6. Dynamic Programming

Those who cannot remember the
past are condemned to repeat it

-Santayana

Algorithmic Paradigms

Greed. Build up a solution incrementally, myopically optimizing some local criterion.

Divide-and-conquer. Break up a problem into two sub-problems, solve each sub-problem independently, and combine solution to sub-problems to form solution to original problem.

Dynamic programming. Break up a problem into a series of overlapping sub-problems, and build up solutions to larger and larger sub-problems.

6.6 Sequence Alignment

String Similarity

How similar are two strings?

- **ocurrance**
- **occurrence**

o	c	u	r	r	a	n	c	e	-
o	c	c	u	r	r	e	n	c	e

5 mismatches, 1 gap

o	c	-	u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e

1 mismatch, 1 gap

o	c	-	u	r	r	-	a	n	c	e
o	c	c	u	r	r	e	-	n	c	e

0 mismatches, 3 gaps

Edit Distance

Applications.

- Basis for Unix diff.
- Speech recognition.
- Computational biology.

Edit distance. [Levenshtein 1966, Needleman-Wunsch 1970, Smith-Waterman 1981]

- Gap penalty δ; mismatch penalty α_{pq}.
- Cost = sum of gap and mismatch penalties.

C	T	G	A	C	C	T	A	C	C	T
C	C	T	G	A	C	T	A	C	A	T

$\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA}$

-	C	T	G	A	C	C	T	A	C	C	T
C	C	T	G	A	C	-	T	A	C	A	T

$2\delta + \alpha_{CA}$

Sequence Alignment

Goal: Given two strings $X = x_1\, x_2 \ldots x_m$ and $Y = y_1\, y_2 \ldots y_n$ find alignment of minimum cost.

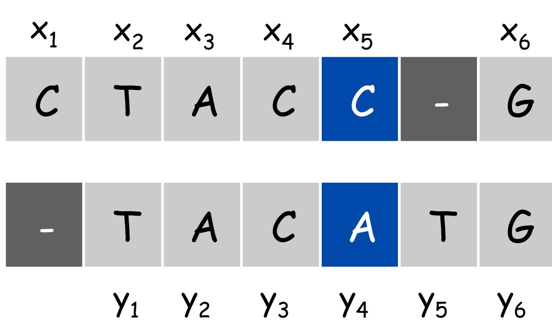
Def. An alignment M is a set of ordered pairs x_i-y_j such that each item occurs in at most one pair and no crossings.

Def. The pair x_i-y_j and $x_{i'}$-$y_{j'}$ cross if $i < i'$, but $j > j'$.

$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i\,:\, x_i \text{ unmatched}} \delta + \sum_{j\,:\, y_j \text{ unmatched}} \delta}_{\text{gap}}$$

Ex: CTACCG vs. TACATG.

Sol: $M = x_2$-y_1, x_3-y_2, x_4-y_3, x_5-y_4, x_6-y_6.



Sequence Alignment: Problem Structure

Def. OPT(i, j) = min cost of aligning strings $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j$.

- Case 1: OPT matches x_i-y_j.
  - pay mismatch for x_i-y_j + min cost of aligning two strings $x_1 x_2 \ldots x_{i-1}$ and $y_1 y_2 \ldots y_{j-1}$
- Case 2a: OPT leaves x_i unmatched.
  - pay gap for x_i and min cost of aligning $x_1 x_2 \ldots x_{i-1}$ and $y_1 y_2 \ldots y_j$
- Case 2b: OPT leaves y_j unmatched.
  - pay gap for y_j and min cost of aligning $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_{j-1}$

$$OPT(i, j) = \begin{cases} j\delta & \text{if } i = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if } j = 0 \end{cases}$$

Sequence Alignment: Algorithm

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α) {
   for i = 0 to m
      M[0, i] = iδ
   for j = 0 to n
      M[j, 0] = jδ

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
}
```

Analysis. $\Theta(mn)$ time and space.

English words or sentences: $m, n \leq 10$.

Computational biology: $m = n = 100{,}000$. 10 billions ops OK, but 10GB array?

6.7 Sequence Alignment in Linear Space

Sequence Alignment: Linear Space

Q. Can we avoid using quadratic space?

Easy. Optimal value in $O(m + n)$ space and $O(mn)$ time.

- Compute OPT(i, •) from OPT(i-1, •).
- No longer a simple way to recover alignment itself.

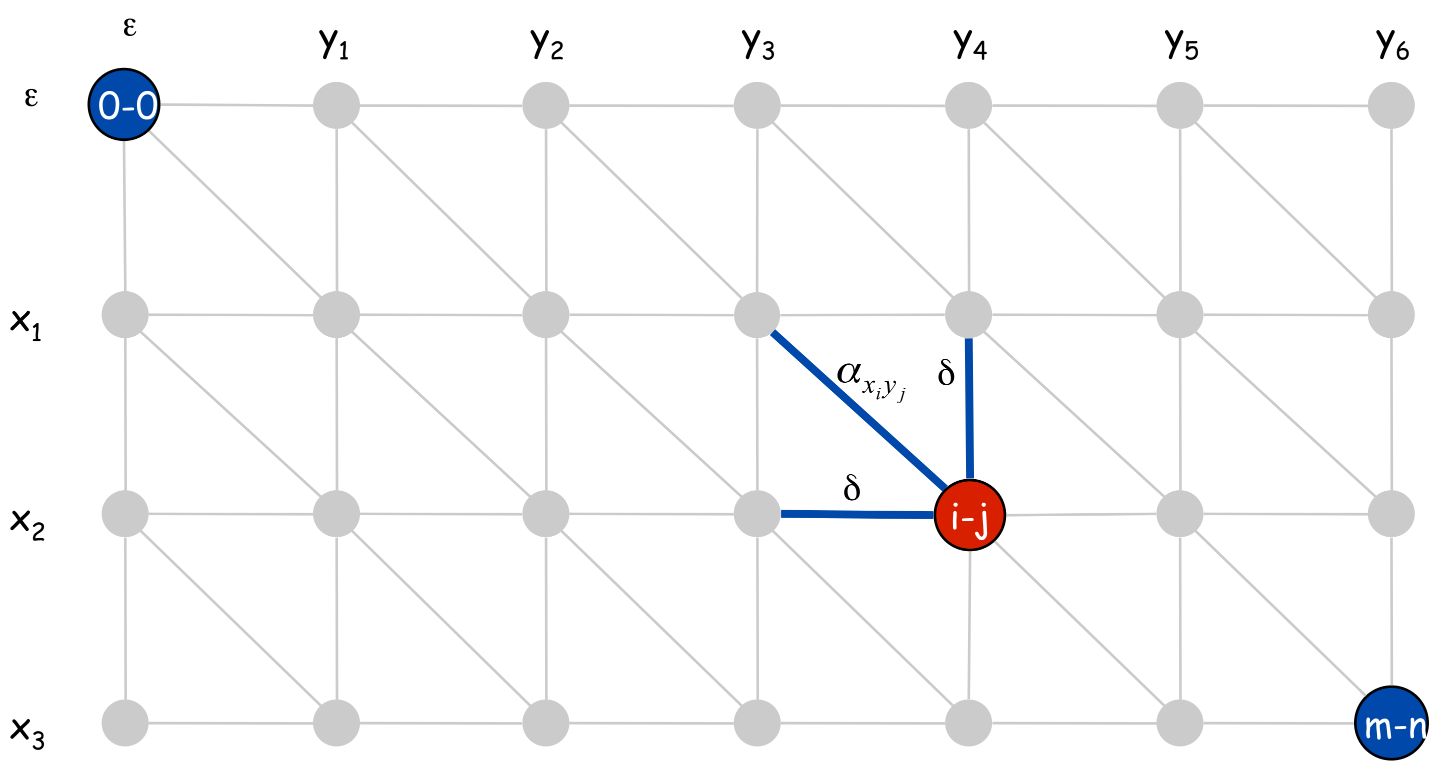
Theorem. [Hirschberg, 1975] Optimal alignment in $O(m + n)$ space and $O(mn)$ time.

- Clever combination of divide-and-conquer and dynamic programming.
- Inspired by idea of Savitch from complexity theory.

Sequence Alignment: Linear Space

Edit distance graph.

- Let f(i, j) be shortest path from (0,0) to (i, j).
- Observation: f(i, j) = OPT(i, j).



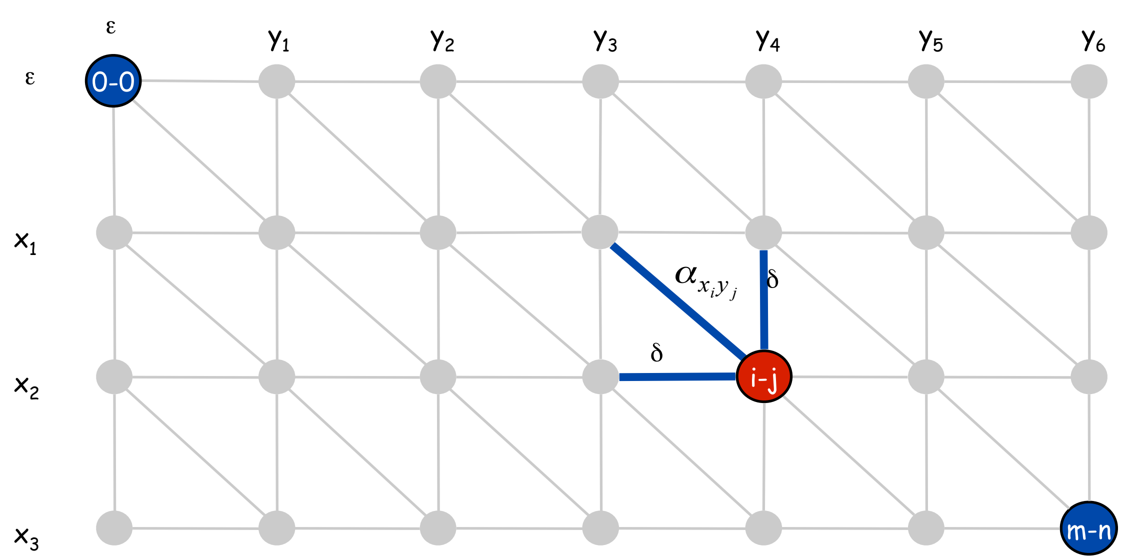
Sequence Alignment: Linear Space

Claim. f(i, j) = OPT(i, j).

Pf. (by induction on i + j)

- Base case: f(0, 0) = OPT(0, 0) = 0.
- Inductive step: assume f(i', j') = OPT(i', j') for all i' + j' < i + j.
- Last edge on path to (i, j) is either from (i-1, j-1), (i-1, j), or (i, j-1).

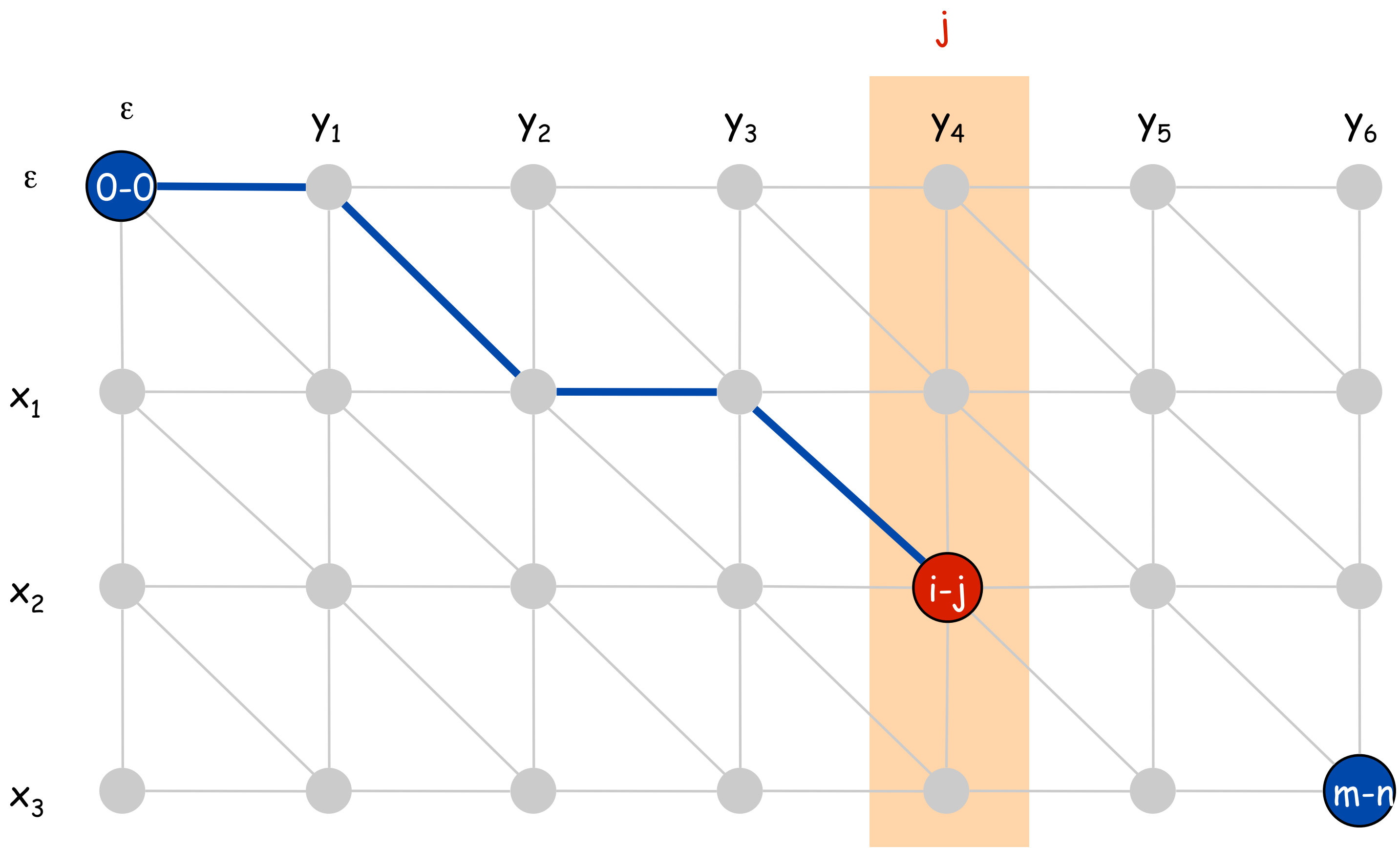
$$\begin{aligned} f(i,j) &= \min\{\alpha_{x_i y_j} + f(i-1, j-1),\ \delta + f(i-1, j),\ \delta + f(i, j-1)\} \\ &= \min\{\alpha_{x_i y_j} + OPT(i-1, j-1),\ \delta + OPT(i-1, j),\ \delta + OPT(i, j-1)\} \\ &= OPT(i,j) \end{aligned}$$



Sequence Alignment: Linear Space

Edit distance graph.

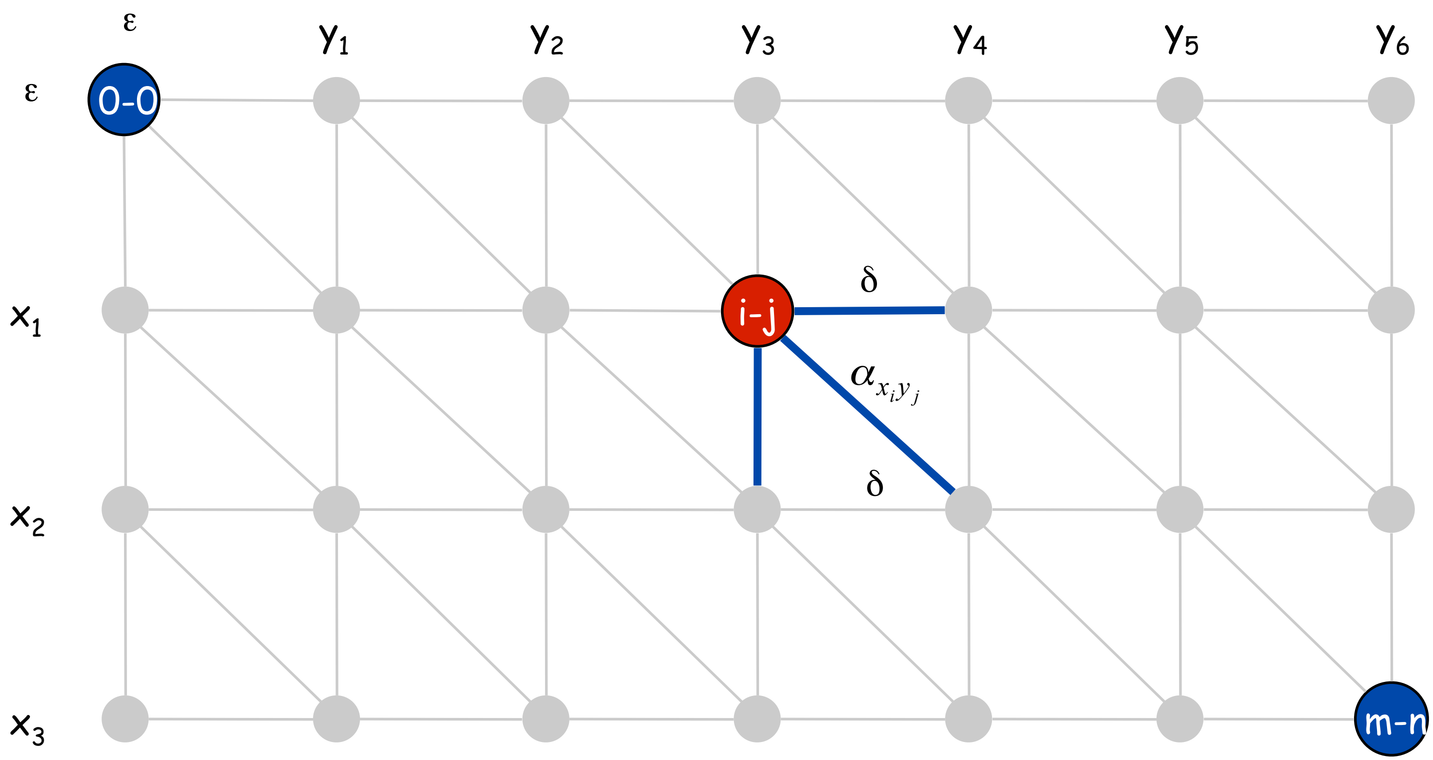
- Let f(i, j) be shortest path from (0,0) to (i, j).
- Can compute f (•, j) for any j in O(mn) time and O(m + n) space.



Sequence Alignment: Linear Space

Edit distance graph.

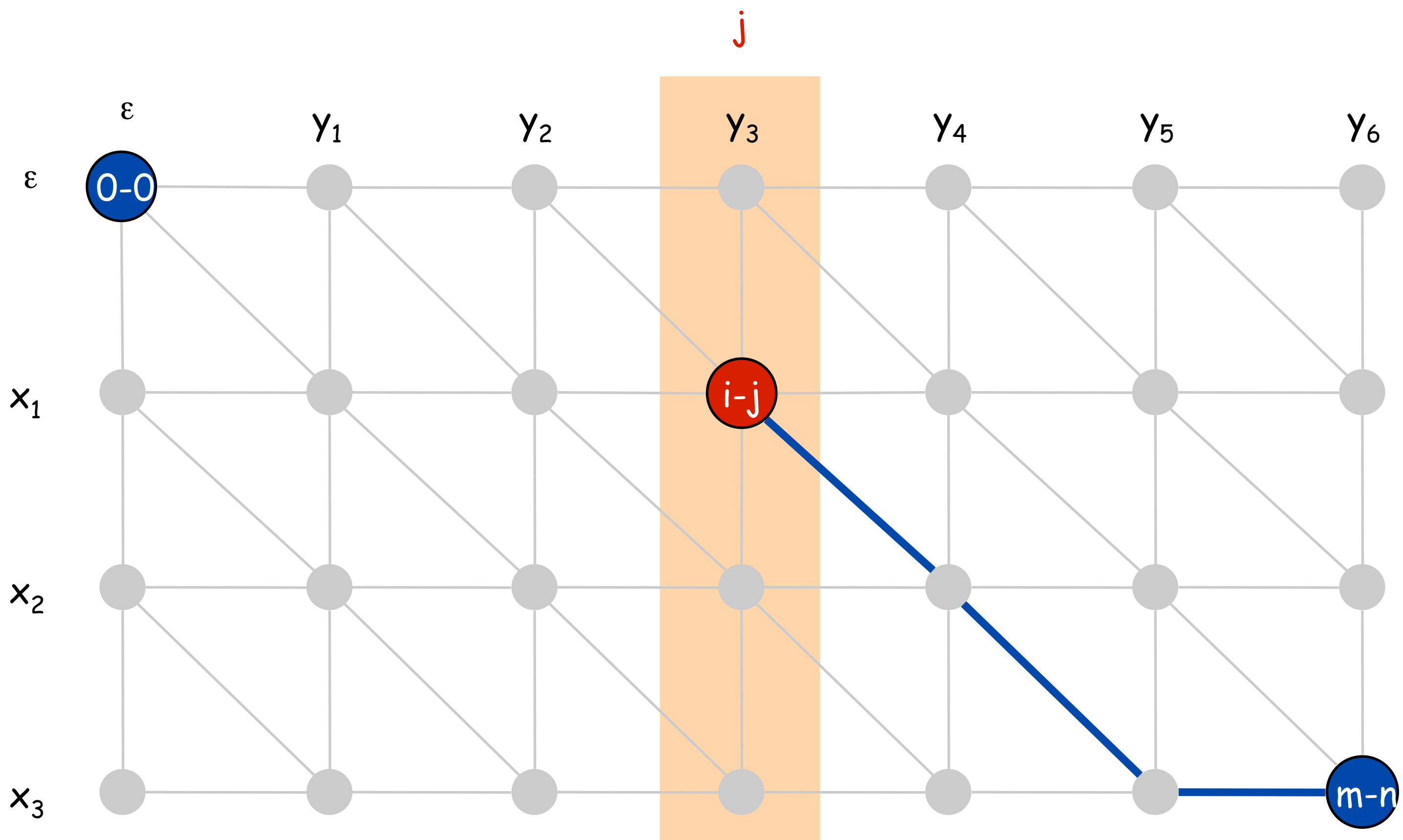
- Let $g(i, j)$ be shortest path from (i, j) to (m, n).
- Can compute by reversing the edge orientations and inverting the roles of $(0, 0)$ and (m, n)



Sequence Alignment: Linear Space

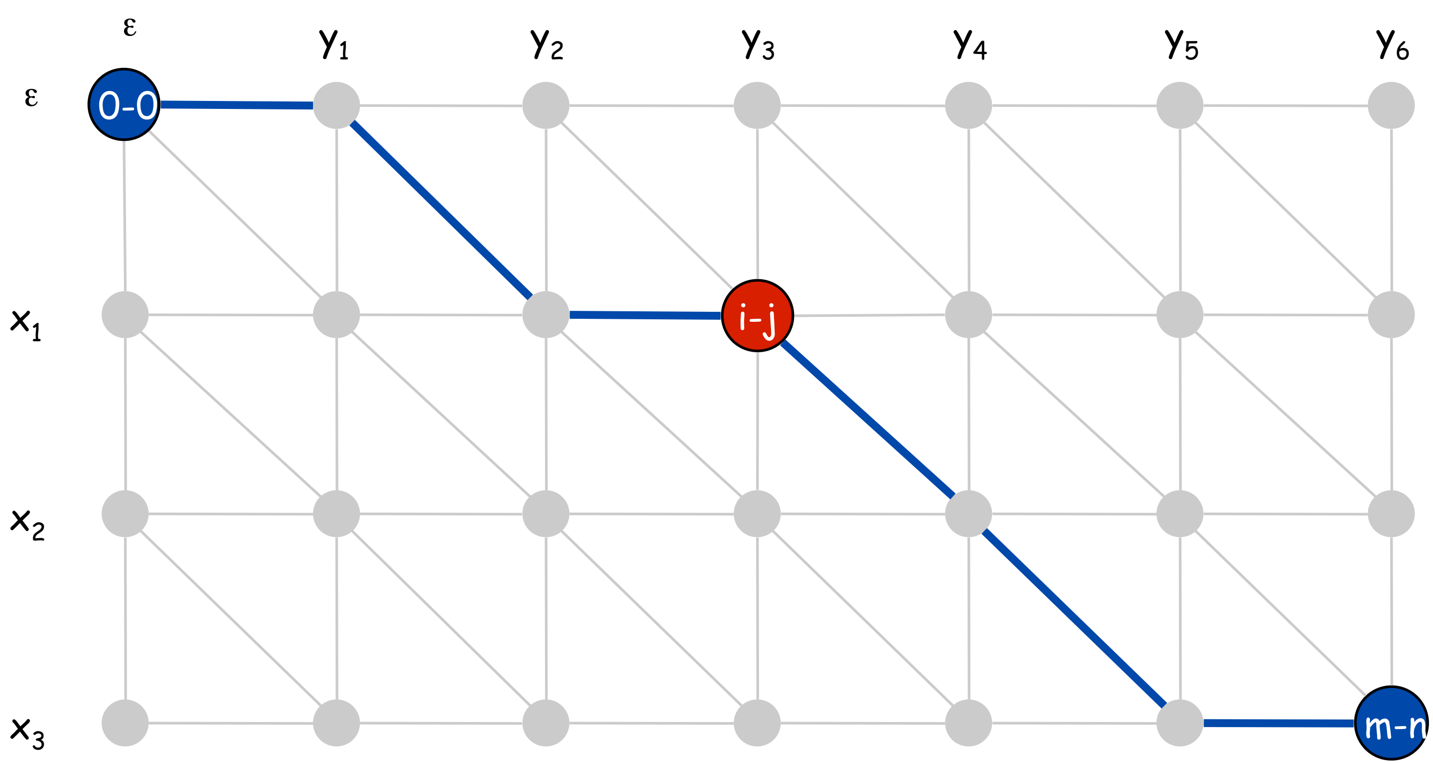
Edit distance graph.

- Let $g(i, j)$ be shortest path from (i, j) to (m, n).
- Can compute $g(\bullet, j)$ for any j in $O(mn)$ time and $O(m + n)$ space.



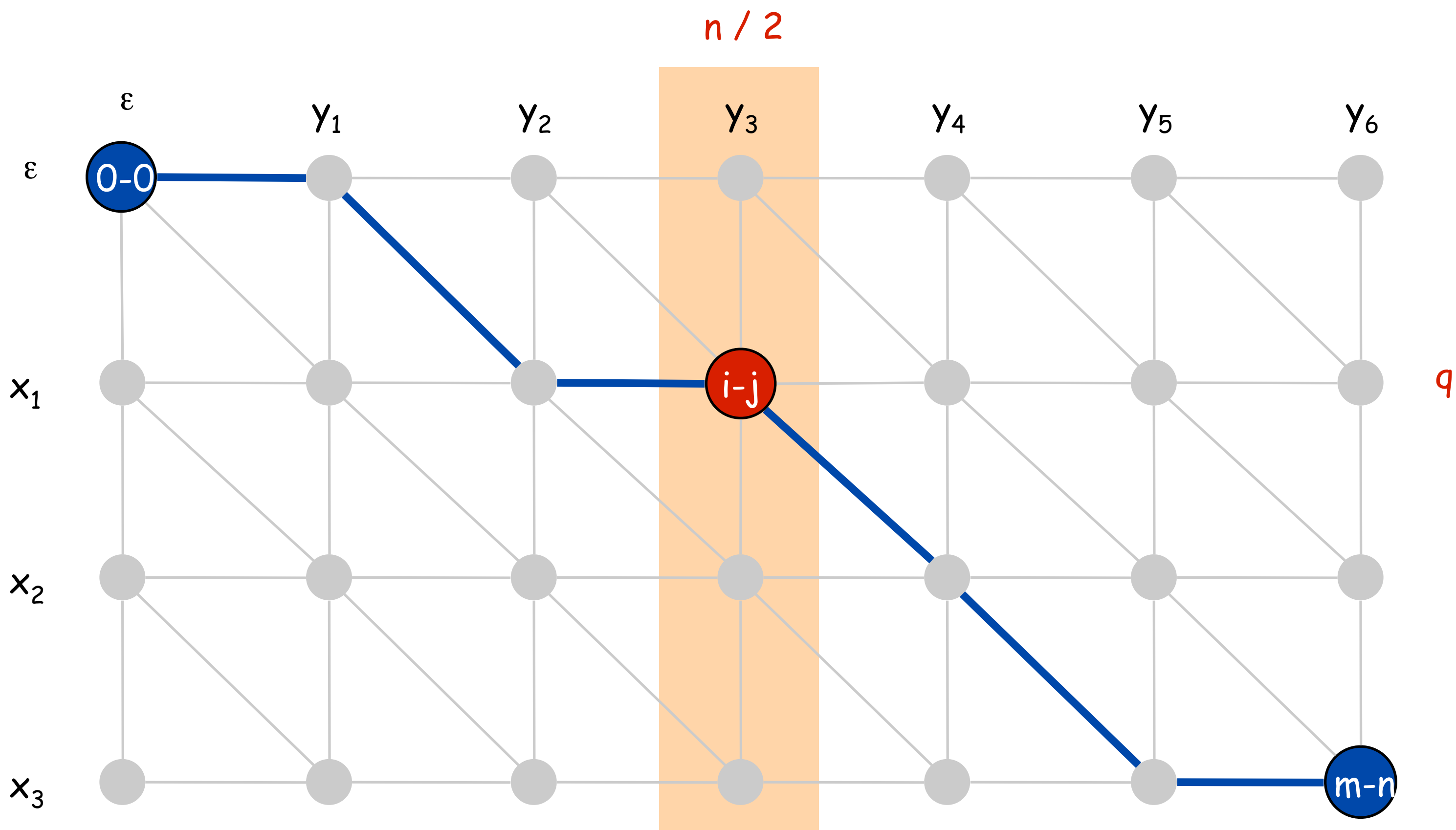
Sequence Alignment: Linear Space

Observation 1. The cost of the shortest path that uses (i, j) is $f(i, j) + g(i, j)$.



Sequence Alignment: Linear Space

Observation 2. let q be an index that minimizes f(q, n/2) + g(q, n/2). Then, the shortest path from (0, 0) to (m, n) uses (q, n/2).

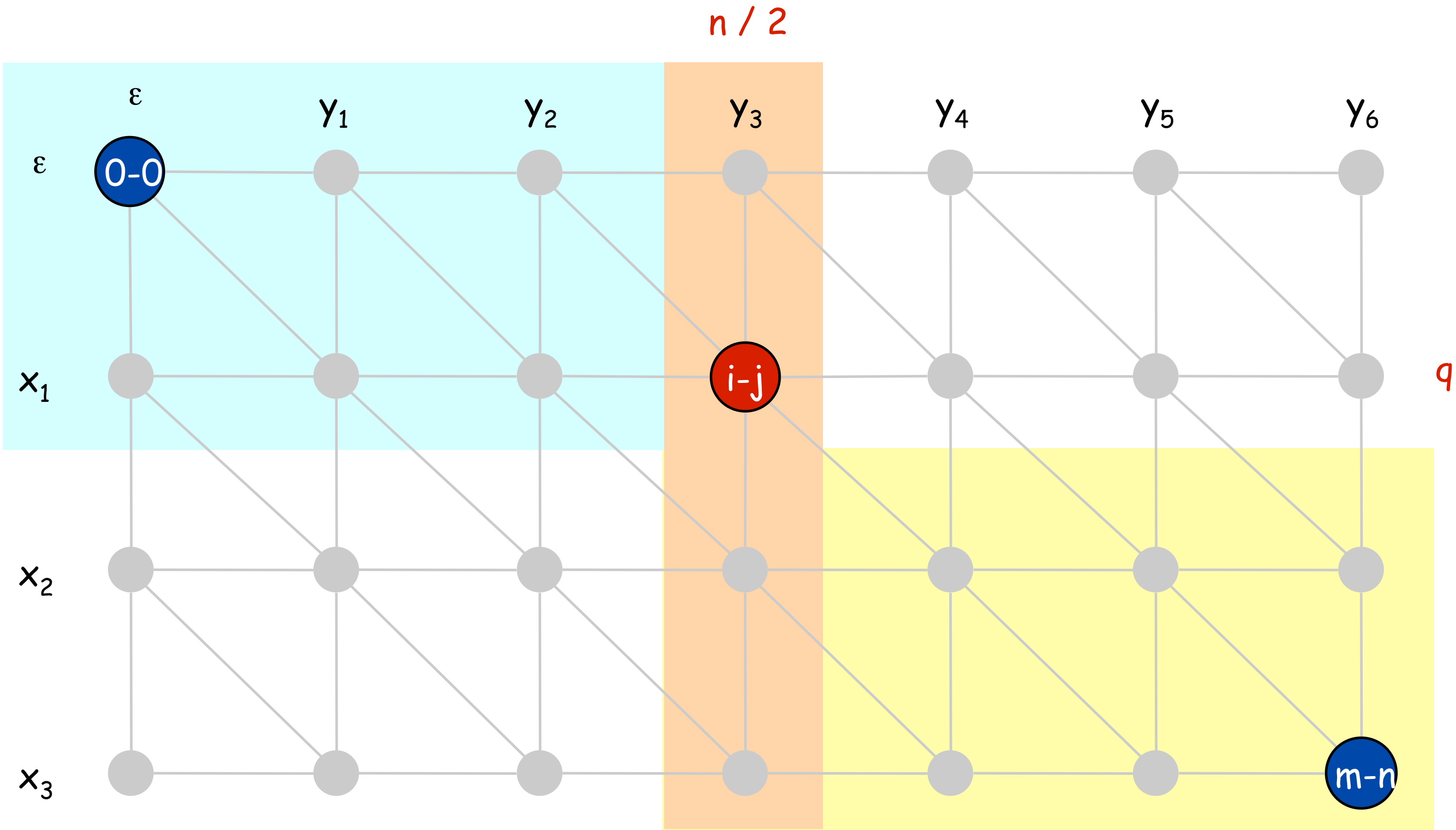


Sequence Alignment: Linear Space

Divide: find index q that minimizes $f(q, n/2) + g(q, n/2)$ using DP.

- Align x_q and $y_{n/2}$.

Conquer: recursively compute optimal alignment in each piece.



Sequence Alignment: Running Time Analysis Warmup

Theorem. Let T(m, n) = max running time of algorithm on strings of length at most m and n. T(m, n) = O(mn log n).

$$T(m,n) \leq 2T(m, n/2) + O(mn) \Rightarrow T(m,n) = O(mn \log n)$$

Remark. Analysis is not tight because two sub-problems are of size (q, n/2) and (m - q, n/2). In next slide, we save log n factor.

Sequence Alignment: Running Time Analysis

Theorem. Let $T(m, n)$ = max running time of algorithm on strings of length m and n. $T(m, n) = O(mn)$.

Pf. (by induction on n)

- $O(mn)$ time to compute $f(\cdot, n/2)$ and $g(\cdot, n/2)$ and find index q.
- $T(q, n/2) + T(m - q, n/2)$ time for two recursive calls.
- Choose constant c so that:

$$
\begin{aligned}
T(m,\ 2) &\le cm \\
T(2,\ n) &\le cn \\
T(m,\ n) &\le cmn + T(q,\ n/2) + T(m-q,\ n/2)
\end{aligned}
$$

- Base cases: m = 2 or n = 2.
- Inductive hypothesis: for $m' < m$ or $n' < n$, $T(m', n') \le 2cm'n'$.

$$
\begin{aligned}
T(m,n) &\le T(q,n/2) + T(m-q,n/2) + cmn \\
&\le 2cqn/2 + 2c(m-q)n/2 + cmn \\
&= cqn + cmn - cqn + cmn \\
&= 2cmn
\end{aligned}
$$