13 Sequence Alignment / Edit Distance [KT 6.6]

Definition. δ: gap penalty. $\alpha_{p,q}$: cost of matching $p, q \in \Sigma$ with $\alpha(p,p) = 0$ for all $p \in \Sigma$. We want to find an alignment between the two sequences $X \in \Sigma^m$ and $Y \in \Sigma^n$, such that the cost, which is measured by the *sum* of gap penalty and matching cost, is minimized.

Key observation. In an optimal alignment M, at least one of the following is true ($m = |X|, n = |Y|$). Proof by contradiction – otherwise it leads to crossing edges.

1. $(m, n) \in M$
2. $X[m]$ is not matched
3. $Y[n]$ is not matched

The recursion.

$$OPT(i,j) = \min\{\alpha_{x_i y_j} + OPT(i-1, j-1), \delta + OPT(i-1, j), \delta + OPT(i, j-1)\}.$$

Running time. Naive implementation needs $O(mn)$ time and space. We only need to compute at most $O(mn)$ of $OPT(\cdot, \cdot)$, and each can be computed in $O(1)$ time (from previously already computed values).

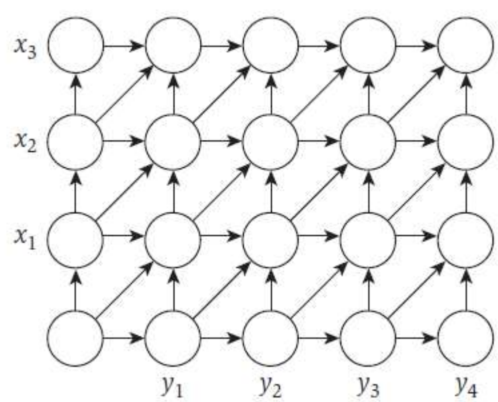

Figure 6.17 A graph-based picture of sequence alignment.

Equivalent to finding shortest path in a 2D grid. For example, to match 'mean' and 'name', suppose the gap penalty is $\delta = 2$, and $\alpha_{a,e} = \alpha_{m,n} = 1$, $\alpha_{a,m} = \alpha_{a,n} = \alpha_{e,m} = \alpha_{e,n} = 3$.

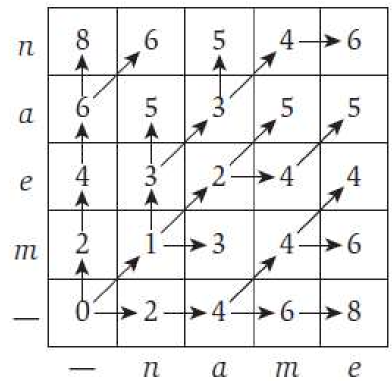

Figure 6.18 The OPT values for the problem of aligning the words *mean* to *name*.

Better space (value only). Better implementation can improve the space to $O(m+n)$. Easy to do from the shortest path representation: We compute the DP matrix row-by-row bottom-up, and at any time step we only keep the two most recent rows.

Why $O(mn)$ space is *not* good? Since for example for DNA sequences, m and n can be very large. Say if they have $1,000,000$ symbols, then the space would be 1 TB. On the other hand, if we have a 2.0GHz CPU, we can run an $O(mn)$ time algorithm in about 500 seconds, which is a fairly reasonable time.

Find the actual alignment in $O(m+n)$ space. The above $O(m+n)$ space implementation cannot be used to find the optimal solution.

A better algo uses divide-and-conquer, and recursively finds the middle point in the shortest path, via a *bi-direction* dynamic programming.

Let $f(i,j)$ denote the length of the shortest path from $(0,0)$ to (i,j), and $g(i,j)$ denote the length of the shortest path from (m,n) (back) to (i,j) (to better see this, one can rotate the DP matrix by 180 degree):

$$f(i,j) = \min\{\alpha_{x_i y_j} + f(i-1,j-1), \delta + f(i-1,j), \delta + f(i,j-1)\}.$$

$$g(i,j) = \min\{\alpha_{x_{i+1} y_{j+1}} + g(i+1,j+1), \delta + g(i+1,j), \delta + g(i,j+1)\}.$$

We have the following observation: The length of the shortest bottom-left corner to top-right corner path in the DP matrix that pass through (i,j) is $f(i,j)+g(i,j)$. More generally, let $k \in \{0,\ldots,n\}$, and let $q \in \{0,\ldots,m\}$ be an index that minimize $f(q,k)+g(q,k)$. Then there is a corner-to-corner path of minimum length that passes through the node (q,k). We then try out the q values one by one, and then find the one with the minimum cost. Finally, we recuse on two subproblems: to find the shortest path from $(0,0)$ to (q,k), and that from (k,q) to (n,m). See the Figure 6.19 for an illustration.

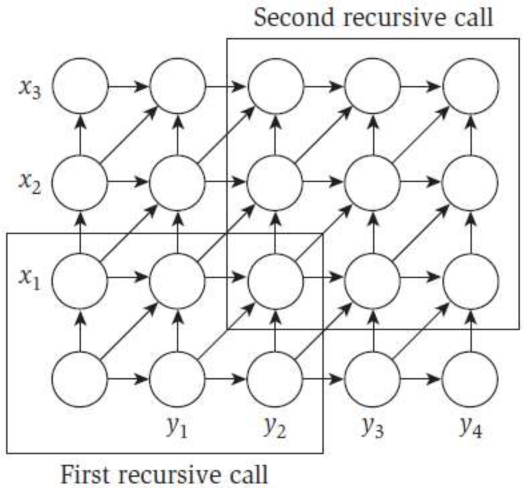

Figure 6.19 The first level of recurrence for the space-efficient `Divide-and-Conquer-Alignment`. The two boxed regions indicate the input to the two recursive cells.

The pseudocode of the algorithm is presented below.

Divide-And-Conquer-Alignment$(x_1, \ldots, x_n; y_1, \ldots, y_m)$

IF $(n \leq 2)$ or $(m \leq 2)$ THEN compute the best alignment using the old algorithm and stop;

Let $k = n/2$;

Compute $f[k, j]$, $g[k, j]$ for all $j \in \{0, 1, \ldots, m\}$ in time $O(nm)$ and space $O(n + m)$. Find a q such that $f[k, q] + g[k, q]$ is minimized;

Insert the alignment path point (k, q) in the global path array;

Divide-And-Conquer-Alignment$(x_1, \ldots, x_k; y_1, \ldots, y_q)$;

Divide-And-Conquer-Alignment$(x_{k+1}, \ldots, x_n; y_{q+1}, \ldots, y_m)$;

The running time analysis of this algorithm is to solve the following recursion.

$$T(n, m) \leq cmn + T(n/2, q) + T(n/2, m - q); T(2, m) \leq cm; T(n, 2) \leq cn.$$

Solve the recursion by guessing $T(m, n) \leq \alpha mn$, and solve for $\alpha = 2c$.