Dynamic Programming

- Sequence Alignment
- Longest Common Subsequence
- Longest Increasing Subsequence

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

6 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]

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

x_1	x_2	x_3	x_4	x_5		x_6
C	T	A	C	C	-	G
-	T	A	C	A	T	G
	y_1	y_2	y_3	y_4	y_5	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[i, 0] = iδ
   for j = 0 to n
      M[0, j] = 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: $m, n \le 10$.

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

Sequence Alignment in Linear Space

Sequence Alignment: Linear Space

Q. Can we avoid using quadratic space?

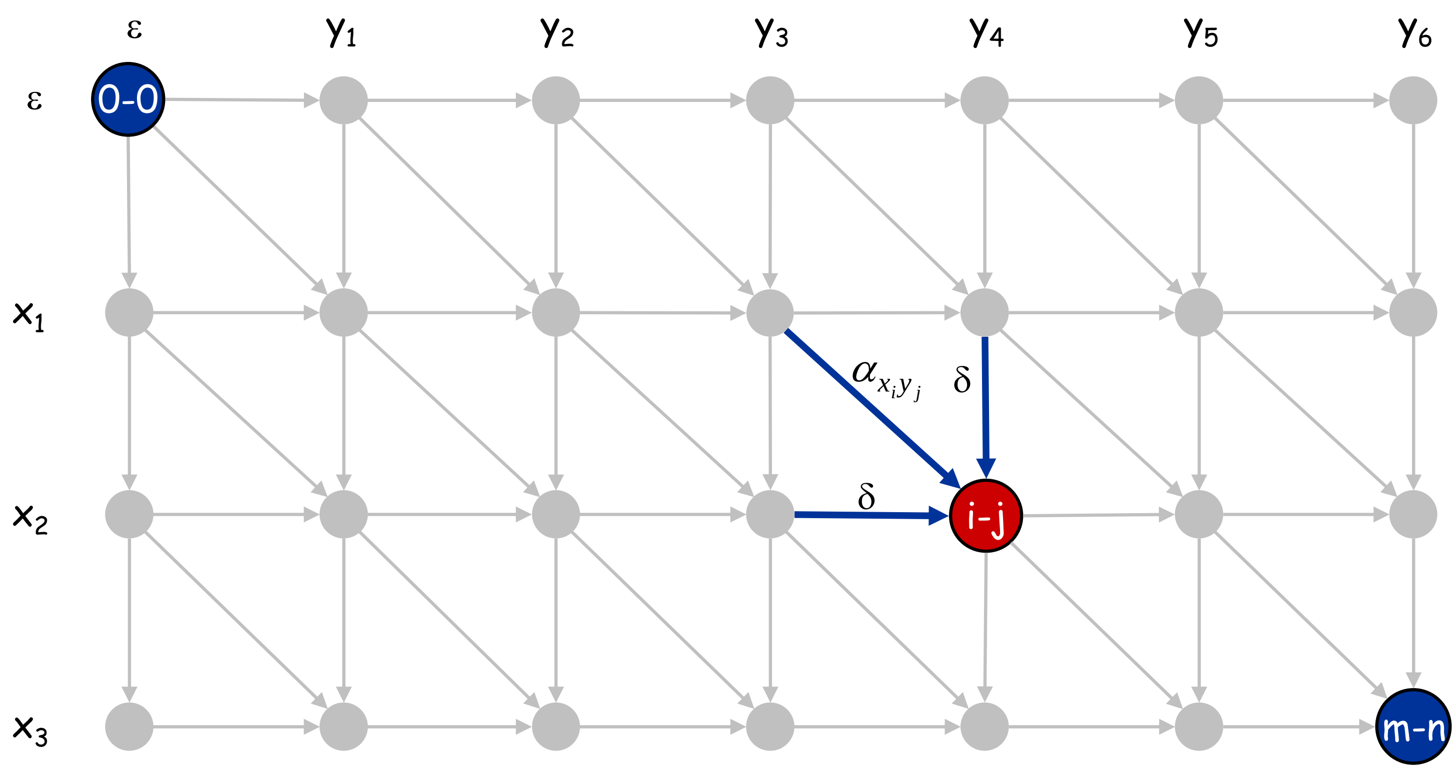
Easy. Optimal value in $O(m + n)$ space and $O(mn)$ time.
- Compute $OPT(i, \bullet)$ from $OPT(i-1, \bullet)$.
- No longer a simple way to recover alignment itself.

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.

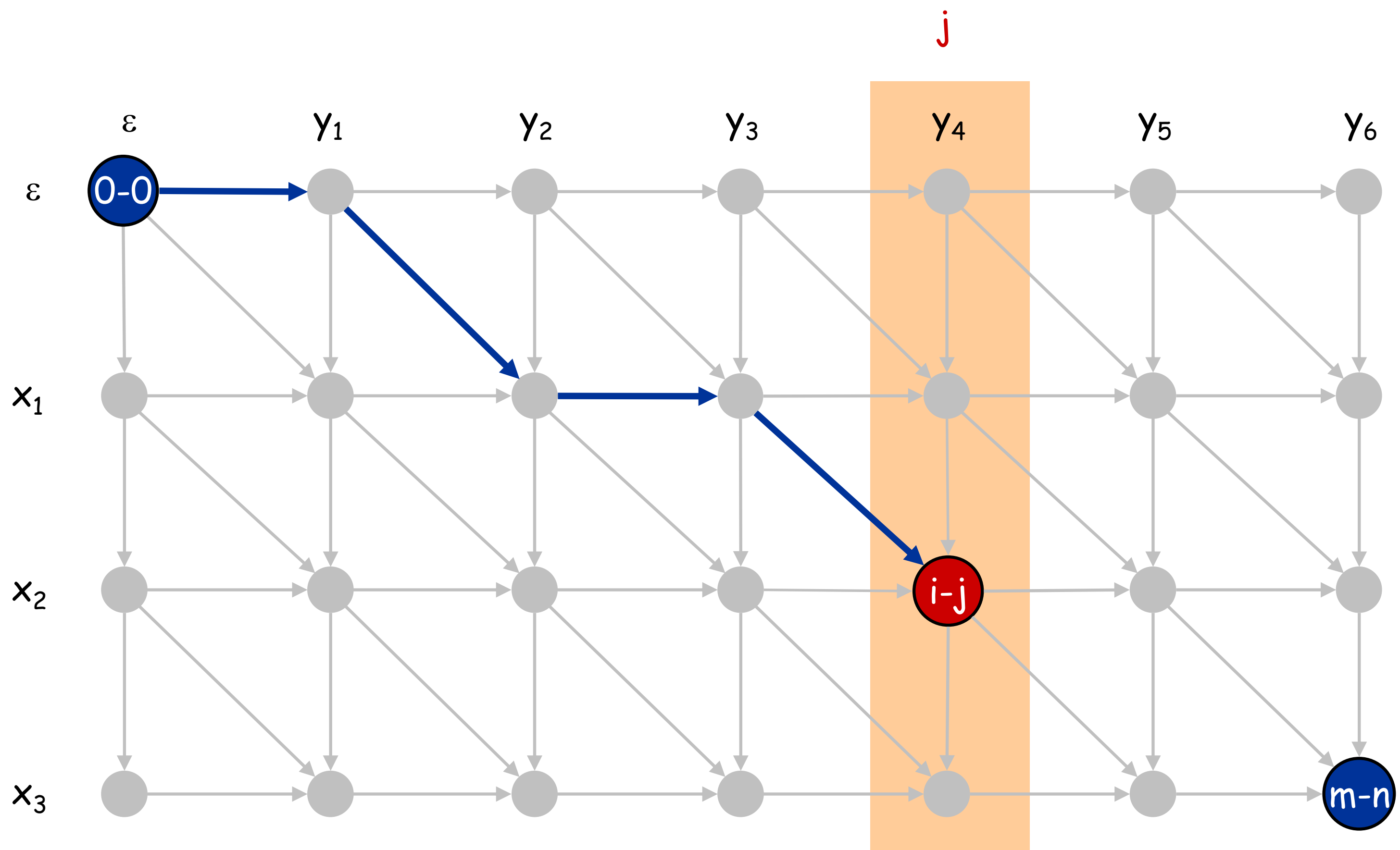
- Let f(i, j) be shortest path from (0,0) to (i, j).
- Observation: f(i, j) = OPT(i, j).



Sequence Alignment: Linear Space

Edit distance graph.

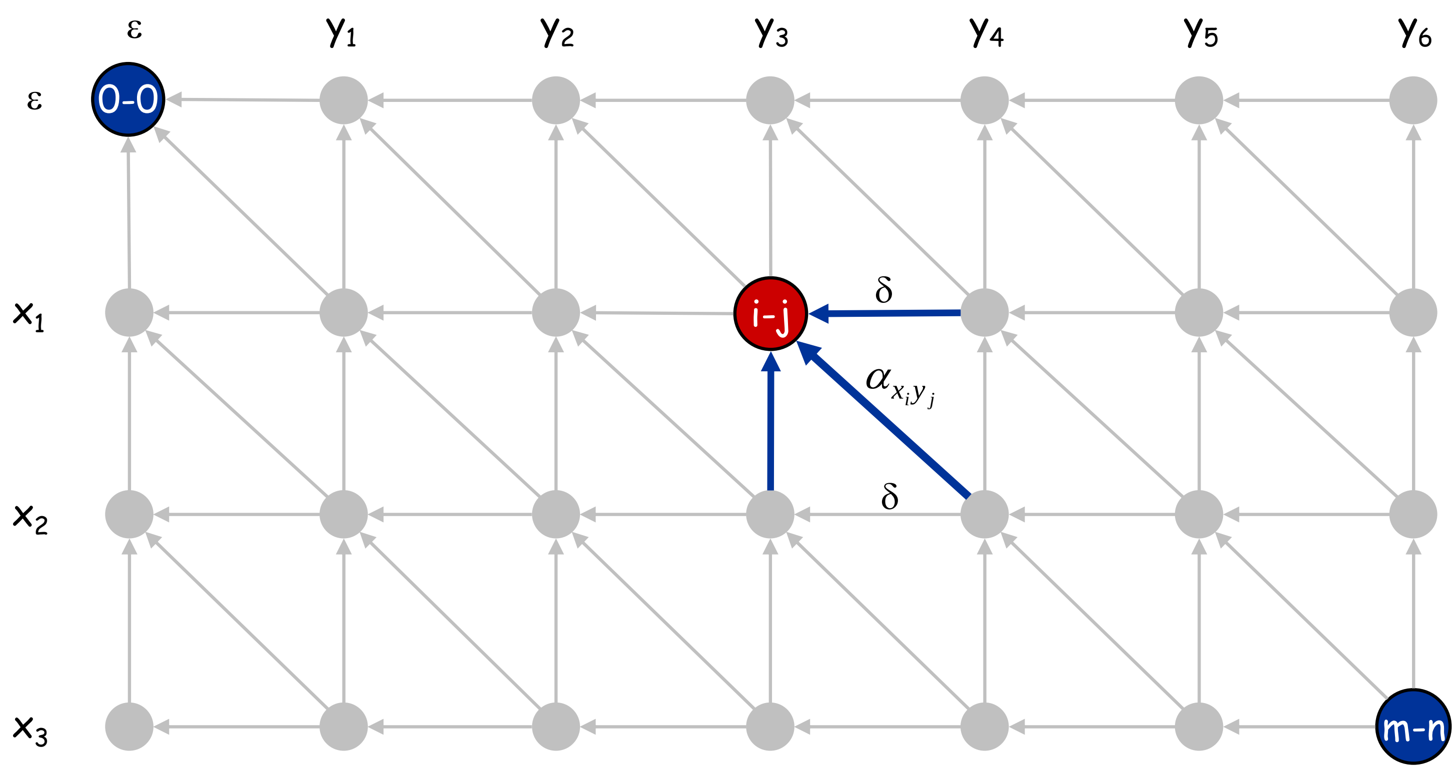
- Let f(i, j) be shortest path from (0,0) to (i, j).
- Can compute f (•, j) for any j in O(mn) time and O(m + n) space.



Sequence Alignment: Linear Space

Edit distance graph.

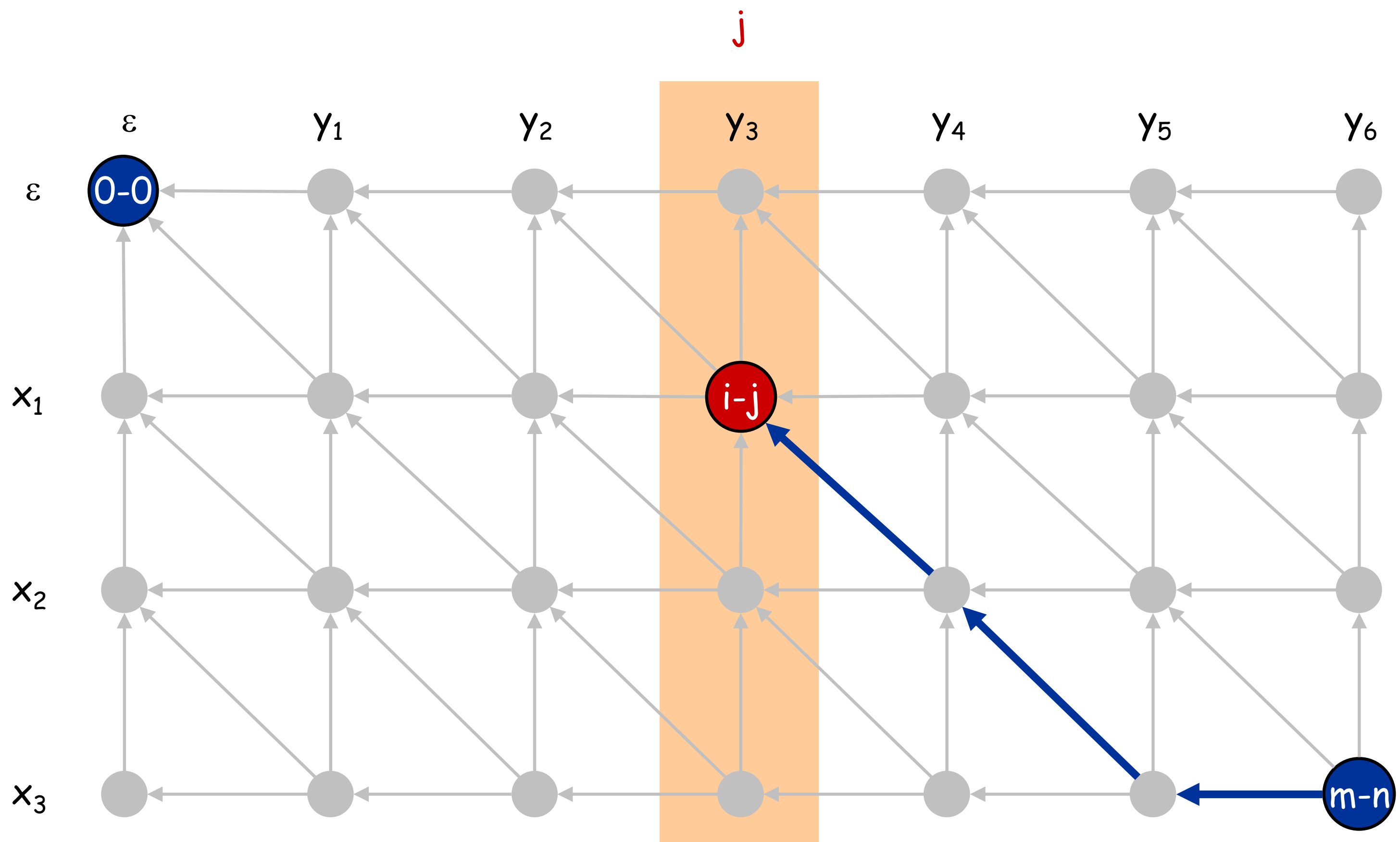
- Let g(i, j) be shortest path from (i, j) to (m, n).
- Can compute by reversing the edge orientations and inverting the roles of (0, 0) and (m, n)



Sequence Alignment: Linear Space

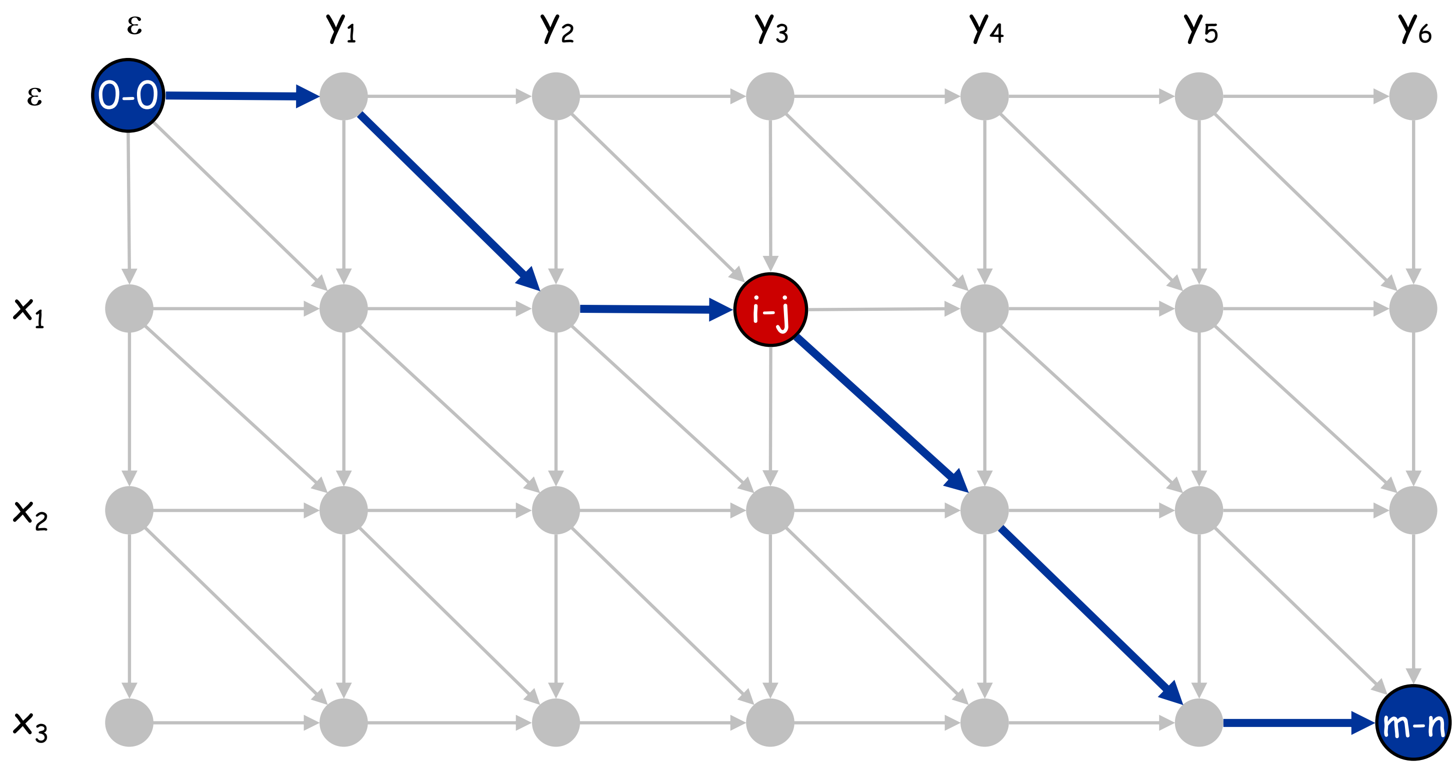
Edit distance graph.

- 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.



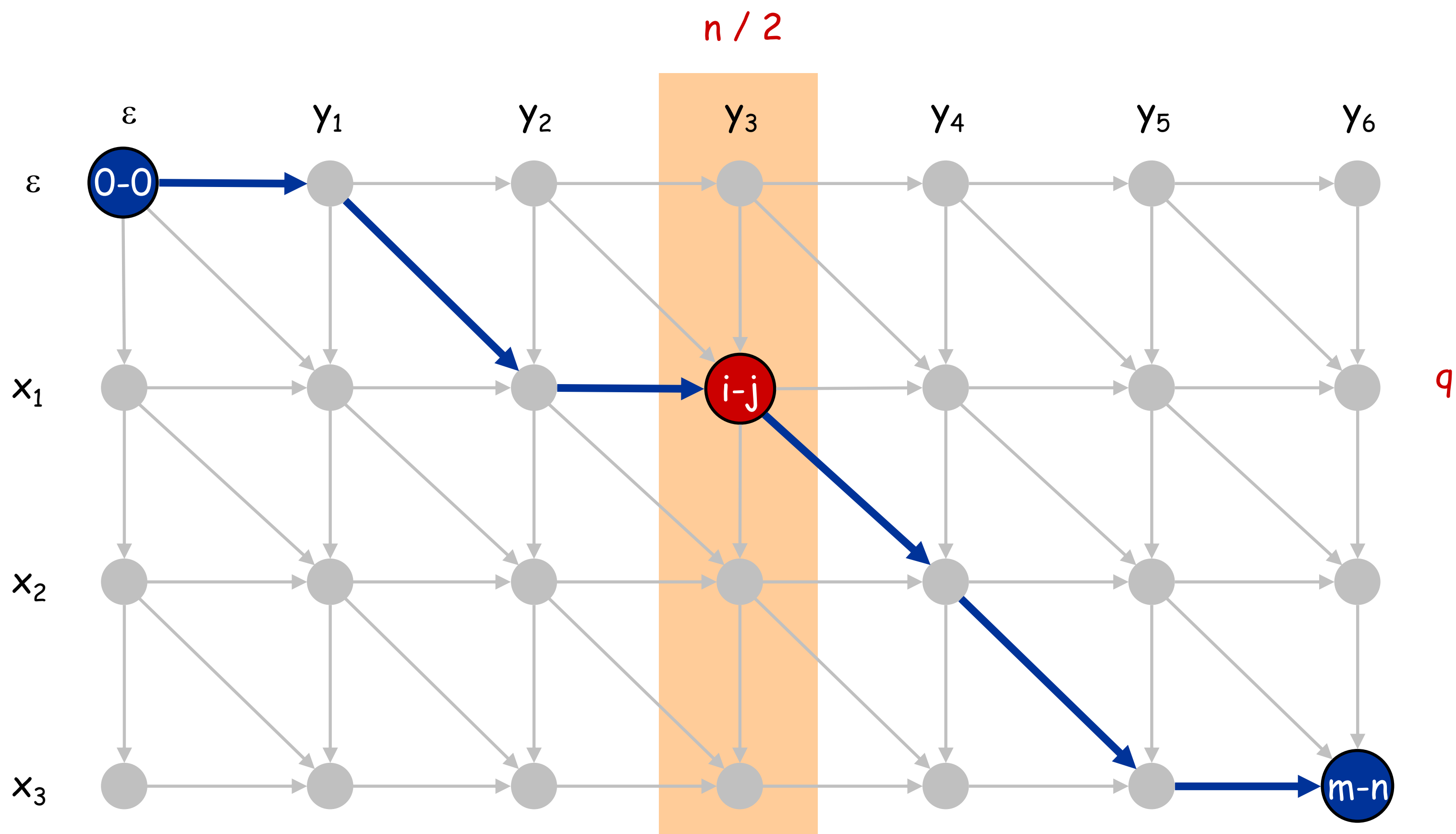
Sequence Alignment: Linear Space

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 $(q, n/2)$.

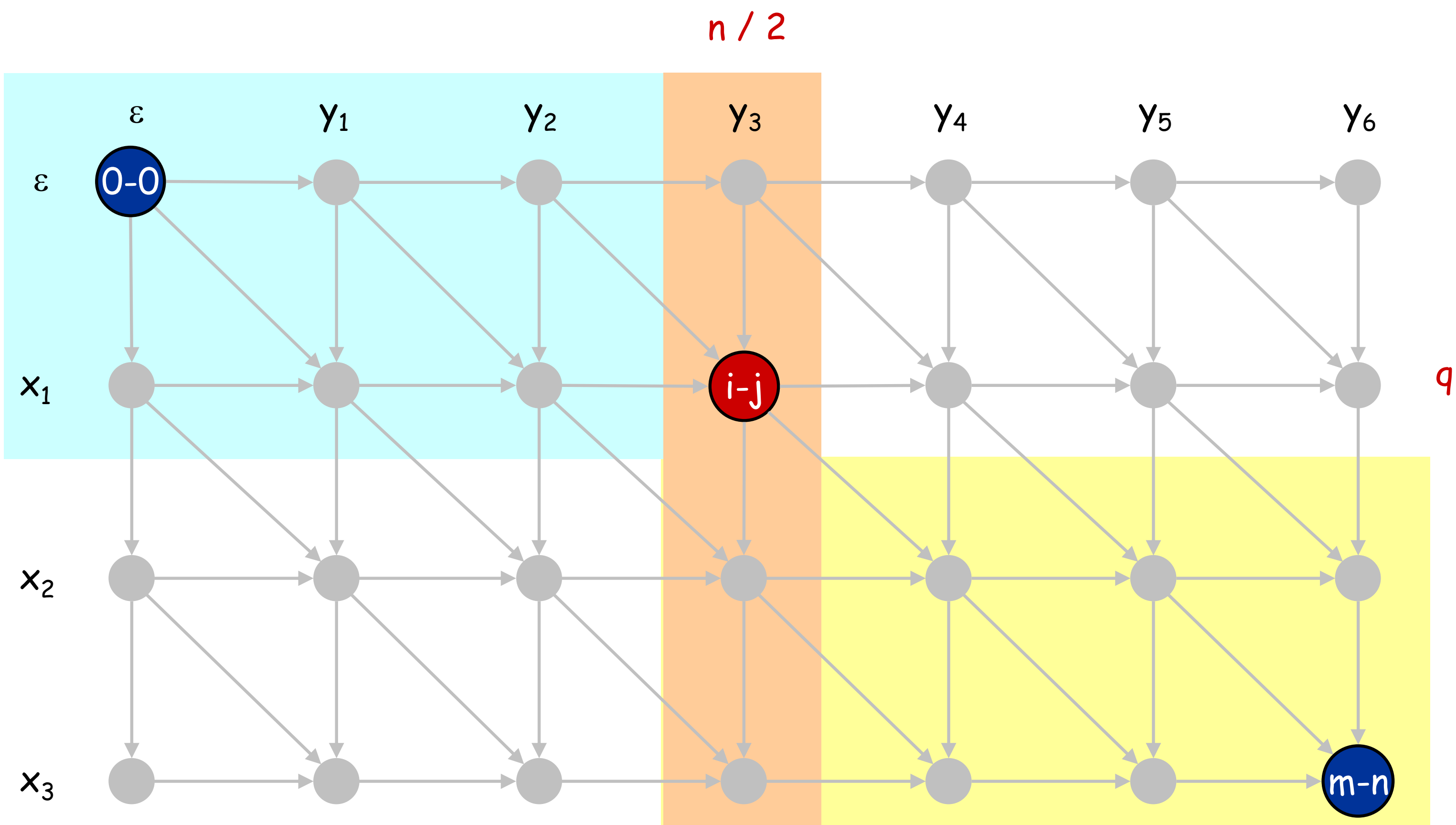


Sequence Alignment: Linear Space

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).

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

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(•, n/2) and g (•, 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: T(m, n) ≤ 2cmn.

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

Longest Common Subsequence (LCS)

Longest Common Subsequence (LCS)

LCS problem. Given two sequences

$$X = \langle x_1, x_2, \ldots, x_m \rangle$$

$$Y = \langle y_1, y_2, \ldots, y_n \rangle$$

find a longest common subsequence (LCS) of X and Y

Ex.

Subsequences of $X = \langle A, B, C, B, D, A, B \rangle$

- A subset of elements in the sequence taken in order

  $\langle A, B, D \rangle$, $\langle B, C, D, B \rangle$, etc.

LCS(X,Y).

$X = \langle A, B, C, B, D, A, B \rangle$ $\quad$ $X = \langle A, B, C, B, D, A, B \rangle$

$Y = \langle B, D, C, A, B, A \rangle$ $\quad$ $Y = \langle B, D, C, A, B, A \rangle$

$\langle B, C, B, A \rangle$ and $\langle B, D, A, B \rangle$ are longest common subsequences of X and Y (length = 4)

$\langle B, C, A \rangle$, however is not a LCS of X and Y

Notation

The i-th prefix. Given a sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$ we define the i-th prefix of X, for $i = 0, 1, 2, \ldots, m$ to be

$$X_i = \langle x_1, x_2, \ldots, x_i \rangle$$

$c[i, j]$ = the length of a LCS of the sequences $X_i = \langle x_1, x_2, \ldots, x_i \rangle$ and $Y_j = \langle y_1, y_2, \ldots, y_j \rangle$

Recursive Solution

Case 1: $x_i = y_j$

Ex. $X_i = \langle A, B, D, E \rangle$

$Y_j = \langle Z, B, E \rangle$

- Append $x_i = y_j$ to the LCS of X_{i-1} and Y_{j-1}
- Must find a LCS of X_{i-1} and Y_{j-1}

Case 2: $x_i \neq y_j$

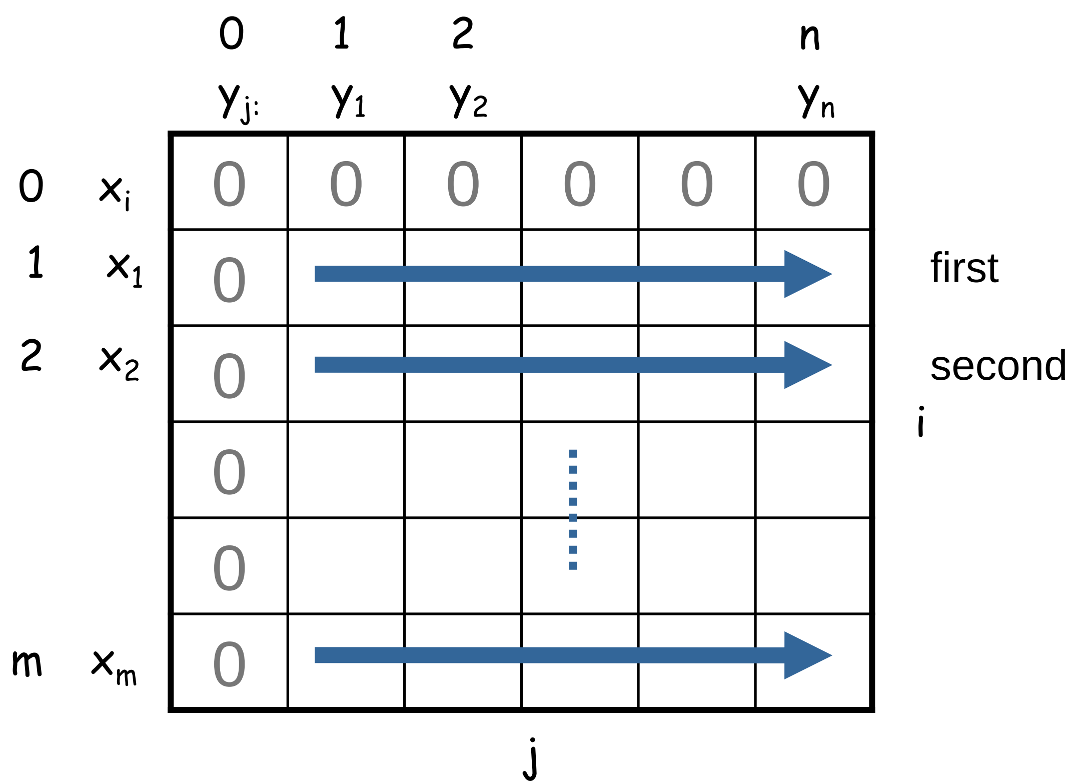
Ex. $X_i = \langle A, B, D, G \rangle$

$Y_j = \langle Z, B, D \rangle$

- Must solve two problems
  - find a LCS of X_{i-1} and Y_j: $X_{i-1} = \langle A, B, D \rangle$ and $Y_j = \langle Z, B, D \rangle$
  - find a LCS of X_i and Y_{j-1}: $X_i = \langle A, B, D, G \rangle$ and $Y_j = \langle Z, B \rangle$

Computing the length of the LCS

$$
c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{if } x_i \neq y_j \end{cases}
$$



Additional Information

$$c[i, j] = \begin{cases} 0 & \text{if } i,j = 0 \\ c[i-1, j-1] + 1 & \text{if } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{if } x_i \neq y_j \end{cases}$$

b & c:

		0 y_j:	1 A	2 C	3 D		n F
0	x_i	0	0	0	0	0	0
1	A	0					
2	B	0			c[i-1,j]		
3	C	0		↖ c[i,j-1]	↑		
		0					
m	D	0					

i

j

A matrix b[i, j]:

- For a subproblem [i, j] it tells us what choice was made to obtain the optimal value
- If $x_i = y_j$
  b[i, j] = "↖"
- Else, if $c[i - 1, j] \geq c[i, j-1]$
  b[i, j] = " ↑ "

  else

  b[i, j] = " ← "

LCS: Algorithm

```
LCS(m, n, x1x2...xm, y1y2...yn) {
   for i = 0 to m do
      c[i, 0] = 0,
   for j = 0 to n do
      c[0, j] = 0
   for i = 1 to m do
      for j = 1 to n do
        if xi=yj then
           c[i,j] = c[i-1, j-1]+1
           b[I,j]= ↖
        else if c[i-1,j] ≥ c[i,j-1] then
           c[i,j] = c[i-1, j]
           b[i,j]= ↑
        else
          c[i,j] = c[i, j-1]
          b[i,j]= ←
return c[m, n]
}
```

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

Example

$X = \langle A, B, C, B, D, A \rangle$
$Y = \langle B, D, C, A, B, A \rangle$

		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Constructing a LCS

Start at b[m, n] and follow the arrows

When we encounter a " ↖" in b[i, j] $\Rightarrow x_i = y_j$ is an element of the LCS

		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Improving the Code

If we only need the length of the LCS

- LCS-LENGTH works only on two rows of c at a time
  - The row being computed and the previous row
- We can reduce the asymptotic space requirements by storing only these two rows. So the space can be reduced to O(min(m,n)).

Reduction to Sequence Alignment

In the sequence alignment problem:

- See sequences as strings
- Set gap penalty δ to be 0
- Set α_{pq} to be -1 if p=q. otherwise set to be 0.

Longest Increasing Subsequence

Longest Increasing Subsequence (LIS)

LIS problem. Given a sequence of numbers

$$X = \langle x_1, x_2, \dots, x_n \rangle$$

find a longest increasing subsequence (LIS) of X

Ex. $X=\langle 7, 2, 5, 1, 13, 12, 19\rangle$

- $LIS(X)= \langle 2, 5, 13, 19\rangle$

Reduction to LCS

LIS can be reduced to LCS as follows:

- $X = \langle x_1, x_2, \ldots, x_n \rangle$
- Y = sort of X

Then LCS(X,Y)=LIS(X)

Ex. $X=\langle 7, 2, 5, 1, 13, 12, 19 \rangle$

- $X=\langle 7, 2, 5, 1, 13, 12, 19 \rangle$
- $Y= \langle 1, 2, 5, 7, 12, 13, 19 \rangle$

Then LCS(X, Y)= $\langle 2, 5, 13, 19 \rangle$=LIS(X)

Running time: $O(n^2)$

Remark. There is an efficient DP running in O(n log n) time.

References

References

- Sections 6.6, and 6.7 of the text book “algorithm design” by Jon Kleinberg and Eva Tardos
- Section 15.4 of the text book “introduction to algorithms” by CLRS, 3rd edition.
- The original slides were prepared by Kevin Wayne. The slides are distributed by Pearson Addison-Wesley.
- The LCS part is from the slides prepared by George Bebbis.