



南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# Algorithm Design and Analysis (H)

## CS216

Instructor: Shan CHEN (陈杉)

[chens3@sustech.edu.cn](mailto:chens3@sustech.edu.cn)

(slides edited from Prof. Shiqi Yu)



# Dynamic Programming

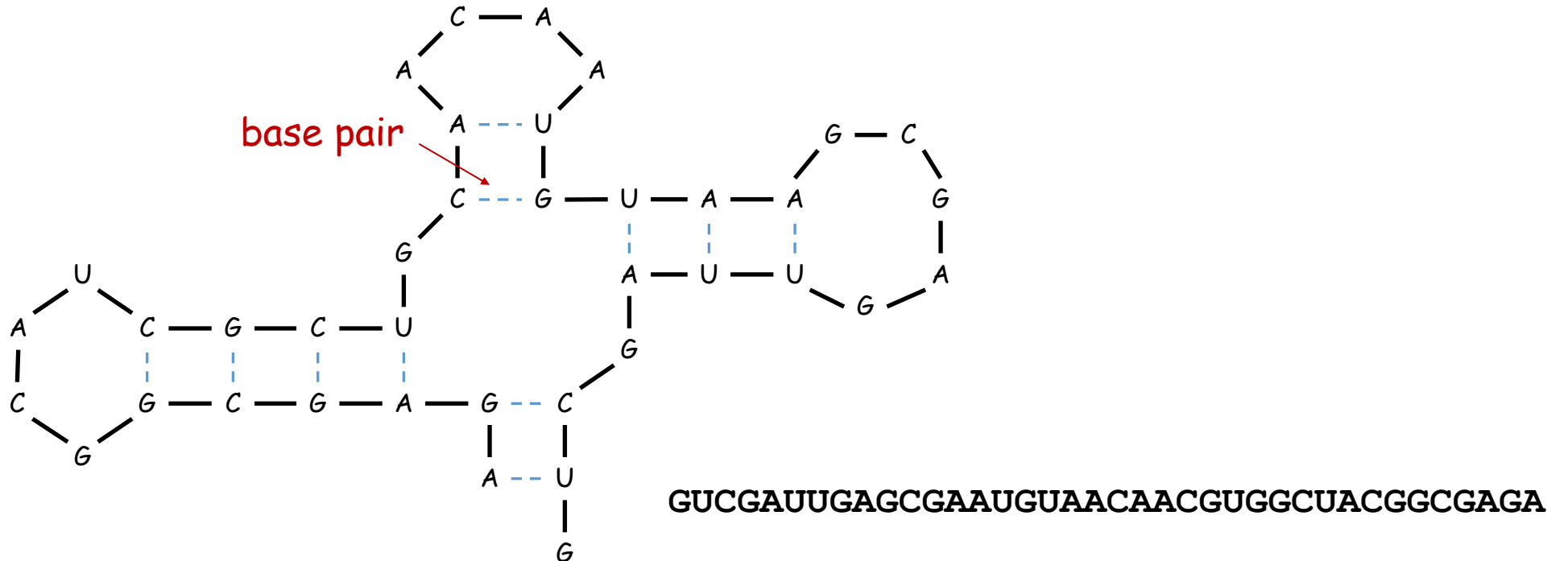


# 4. RNA Secondary Structure



# RNA Secondary Structure

- **RNA.** String  $B = b_1b_2\dots b_n$  over alphabet  $\{ A, C, G, U \}$ .
- **Secondary structure.** RNA is single-stranded so it tends to loop back and form **base pairs** with itself. This structure is essential for understanding behavior of molecule.

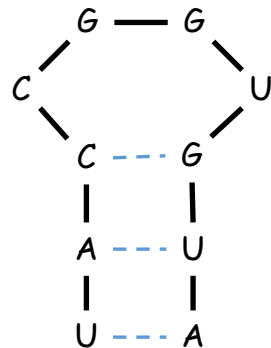




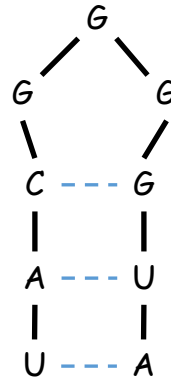
# RNA Secondary Structure

- **Secondary structure.** A set of pairs  $S = \{ (b_i, b_j) \}$  that satisfy:
  - **[Watson-Crick]**  $S$  is a matching and each pair in  $S$  is a Watson-Crick complement: A-U, U-A, C-G, or G-C.
  - **[No sharp turns]** The ends of each pair are separated by at least 4 intervening bases. If  $(b_i, b_j) \in S$ , then  $i < j - 4$ .
  - **[Non-crossing]** If  $(b_i, b_j)$  and  $(b_k, b_l)$  are two pairs in  $S$ , we cannot have  $i < k < j < l$ .

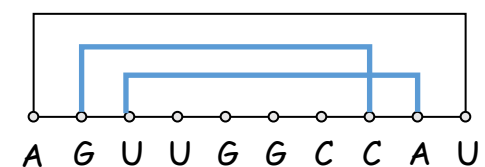
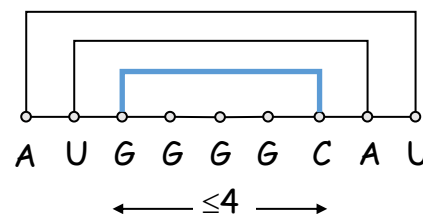
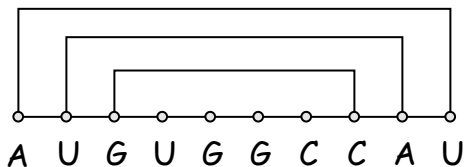
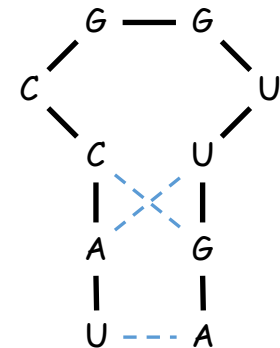
Watson-Crick  
complements



sharp turn



crossing





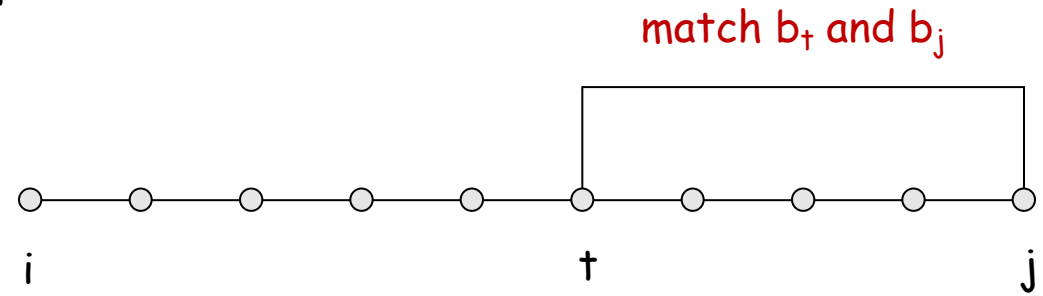
# RNA Secondary Structure

- **Secondary structure.** A set of pairs  $S = \{ (b_i, b_j) \}$  that satisfy:
  - **[Watson-Crick]**  $S$  is a matching and each pair in  $S$  is a Watson-Crick complement: A-U, U-A, C-G, or G-C.
  - **[No sharp turns]** The ends of each pair are separated by at least 4 intervening bases. If  $(b_i, b_j) \in S$ , then  $i < j - 4$ .
  - **[Non-crossing]** If  $(b_i, b_j)$  and  $(b_k, b_l)$  are two pairs in  $S$ , we cannot have  $i < k < j < l$ .
- **Free energy hypothesis.** RNA molecule will form the secondary structure with the minimum total **free energy**.
  - ← approximate by number of base pairs  
more base pairs  $\Rightarrow$  lower free energy
- **Goal.** Given an RNA molecule  $B = b_1b_2\dots b_n$ , find a secondary structure  $S$  that maximizes the number of base pairs.



# RNA Secondary Structure: Subproblems

- **First attempt.**  $\text{OPT}(j)$  = maximum number of base pairs in a secondary structure of the substring  $b_1b_2\dots b_j$ .
- **Goal.**  $\text{OPT}(n)$

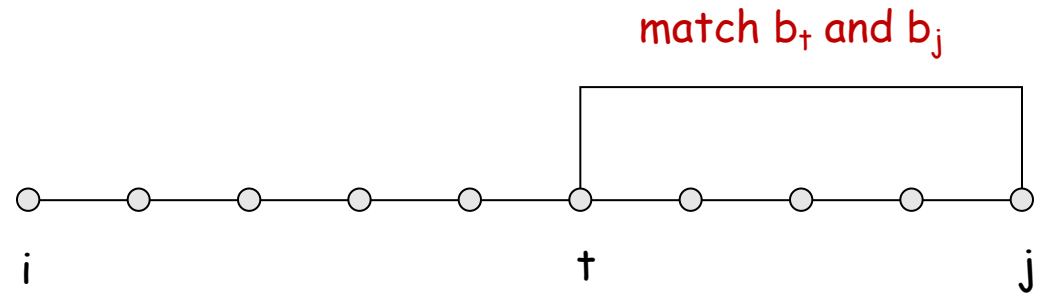


- **Difficulty.**  $\text{OPT}(j)$  results in two sub-problems (but one of wrong form).
  - Find secondary structure in:  $b_1b_2\dots b_{t-1}$ . ←  $\text{OPT}(t - 1)$
  - Find secondary structure in:  $b_{t+1}b_{t+2}\dots b_{j-1}$ . ← need more subproblems



# Dynamic Programming over Intervals

- **Def.**  $\text{OPT}(i, j)$  = maximum number of base pairs in a secondary structure of the substring  $b_i b_{i+1} \dots b_j$ .
- **Goal.**  $\text{OPT}(1, n)$



- **To compute  $\text{OPT}(i, j)$ :**
  - Case 1.  $i \geq j - 4$ :  $\text{OPT}(i, j) = 0$  by **no-sharp-turns** condition.
  - Case 2. Base  $b_j$  is not involved in a pair:  $\text{OPT}(i, j) = \text{OPT}(i, j - 1)$
  - Case 3. Base  $b_j$  pairs with  $b_t$  for some  $i \leq t < j - 4$ .
    - ✓ non-crossing constraint decouples resulting sub-problems
    - ✓  $\text{OPT}(i, j) = 1 + \max_t \{ \text{OPT}(i, t - 1) + \text{OPT}(t + 1, j - 1) \}$

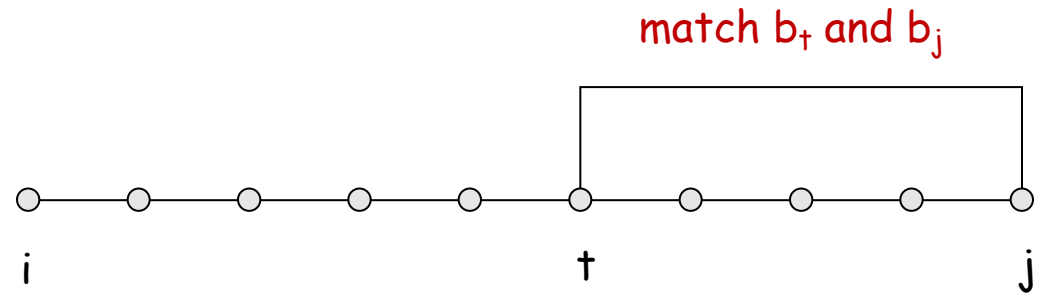
take max over  $t$  such that  $i \leq t < j - 4$  and  $b_t$  and  $b_j$  are Watson-Crick complements





# Dynamic Programming over Intervals

- **Def.**  $OPT(i, j)$  = maximum number of base pairs in a secondary structure of the substring  $b_i b_{i+1} \dots b_j$ .
- **Goal.**  $OPT(1, n)$



- **To compute  $OPT(i, j)$ :**

← compute in which order?

$$OPT(i, j) = \begin{cases} 0, & \text{if } i \geq j - 4 \\ OPT(i, j - 1), & b_j \text{ cannot be paired} \\ 1 + \max_t \{OPT(i, t - 1) + OPT(t + 1, j - 1)\}, & b_j \text{ can be paired} \end{cases}$$

← take max over  $t$  such that  $i \leq t < j - 4$  and  $b_t$  and  $b_j$  are Watson-Crick complements



# Dynamic Programming over Intervals

- **Dynamic programming algorithm (bottom-up).**

```
RNA( $b_1, \dots, b_n$ ) {  
  for  $k = 5, 6, \dots, n - 1$  ← shortest intervals first  
    for  $i = 1, 2, \dots, n - k$   
       $j = i + k$   
      Compute  $M[i, j]$  using formula  
  
  return  $M[1, n]$   
}
```

all needed  $M[ , ]$  values already computed

$$\text{OPT}(i, j) = \begin{cases} 0, & \text{if } i \geq j - 4 \\ \text{OPT}(i, j - 1), & b_j \text{ cannot be paired} \\ 1 + \max_t \{ \text{OPT}(i, t - 1) + \text{OPT}(t + 1, j - 1) \}, & b_j \text{ can be paired} \end{cases}$$

- **Running time.**  $O(n^3)$       **Space.**  $O(n^2)$



# RNA-Secondary-Structure Algorithm: Demo

RNA sequence *ACCGGUAGU*

4	0	0	0	
3	0	0		
2	0			
$i = 1$				

$j = 6 \quad 7 \quad 8 \quad 9$

**Initial values**

4	0	0	0	0
3	0	0	1	
2	0	0		
$i = 1$	1			

$j = 6 \quad 7 \quad 8 \quad 9$

**Filling in the values  
for  $k = 5$**

4	0	0	0	0
3	0	0	1	1
2	0	0	1	
$i = 1$	1	1		

$j = 6 \quad 7 \quad 8 \quad 9$

**Filling in the values  
for  $k = 6$**

4	0	0	0	0
3	0	0	1	1
2	0	0	1	1
$i = 1$	1	1	1	

$j = 6 \quad 7 \quad 8 \quad 9$

**Filling in the values  
for  $k = 7$**

4	0	0	0	0
3	0	0	1	1
2	0	0	1	1
$i = 1$	1	1	1	2

$j = 6 \quad 7 \quad 8 \quad 9$

**Filling in the values  
for  $k = 8$**



# 5. Sequence Alignment



# Spell Correction

Hi,

This is a test for the typo/grammar checker. Can I get your responses?

陈杉

Shan Chen

Computer Science and Engineering

Southern University of Science and Technology



Correct the spelling error

**response**

Ignore



Rewrite sentence



Scientists discovered a  
substance that could prove  
that there is life on Venus.



## • SPELLING

~~Scientists~~ → **Scientists**

The word **Scientists** is not in our dictionary. If you're sure this spelling is correct, you can add it to your personal dictionary to prevent future alerts.

Add to dictionary





# String Similarity

- **Q.** How similar are two strings?
- **Ex.** **ocurrance** and **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

0 mismatches, 3 gaps

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

1 mismatch, 1 gap



# Edit Distance

- **Edit distance.** [Levenshtein 1966, Needleman-Wunsch 1970]
  - Gap penalty  $\delta$ ; mismatch penalty  $\alpha_{pq}$ . (Typically,  $\alpha_{pp} = 0$ .)
  - Cost = sum of gap and mismatch penalties.
  - Edit distance is the **minimum cost**.

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

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

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

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

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

$$2\delta + \alpha_{CA}$$

- **Applications.** Bioinformatics, spell correction, machine translation, speech recognition, information extraction, ...



# Sequence Alignment

- **Goal.** Given two strings  $x_1x_2 \dots x_m$  and  $y_1y_2 \dots y_n$  find a **min-cost** alignment.
- **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 there are no **crossings**.

$x_i-y_j$  and  $x_{i'}-y_{j'}$  cross if  $i < i'$  but  $j > j'$

- **Ex.** CTACCG vs. TACATG (gap penalty  $\delta = 2$ ; mismatch penalty  $\alpha_{pq} = 1$ )
- **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.**  $\text{OPT}(i, j)$  = min cost of aligning prefix strings  $x_1x_2 \dots x_i$  and  $y_1y_2 \dots y_j$
- **Goal.**  $\text{OPT}(m, n)$
- **To compute  $\text{OPT}(i, j)$ :**
  - **Case 1:** OPT matches  $x_i$ - $y_j$ .
    - ✓ Pay mismatch for  $x_i$ - $y_j$  + min cost of aligning  $x_1x_2 \dots x_{i-1}$  and  $y_1y_2 \dots y_{j-1}$
  - **Case 2a:** OPT leaves  $x_i$  unmatched.
    - ✓ Pay gap for  $x_i$  + min cost of aligning  $x_1x_2 \dots x_{i-1}$  and  $y_1y_2 \dots y_j$
  - **Case 2b:** OPT leaves  $y_j$  unmatched.
    - ✓ Pay gap for  $y_j$  + min cost of aligning  $x_1x_2 \dots x_i$  and  $y_1y_2 \dots y_{j-1}$



# Sequence Alignment: Problem Structure

- **Def.**  $\text{OPT}(i, j)$  = min cost of aligning prefix strings  $x_1x_2 \dots x_i$  and  $y_1y_2 \dots y_j$ .
- **Goal.**  $\text{OPT}(m, n)$
- **Bellman equation.**

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



# Sequence Alignment: Algorithm

- **Dynamic programming algorithm (bottom-up).**

```
Sequence-Alignment( $m, n, x_1x_2\ldots x_m, y_1y_2\ldots y_n, \delta, \alpha$ ) {  
  for  $i = 0$  to  $m$   
     $M[i, 0] = i\delta$   
  for  $j = 0$  to  $n$   
     $M[0, j] = j\delta$   
  
  for  $i = 1$  to  $m$   
    for  $j = 1$  to  $n$   
       $M[i, j] = \min(\alpha[x_i, y_j] + M[i-1, j-1],$   
                     $\delta + M[i-1, j],$   
                     $\delta + M[i, j-1])$   
  
  return  $M[m, n]$   
}
```

all needed  $M[ , ]$  values already computed

- **Running time and space.**  $O(mn)$   $\leftarrow$   $m, n \leq 10$  for English words  
 $m, n \approx 10^5$  for biology applications



# Sequence Alignment: Impossibility Result

- **Theorem.** [Backurs–Indyk 2015] If can compute edit distance of two strings of length  $n$  in  $O(n^{2-\epsilon})$  time for some constant  $\epsilon > 0$ , then can solve SAT with  $n$  variables and  $m$  clauses in  $\text{poly}(m)2^{(1-\delta)n}$  time for some constant  $\delta > 0$ .

Edit Distance Cannot Be Computed  
in Strongly Subquadratic Time  
(unless SETH is false)\*

SETH: strong exponential time hypothesis

Arturs Backurs<sup>†</sup>  
MIT

Piotr Indyk<sup>‡</sup>  
MIT

- **Takeaway.** It is very difficult (if not impossible) to improve the  $O(mn)$  running time of computing the edit distance of two strings.



# Sequence Alignment in Linear Space

- **Theorem.** [Hirschberg] There exists an algorithm to find an optimal alignment in  $O(mn)$  time and  $O(m + n)$  space.
  - Clever combination of divide-and-conquer and dynamic programming.
  - Inspired by idea of Savitch from complexity theory.

Programming  
Techniques

G. Manacher  
Editor

---

## A Linear Space Algorithm for Computing Maximal Common Subsequences

D.S. Hirschberg  
Princeton University

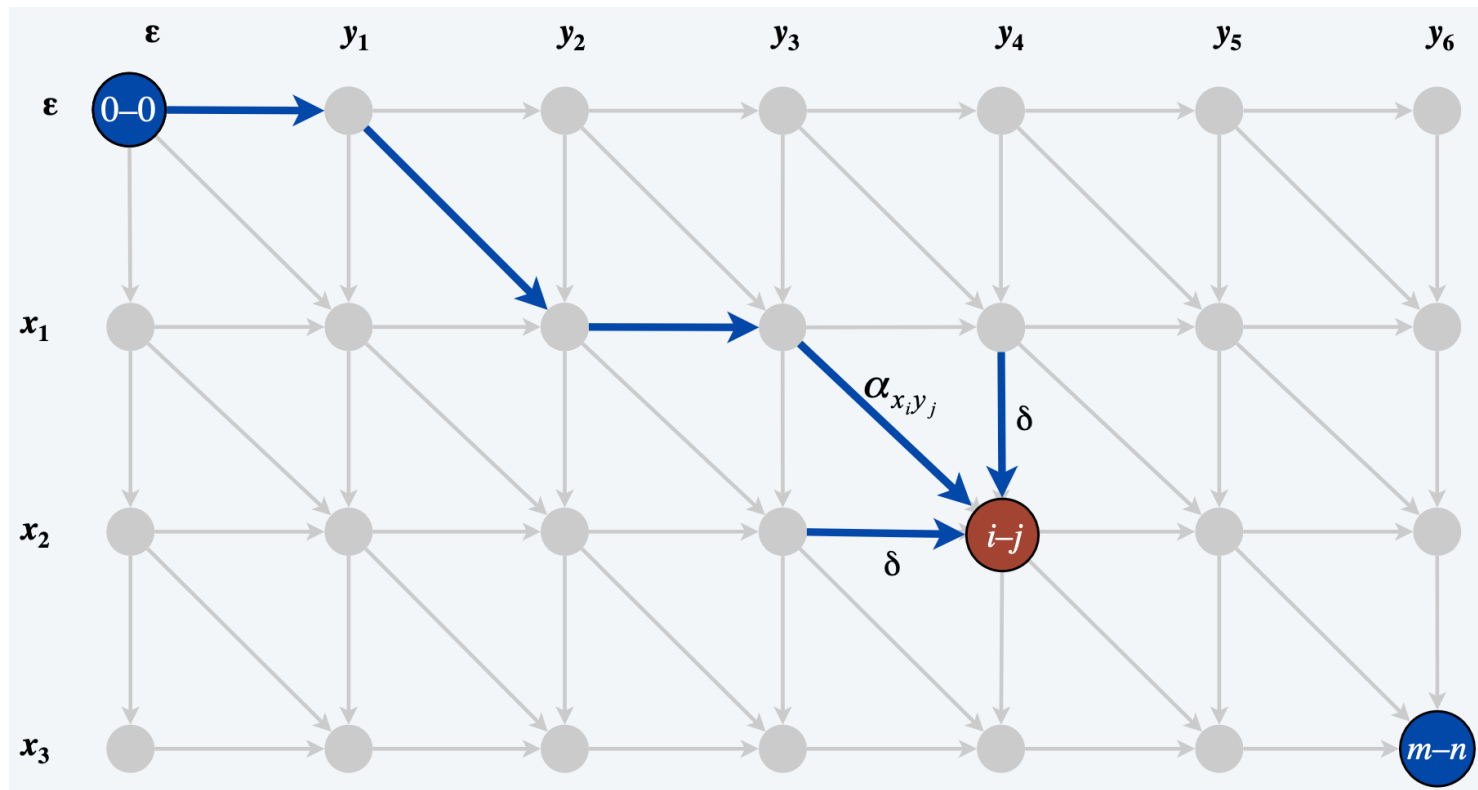




# Sequence Alignment: Hirschberg's Algorithm

- **Edit distance graph.**

- Let  $f(i, j)$  denote length of shortest path from  $(0, 0)$  to  $(i, j)$ .
- **Lemma:**  $f(i, j) = OPT(i, j)$  for all  $i$  and  $j$ .





# Sequence Alignment: Hirschberg's Algorithm

- **Edit distance graph.**

- Let  $f(i, j)$  denote length of shortest path from  $(0, 0)$  to  $(i, j)$ .
- **Lemma:**  $f(i, j) = OPT(i, j)$  for all  $i$  and  $j$ .

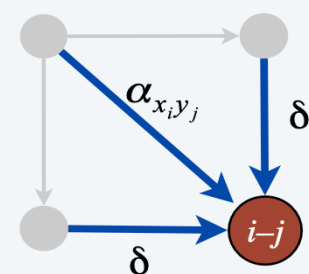
- Pf of Lemma. **(by strong induction on  $i + j$ )**

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

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

inductive hypothesis

Bellman equation

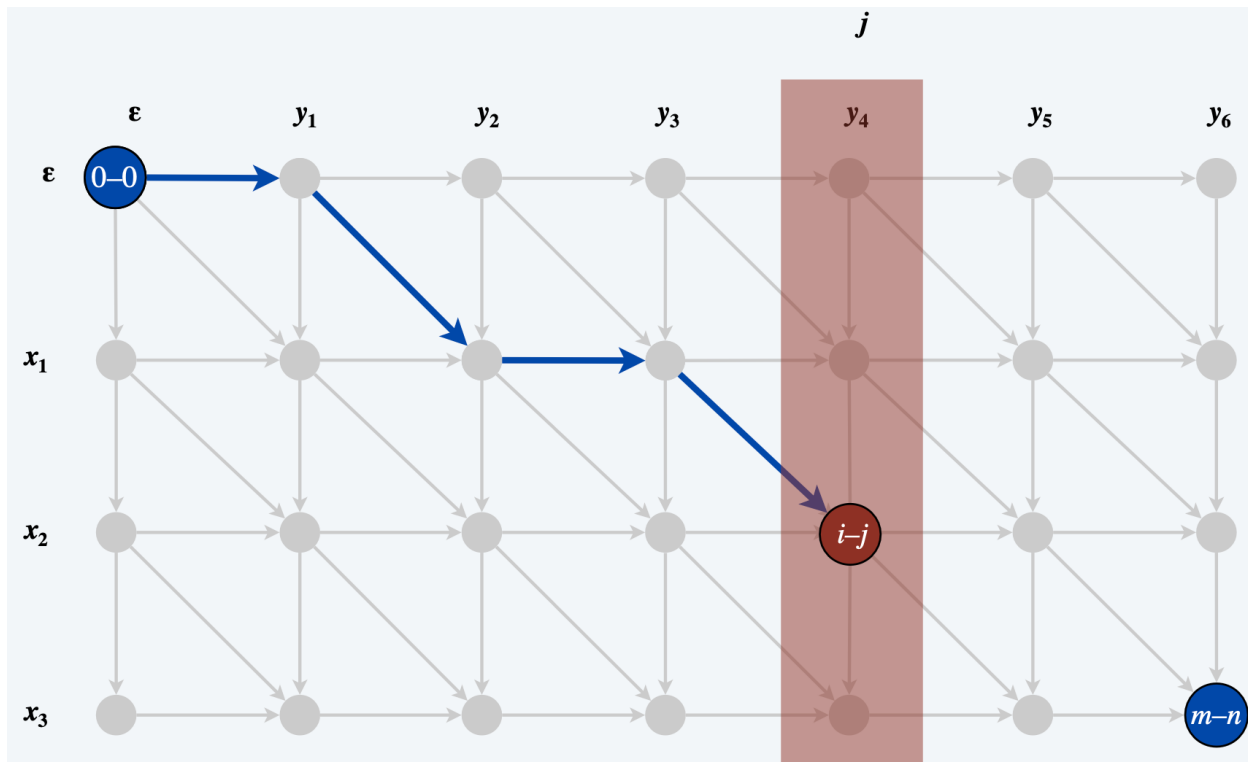




# Sequence Alignment: Hirschberg's Algorithm

- **Edit distance graph.**

- Let  $f(i, j)$  denote length of shortest path from  $(0, 0)$  to  $(i, j)$ .
- **Lemma:**  $f(i, j) = OPT(i, j)$  for all  $i$  and  $j$ .
- Can compute  $f(\cdot, j)$  for any specific  $j$  in  $O(mn)$  time and  $O(m)$  space.



$O(m)$  scrolling array

$f(\cdot, j)$  depends only on  $f(\cdot, j - 1)$

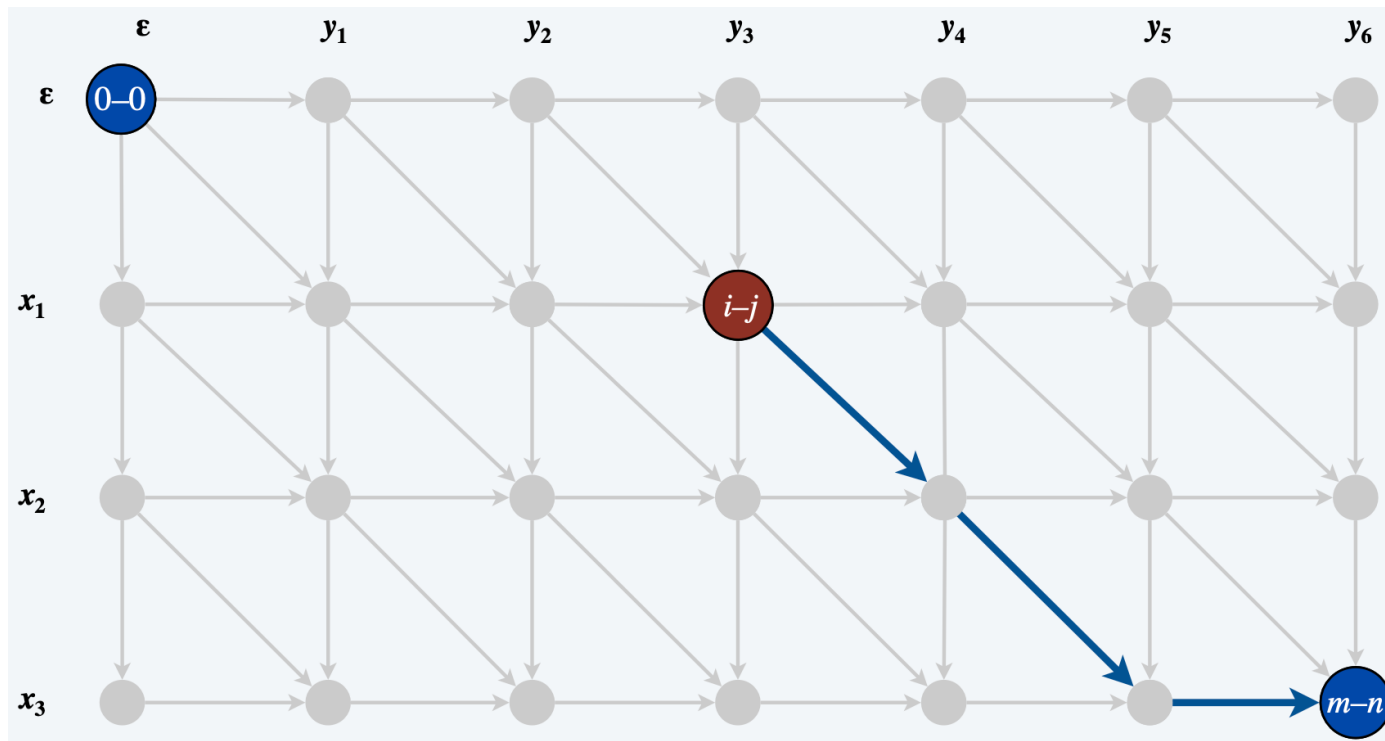




# Sequence Alignment: Hirschberg's Algorithm

- **Edit distance graph.**

➤ Let  $g(i, j)$  denote length of shortest path from  $(i, j)$  to  $(m, n)$ .

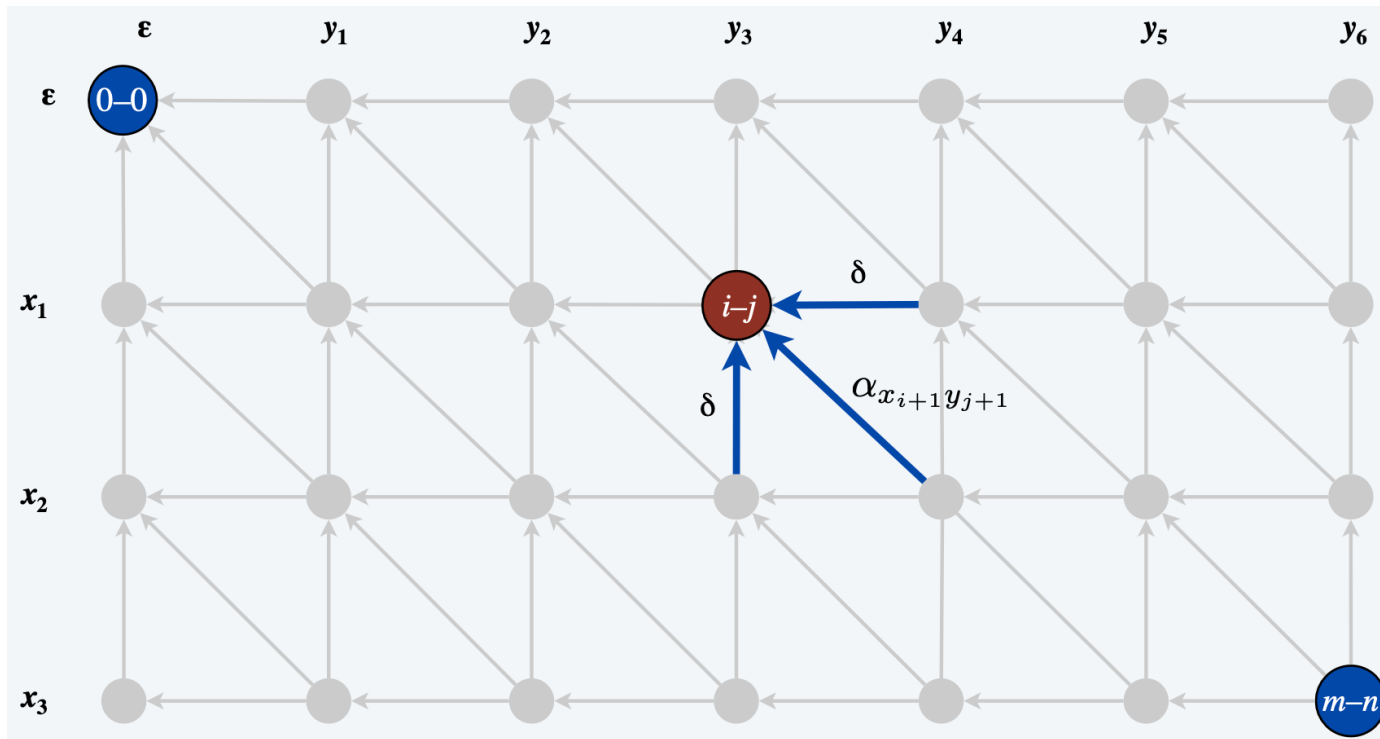




# Sequence Alignment: Hirschberg's Algorithm

- **Edit distance graph.**

- Let  $g(i, j)$  denote length of shortest path from  $(i, j)$  to  $(m, n)$ .
- Can compute  $g(i, j)$  by reversing edges and the roles of  $(0, 0)$  and  $(m, n)$ .

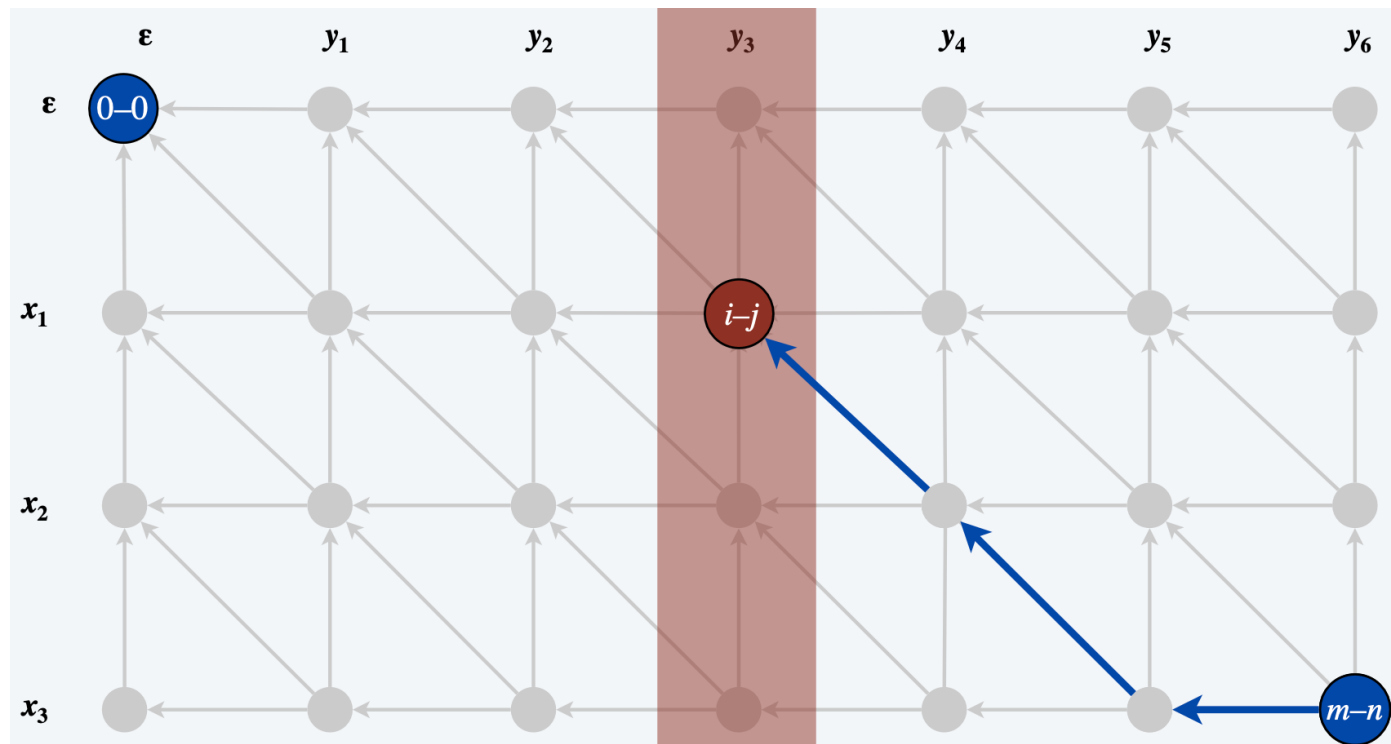




# Sequence Alignment: Hirschberg's Algorithm

- **Edit distance graph.**

- Let  $g(i, j)$  denote length of shortest path from  $(i, j)$  to  $(m, n)$ .
- Can compute  $g(i, j)$  by reversing edges and the roles of  $(0, 0)$  and  $(m, n)$ .
- Can compute  $g(\cdot, j)$  for any specific  $j$  in  $O(mn)$  time and  $O(m)$  space.



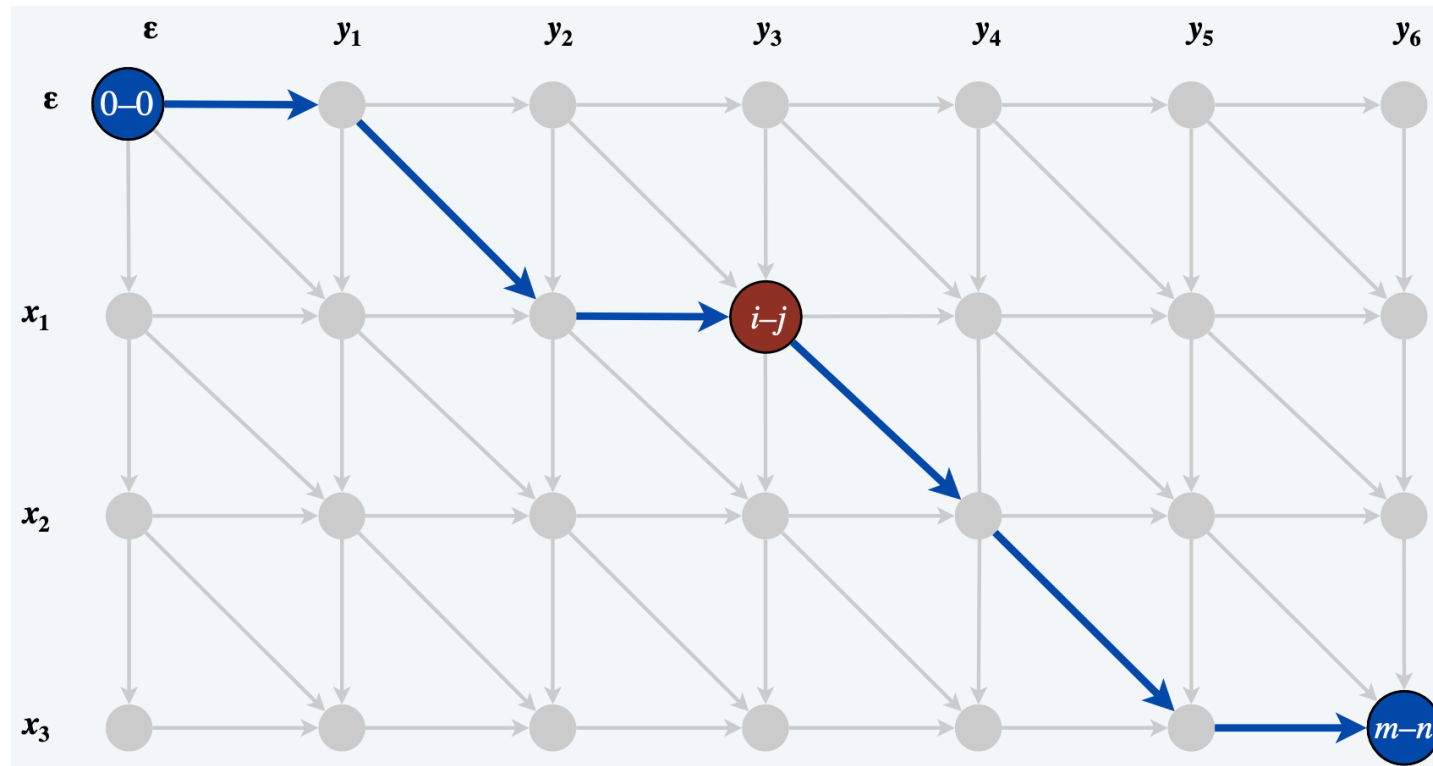
$O(m)$  scrolling array

$\leftarrow g(\cdot, j)$  depends only on  $g(\cdot, j + 1)$



# Sequence Alignment: Hirschberg's Algorithm

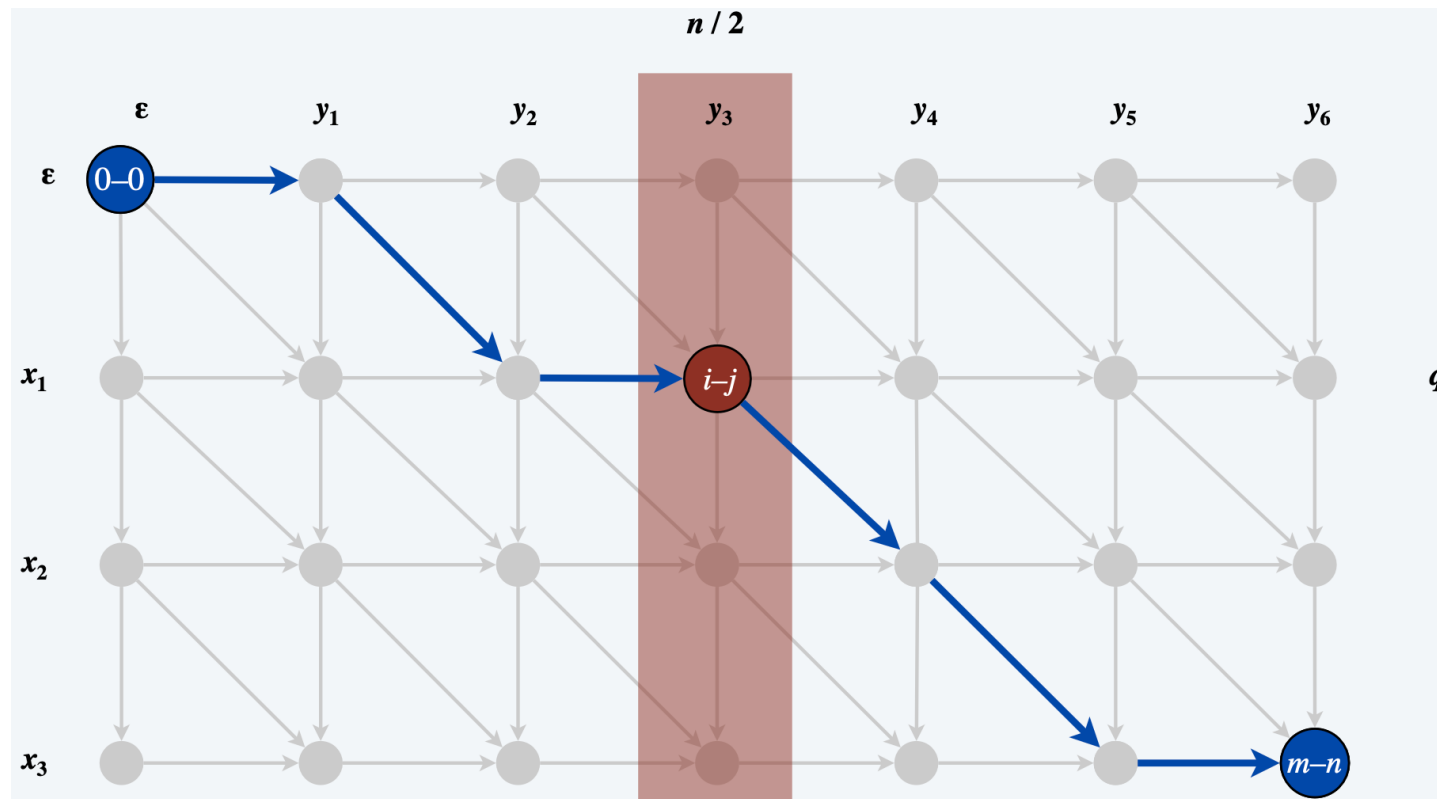
- **Observation.** The length of a shortest path via  $(i, j)$  is  $f(i, j) + g(i, j)$ .





# Sequence Alignment: Hirschberg's Algorithm

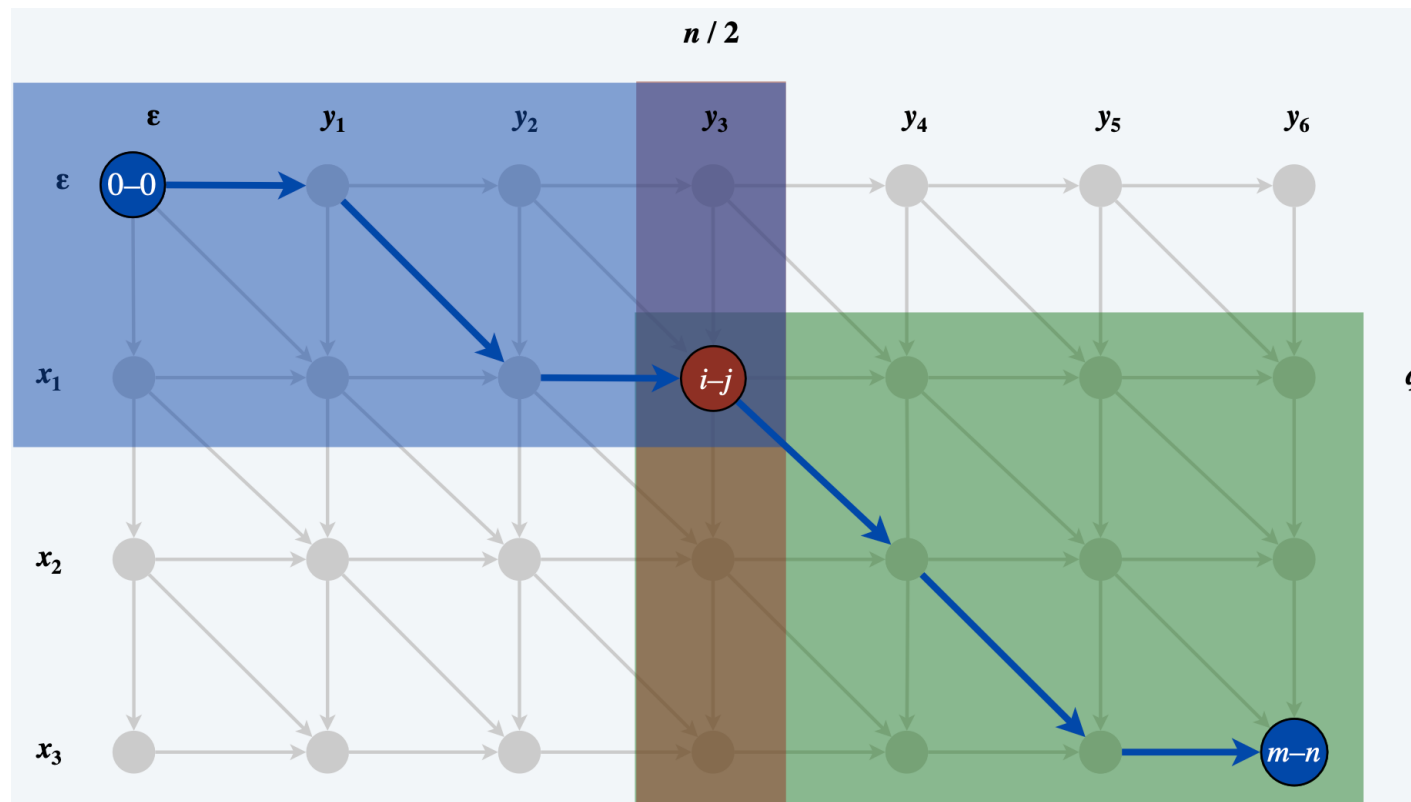
- **Observation.** The length of a shortest path via  $(i, j)$  is  $f(i, j) + g(i, j)$ .
- **Observation.** Let  $q$  be an index that minimizes  $f(q, n/2) + g(q, n/2)$ . Then, there exists a shortest path from  $(0, 0)$  to  $(m, n)$  that uses  $(q, n/2)$ .





# Sequence Alignment: Hirschberg's Algorithm

- **Divide.** Find index  $q$  that minimizes  $f(q, n/2) + g(q, n/2)$ ; save this dividing node as part of solution.
- **Conquer.** Recursively compute optimal alignment in each piece.





# Hirschberg's Algorithm: Analysis

- **Theorem.** Hirschberg's algorithm uses  $O(m + n)$  space.
- Pf.
  - Each recursive call uses  $O(m)$  space to compute  $f(\cdot, n/2)$  and  $g(\cdot, n/2)$ .
  - Only  $O(1)$  space needs to be maintained per recursive call.
  - Number of recursive calls  $\leq n$ . ■
- **Q.** Does Hirschberg's algorithm also run in  $O(mn)$  time?



# Hirschberg's Algorithm: Analysis

- **Theorem.** Hirschberg's algorithm runs in  $O(mn)$  time.
- Pf. (by strong induction on  $m + n$ )
  - $O(mn)$  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:
    - ✓  $T(m, 1) \leq cm$ ,  $T(1, n) \leq cn$ ,  $T(m, n) \leq cmn + T(q, n/2) + T(m - q, n/2)$
  - **Claim.**  $T(m, n) \leq 2cmn$ .
    - ✓ Base cases:  $m = 1$  and  $n = 1$ .
    - ✓ Inductive hypothesis:  $T(m', n') \leq 2cm'n'$  for all  $(m', n')$  with  $m' + n' < m + n$ .
    - ✓  $T(m, n) \leq T(q, n/2) + T(m - q, n/2) + cmn \leq 2cq(n/2) + 2c(m - q)(n/2) + cmn$   
 $= cq(n/2) + c(m - q)(n/2) + cmn = 2cmn$  ■





# Exercise: Longest Common Subsequence

- **Problem.** Given two strings  $X = x_1x_2 \dots x_m$  and  $Y = y_1y_2 \dots y_n$ , find a common subsequence that is **as long as possible**.
- **Alternative viewpoint.** Delete **minimum** number of characters from string  $x$  and string  $y$  such that the resulting strings are the same.
- **Ex.**  $\text{LCS}(\text{GGCACCACG}, \text{ACGGCGGATACG}) = \text{GGCAACG}$
- **Applications.** Unix diff, git, bioinformatics, ...
- **Q.** How to solve this problem via DP?