequence Alignment

- Recovering optimal matching: store each choice, trace back.
- Related to shortest path in weighted directed graph.

Graph has $\sim mn$ nodes and $\sim 3mn$ edges.
Dijkstra’s algorithm runs in $O(|E| \log |V|) \implies O(mn \log(mn))$ time for a graph with $\Theta(mn)$ nodes and edges. Sequence alignment takes only $O(mn)$ time. What can we conclude?

A. We could use dynamic programming to compute shortest paths in any graph asymptotically faster than Dijkstra’s algorithm.
B. By the multiplicativity property of big-O, the $\log |V|$ factor is dominated by $|E|$, so Dijkstra’s running time is $O(|E|) = O(mn)$.
C. The graph in sequence alignment is a special case where we can compute shortest paths faster.
D. Dijkstra’s algorithm only works on undirected graphs.

### Sequence Alignment in Linear Space

Hirschberg’s algorithm: clever combination of DP and divide-and-conquer

**Goal:** find shortest path from $(0,0) \rightarrow (m,n)$

**Board work**
1. $\text{OPT}(i,j) = f(i,j) = \text{length of shortest path from } (0,0) \rightarrow (i,j)$
2. For any $j$, can compute $f(\cdot,j)$ in $O(mn)$ time and $O(m+n)$ space
3. Let $g(i,j) = \text{length of shortest path from } (i,j) \rightarrow (m,n)$
4. For any $j$, can compute $g(\cdot,j)$ in $O(mn)$ time and $O(m+n)$ space
5. **Key idea:** find one node on shortest path. Fix $j = n/2$ and find $q$ to maximize $f(q,n/2) + g(q,n/2)$

$\implies$ node $(q,n/2)$ is on shortest path.
6. Recursively find shortest-path from $(0,0) \rightarrow (q,n/2)$
7. Recursively find shortest-path from $(q,n/2) \rightarrow (m,n)$.
8. Time $T(m,n) = T(q,n/2) + T(m-q,n/2) + O(mn)$. **Solves to $O(mn)$ (recursion tree)**

Space still $O(m+n)$.

### Can We Use Less Space?

We’ve focused on time complexity, but space matters too!

Two sequences of length $10^5$: $mn = 10^{10}$ (10 GB)

for $j = 1, \ldots, n$ do
  for $i = 1, \ldots, m$ do
    $v_1 = C(x_i, y_j) + M[i-1,j-1]$
    $v_2 = \delta + M[i-1,j]$
    $v_3 = \delta + M[i,j-1]$
    $M[i,j] \leftarrow \min\{v_1,v_2,v_3\}$

Can we save space?

- Computing column $M[\cdot,j]$ only needs $M[\cdot,j-1]$
  $\implies$ keep just two columns (current, previous)
  $\implies$ linear space $O(m+n)$
- But: can only compute cost, not recover alignment!

### Hirschberg’s Algorithm

**Divide.** Find index $q$ that minimizes $f(q,n/2) + g(q,n/2)$; save node $i$-$j$ as part of solution.

**Conquer.** Recursively compute optimal alignment in each piece.
Sequence Alignment: Summary

Align sequences $X, Y$
- Binary choice
- Recurse on prefixes
- $O(mn)$ time
- $O(m + n)$ space: more subtle
  - DP + Divide and Conquer

More sequences:
- RNA secondary structure
- match max. # of bases
- problem substructure:
  over intervals

---

Figure 6.13

An RNA secondary structure. Thick lines connect adjacent elements of the sequence; thin lines indicate pairs of elements that are matched.

The Problem

As one learns in introductory biology classes, Watson and Crick posited that double-stranded DNA is "zipped" together by complementary base-pairing. Each strand of DNA can be viewed as a string of bases, where each base is drawn from the set \{A, C, G, T\}. The bases A and T pair with each other, and the bases C and G pair with each other; it is these A-T and C-G pairings that hold the two strands together.

Now, single-stranded RNA molecules are key components in many of the processes that go on inside a cell, and they follow more or less the same structural principles. However, unlike double-stranded DNA, there’s no “second strand” for the RNA to stick to; so it tends to loop back and form base pairs with itself, resulting in interesting shapes like the one depicted in Figure 6.13. The set of pairs (and resulting shape) formed by the RNA molecule through this process is called the secondary structure, and understanding the secondary structure is essential for understanding the behavior of the molecule.