

# Sequence Alignment

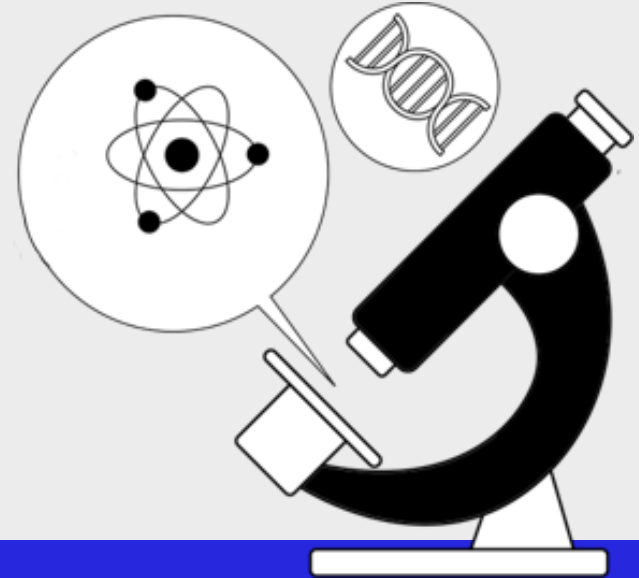
## Dynamic Programming

**Saeedeh Akbari**

Department of Computer Engineering

Sharif University of Technology

Fall 2023



Adapted with modifications from lecture notes prepared by Phillip Compeau  
Bioinformatics Algorithms: An Active Learning Approach



# Table of contents

**01**

Introduction

**02**

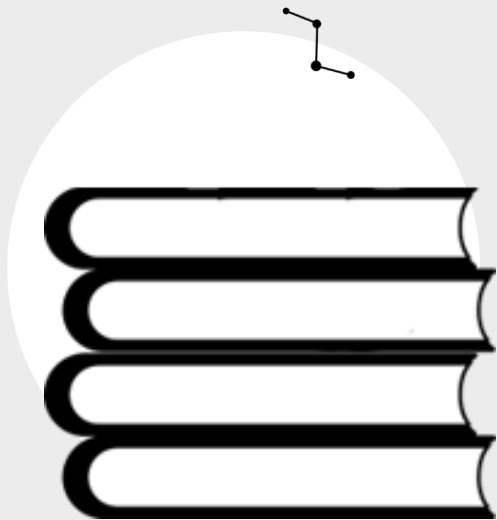
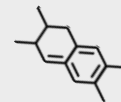
Define the  
problem

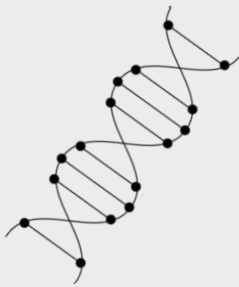
**03**

Solve the  
Problem

**04**

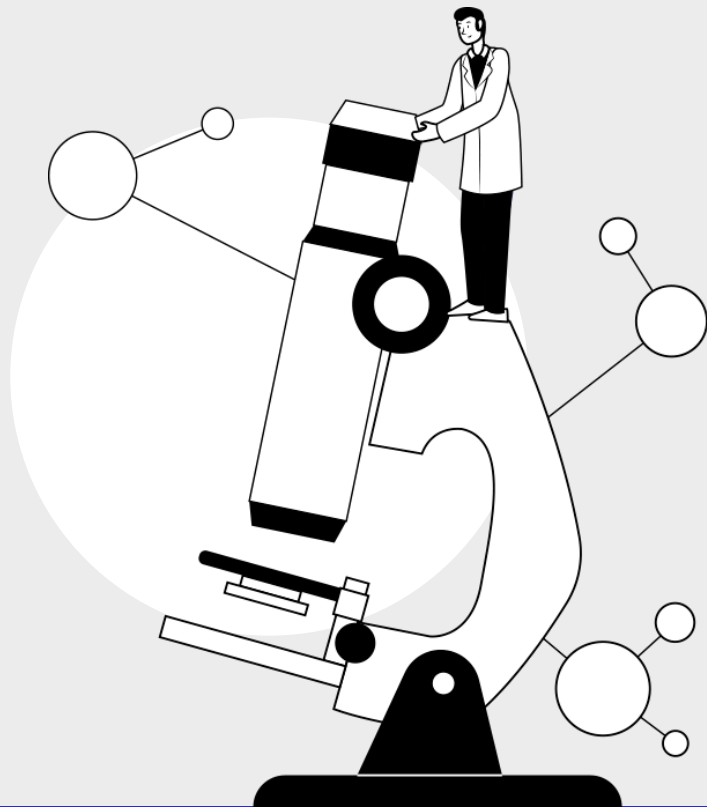
...








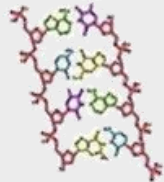




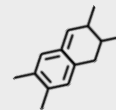
**01**

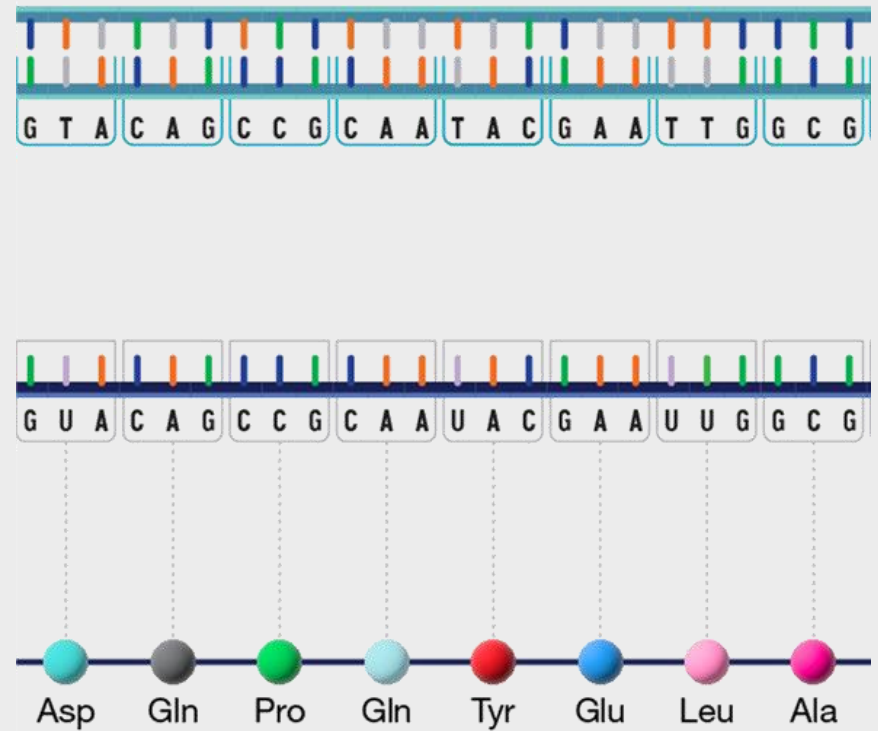
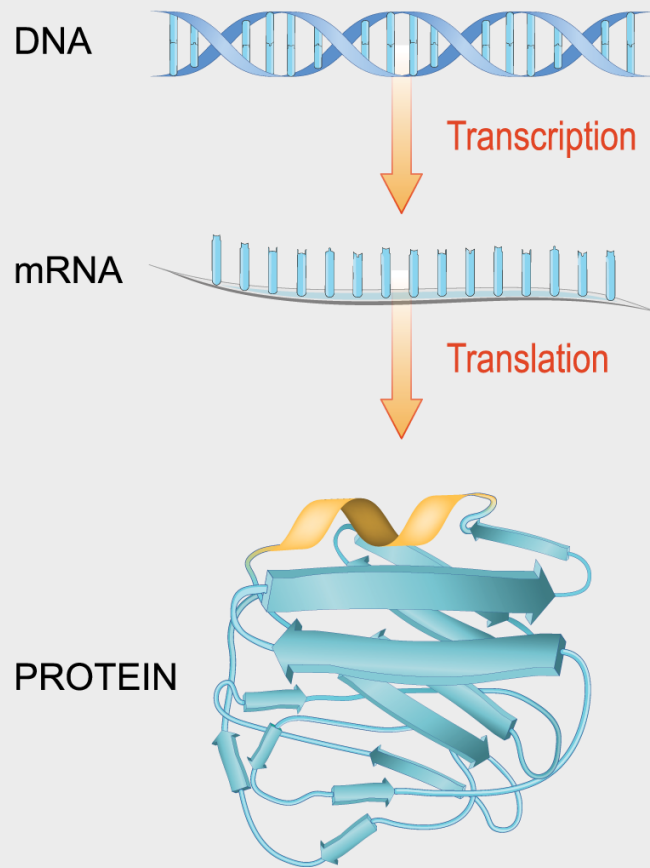
# Introduction



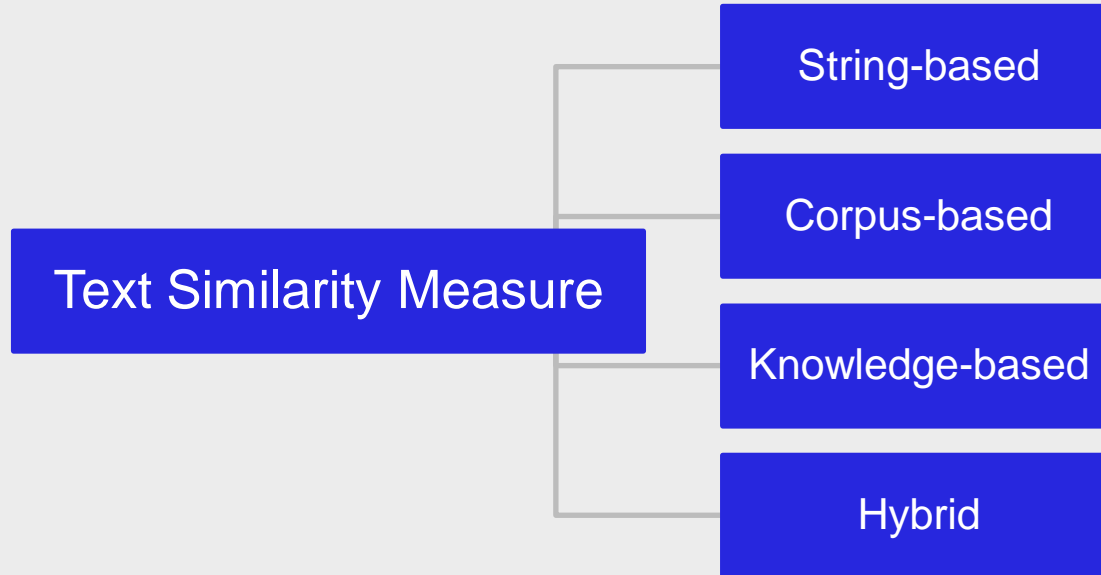


	<p>DNA</p>
	
	<p>GTCA TGGCTATGGC CACAAGGGETGTTGG CTCAATCGACTCCAC CGTCTTTAGCTTGTT ACCAGAGGAATGTT CCTTCACGTGGATCT</p>
	
	

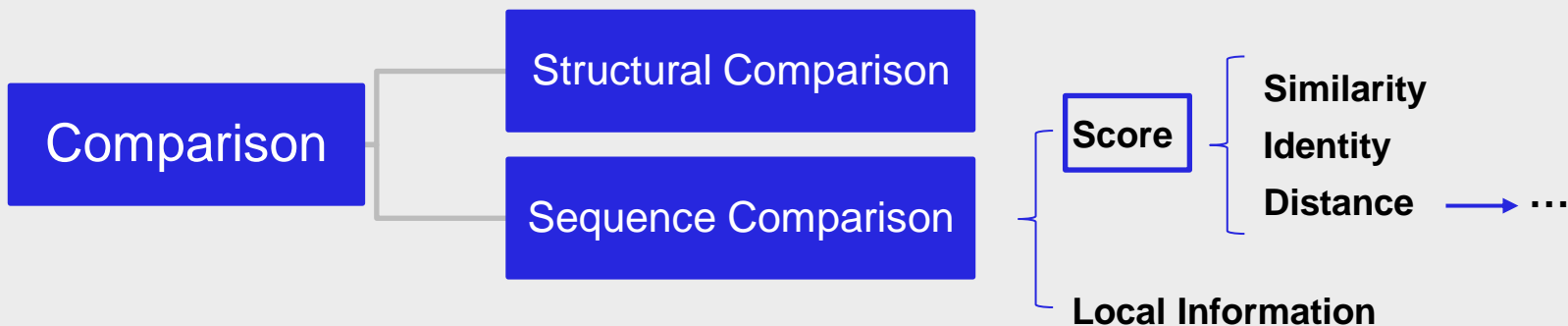




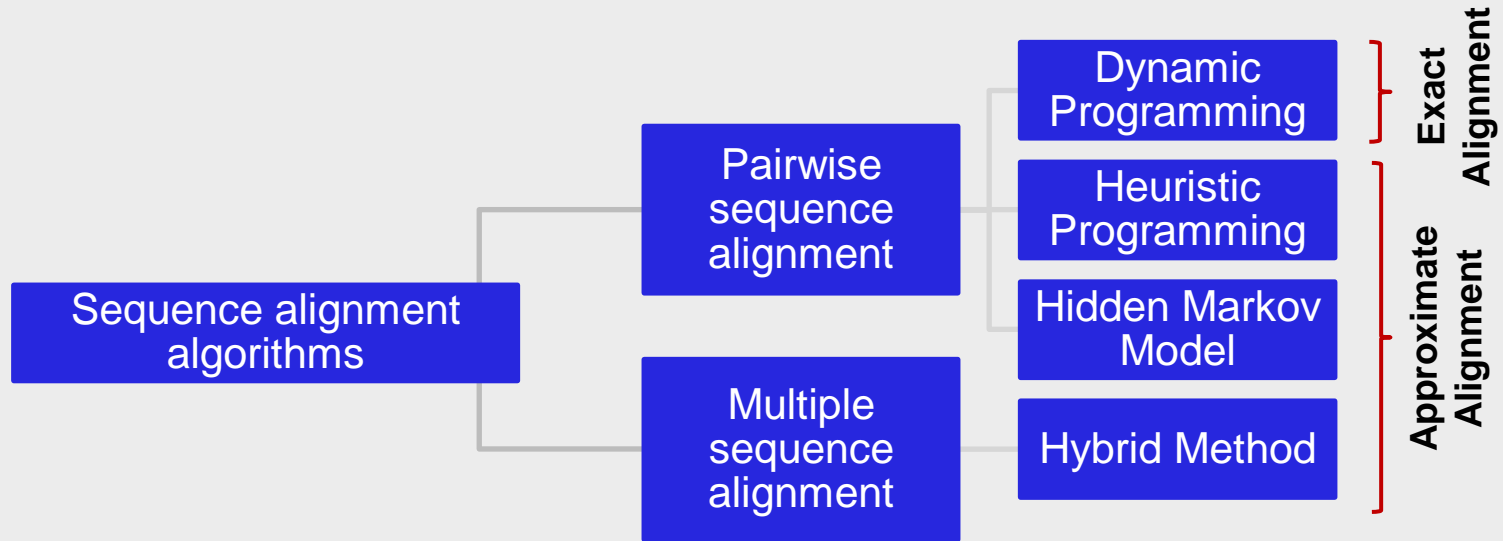
# Text Comparison



# Sequence Comparison



# Sequence Alignment methods



Chang, Xin et al. "Exploring Sequence Alignment Algorithms on FPGA-based Heterogeneous Architectures." International Work-Conference on Bioinformatics and Biomedical Engineering (2014).

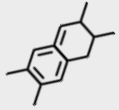


# Sequence Comparison (Data)



<https://www.nature.com/articles/s41467-019-09977-2>

# Sequence Alignment

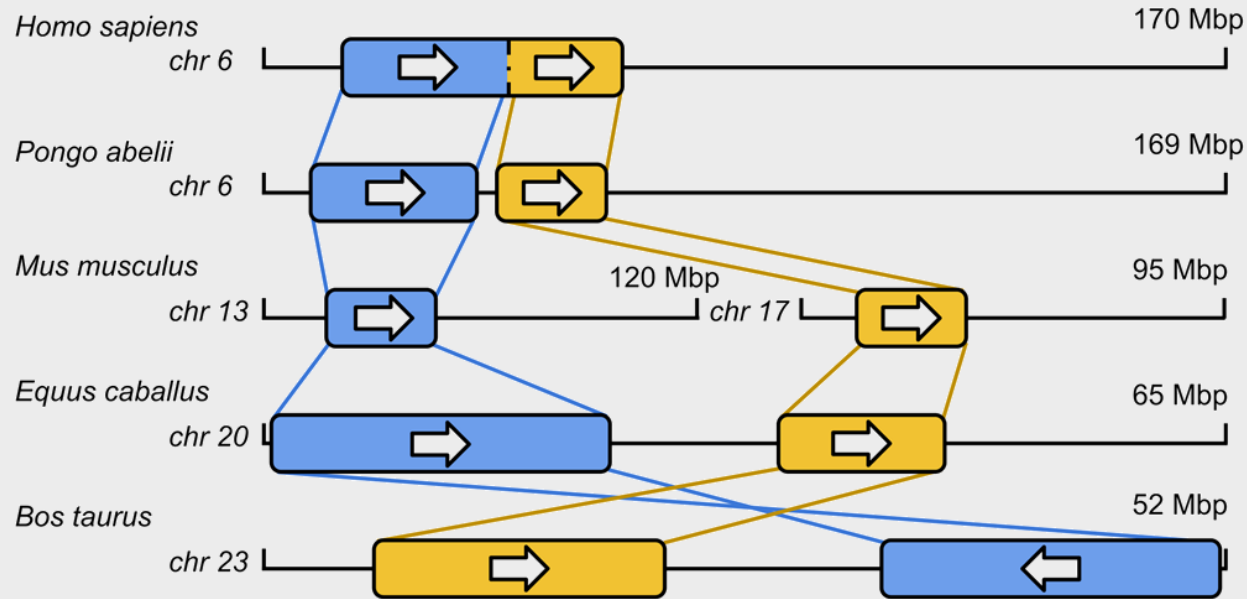


Finding the best-matching piecewise local or global alignments of protein (amino acid) or DNA (nucleic acid) sequences.

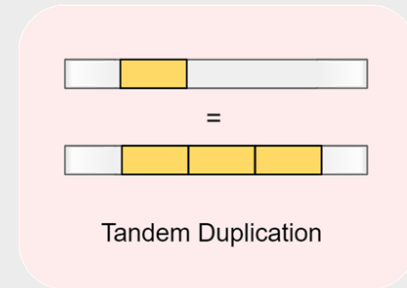
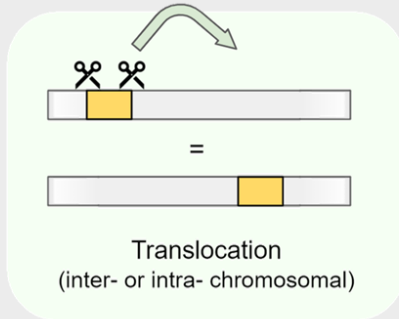
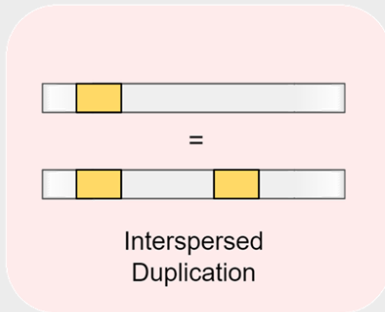
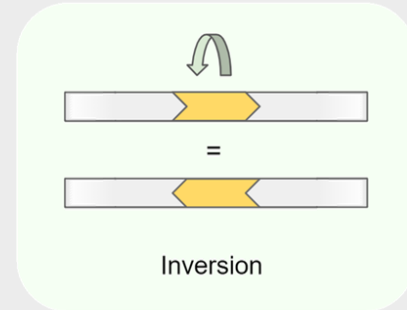
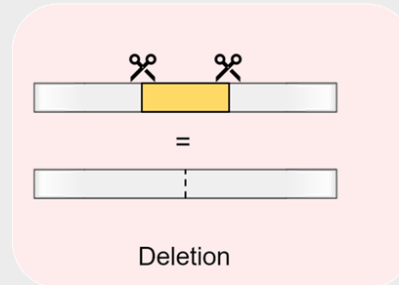
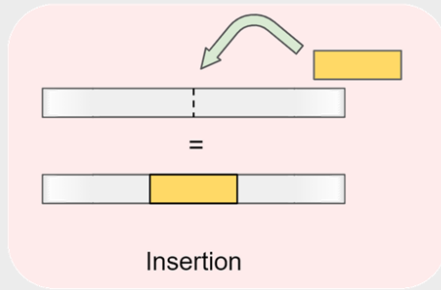
RLA0_PLAF8	-----MAKLSKQQKKQMYIEKLSSLIQQYISKILIVHVDNVGS
RLA0_SULAC	----MIGLAVTTTKKIAKWKVDEVAELTEKCLKTHKTIIIANIEGFPA
RLA0_SULTO	----MRIMAVITQERKIAKWKIEEVKELEOKLREYHTIIIANIEGFPA
RLA0_SULSO	----MKRLALALKQRKVASWKLEEVKELTELKNSNTILIGNLEGFPA
RLA0_AERPE	MSVVS LVGQMYKREKPIPEWKTLMMLRELEELFSKH RVVLFADLTGTPT
RLA0_PYRAE	-MMLAIGKRRYVRTROYPARKVKIVSEATELLQKYPYVFLFDLHGLSS
RLA0_METAC	-----MAEERHHTEHIPQWKKDEIENIKELIQSHKVFGMVGIEGILA
RLA0_METMA	-----MAEERHHTEHIPQWKKDEIENIKELIQSHKVFGMVRIEGILA
RLA0_ARCFU	-----MAAVRGS---PPEYKVRAVEEIKRMISSKPVVAIVSFRNVPA
RLA0_METKA	MAVKAKGQPPSGYE PKVAEWKRREVKELELMDEYENVGLVDLEGIPA



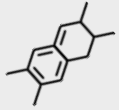
# Structural Variation (SV) Detection



# SV Types



# Sequence Alignment



Finding the best-matching piecewise local or global alignments of protein (amino acid) or DNA (nucleic acid) sequences.

RLA0_PLAF8	-----MAKLSKQQKKQMYIEKLSSLIQQYISKILIVHVDNVGS
RLA0_SULAC	-----MIGLAVTTTKKIAKWKVDEVAELTEKCLKTHKTIIIANIEGFPA
RLA0_SULTO	-----MRIMAVITQERKIAKWKIEEVKELEOKLREYHTIIIANIEGFPA
RLA0_SULSO	-----MKRLALALKQRKVASWKLEEVKELTELKNSNTILIGNLEGFPA
RLA0_AERPE	MSVVS LVGQMYKREKPIPEWKTLMMLRELEELFSKH RVVLFADLTGTPT
RLA0_PYRAE	-MMLAIGKRRYVRTROYPARKVKIVSEATELLQKYPYVFLFDLHGLSS
RLA0_METAC	-----MAEERHHTEHIPQWKKDEIENIKELIQSHKVFGMVGIEGILA
RLA0_METMA	-----MAEERHHTEHIPQWKKDEIENIKELIQSHKVFGMVRIEGILA
RLA0_ARCFU	-----MAAVRGS---PPEYKVRAVEEIKRMISSKPVVAIVSFRNVPA
RLA0_METKA	MAVKAKGQPPSGYE PKVAEWKRREVKELELMDEYENVGLVDLEGIPA



# Pairwise Alignment

Seq1: tcctctgctctgccatcat---caaccccaaagt  
          | | | | | | | | | | | | | | | | | | | | | |  
Seq2: tcctgtgcatctgcaatcatgggcaaccccaaagt

The sequences are padded with gaps (dashes) so that wherever possible, columns contain identical characters from the sequences involved

# Causes for Sequence (Dis)similarity

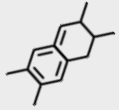
**mutation:** a nucleotide at a certain location is replaced by another nucleotide (e.g.: A**T**A → A**G**A)

**insertion:** at a certain location one new nucleotide is inserted in two existing nucleotides (e.g.: AA → A**G**A)

**deletion:** at a certain location one existing nucleotide is deleted (e.g.: AC**T**G → AC-**G**)

**indel:** an **in**sertion or a **de**letion

# Sequence Alignment



Finding the best-matching piecewise **local or global** alignments of protein (amino acid) or DNA (nucleic acid) sequences.

```
L G P S S K Q T G K G S - S R I W D N
|           |       | | |       |
L N - I T K S A G K G A I M R L G D A
```

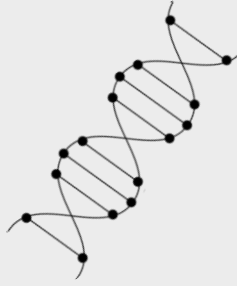
Global alignment

```
- - - - - T G K G - - - - -
          | | |
          A G K G - - - - -
```

Local alignment

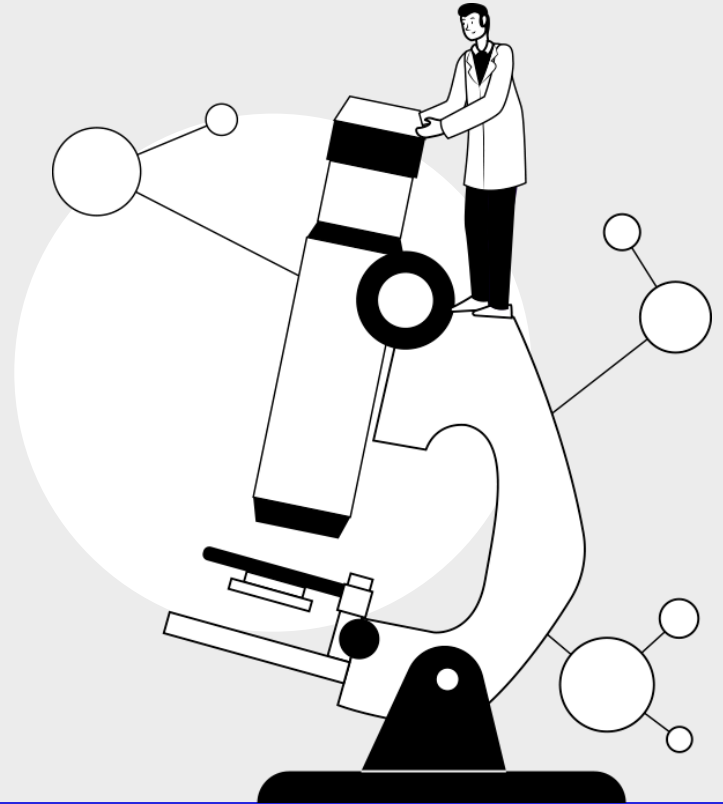






**02**

# Define the problem



# Comparing Genes

**Problem:** *Compare two similar genes.*

- **Input:** Two genes.
- **Output:** How “similar” these genes are.

**STOP and Think:** How can we make this into a well-defined computational problem?

# Hamming Distance

**Hamming Distance Problem:** *Compute the number of mismatching symbols between two strings.*

- **Input:** Two strings.
- **Output:** The number of “mismatched” symbols in the two strings.

**Seq. 1 : ATGCATGC**

**Seq. 2 : TGCATGCA**

Hamming distance = 8

# Toward a More Accurate Problem...

Seq.1 : **A**TGCATGC–

Seq.2 : –TGCATGCA

“Sliding” the strings reveals other similarities.

Furthermore, these strings have different lengths, so how should we compare them?

# Toward a More Accurate Problem...

Seq. 1 : ATGCTTA            A**TGC**–**TTA**–  
Seq. 2 : TGCATTAA      –**TGC****ATTAA**

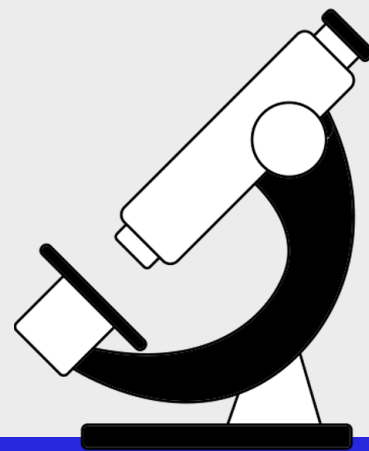
**Symbol Matching Problem:** *Match as many symbols as possible between two strings.*

- **Input:** Two strings.
- **Output:** The greatest number of matched symbols in any “alignment” of the two strings.

# Solve the Problem as a Game

- **At each turn, you have two choices:**
  1. You can remove the first symbol from each sequence, in which case you earn a point if the symbols match;
  2. Alternatively, you can remove the first symbol from either of the two sequences, in which case you earn no points but may set yourself up to earn more points in later moves.
- **Your goal is to maximize the number of points.**

Growing alignment	Remaining symbols	Score
	A T G T T A T A A T C G T C C	
A A	T G T T A T A T C G T C C	+1
A T A T	G T T A T A C G T C C	+1
A T - A T C	G T T A T A G T C C	
A T - G A T C G	T T A T A T C C	+1
A T - G T A T C G T	T A T A C C	+1
A T - G T T A T C G T -	A T A C C	
A T - G T T A A T C G T - C	T A C	
A T - G T T A T A T C G T - C -	A C	
A T - G T T A T A A T C G T - C - C		





# From a Game to an Alignment

Given two strings  $v$  and  $w$ , an **alignment** of  $v$  and  $w$  is a two-row matrix such that:

- the first row contains symbols of  $v$
- the second row contains symbols of  $w$
- each row may also contain **gap symbols** (“-”)

**Seq.1: A T - G T T A T A**

**Seq.2: A T C G T - C - C**

# From a Game to an Alignment

Given two strings  $v$  and  $w$ , an **alignment** of  $v$  and  $w$  is a two-row matrix such that:

- the first row contains symbols of  $v$
- the second row contains symbols of  $w$
- each row may also contain **gap symbols** (“-”)

Seq. 1: **A** **T** - **G** T T A T A

Seq. 2: **A** **T** C **G** T - C - C

**Matches**

# From a Game to an Alignment

Given two strings  $v$  and  $w$ , an **alignment** of  $v$  and  $w$  is a two-row matrix such that:

- the first row contains symbols of  $v$
- the second row contains symbols of  $w$
- each row may also contain **gap symbols** (“-”)

Seq.1: A T - G T T A T A

Seq.2: A T C G T - C - C

**Mismatches**

# From a Game to an Alignment

Given two strings  $v$  and  $w$ , an **alignment** of  $v$  and  $w$  is a two-row matrix such that:

- the first row contains symbols of  $v$
- the second row contains symbols of  $w$
- each row may also contain **gap symbols** (“-”)

Seq. 1: A T - G T T A T A

Seq. 2: A T C G T - C - C

Insert

# From a Game to an Alignment

Given two strings  $v$  and  $w$ , an **alignment** of  $v$  and  $w$  is a two-row matrix such that:

- the first row contains symbols of  $v$
- the second row contains symbols of  $w$
- each row may also contain **gap symbols** (“-”)

Seq.1: A T - G T T A T A

Seq.2: A T C G T - C - C

Delete

# Longest Common Subsequence

A **common subsequence** of  $v$  and  $w$  is a sequence of symbols occurring (not necessarily contiguously) in both  $v$  and  $w$ .

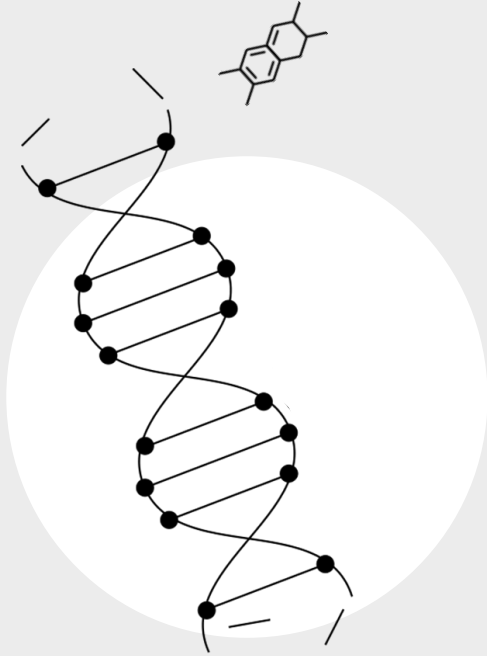
The **matches** in an alignment of  $v$  and  $w$  form a common subsequence of  $v$  and  $w$ .

Seq. 1: **A T** - **G** T T A T A

Seq. 2: **A T** C **G** T - C - C

**Matches**

# DNA Alignment



# DNA Alignment

- Aligning these three sequences from humans, chimpanzees, and macaques allows us to determine which positions in the sequence should be compared with one another.

## Unaligned sequences

Human	A	C	A	T	T	A	T	G	G	A	C	A	G	G	T	A	A	G	T	A	A	A	A	A	C	A	T	A	T	T	
Chimpanzee	A	C	A	T	T	A	T	G	G	A	C	A	G	G	T	A	A	G	T	A	A	A	A	A	C	A	T	A	T	T	
Macaque	A	T	A	T	A	C	A	T	T	A	C	G	G	A	C	A	G	G	T	A	A	G	T	A	A	A	A	A	C	A	T

**15** spots are identical.

## Aligned sequences

Human	A	C	A				T	T	A	T	G	G	A	C	A	G	G	T	A	A	G	T	A	A	A	A	A	A	C	A	T	A	T	T	
Chimpanzee	A	C	A				T	T	A	T	G	G	A	C	A	G	G	T	A	A	G	T	A	A	A	A	A	A	A	C	A	T	A	T	T
Macaque	A	T	A	T	A	C	A	T	T	A	C	G	G	A	C	A	G	G	T	A	A	G	T	A	A	A	A	A	C	A	T				

**22** spots are identical.



# The Problems are the Same!

## **Longest Common Subsequence Problem:**

*Find a longest common subsequence of two strings.*

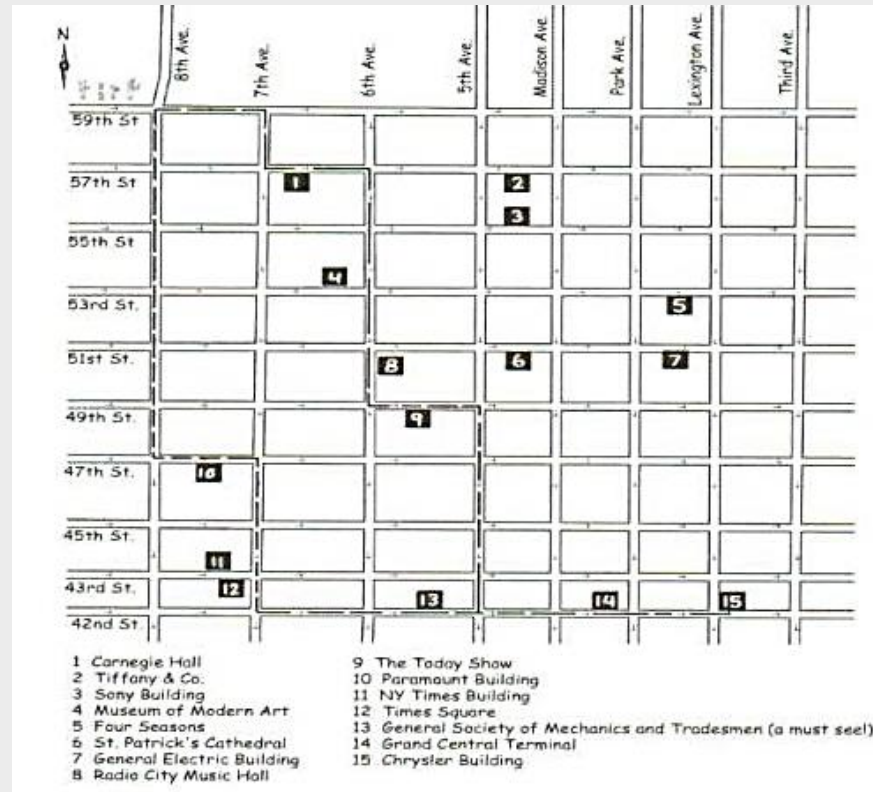
- **Input:** Two strings.
- **Output:** A longest common subsequence of these strings.

## **Symbol Matching Problem:**

*Match as many symbols as possible between two strings.*

- **Input:** Two strings.
- **Output:** The greatest number of matched symbols in any “alignment” of the two strings.

# Manhattan Tourist Problem (MTP)

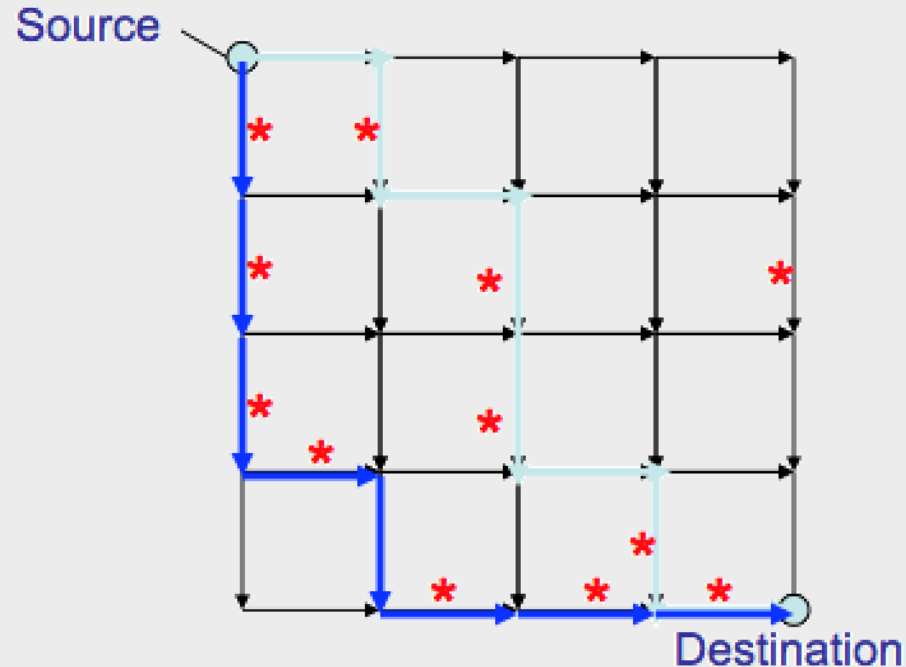


# Manhattan Tourist Problem (MTP)

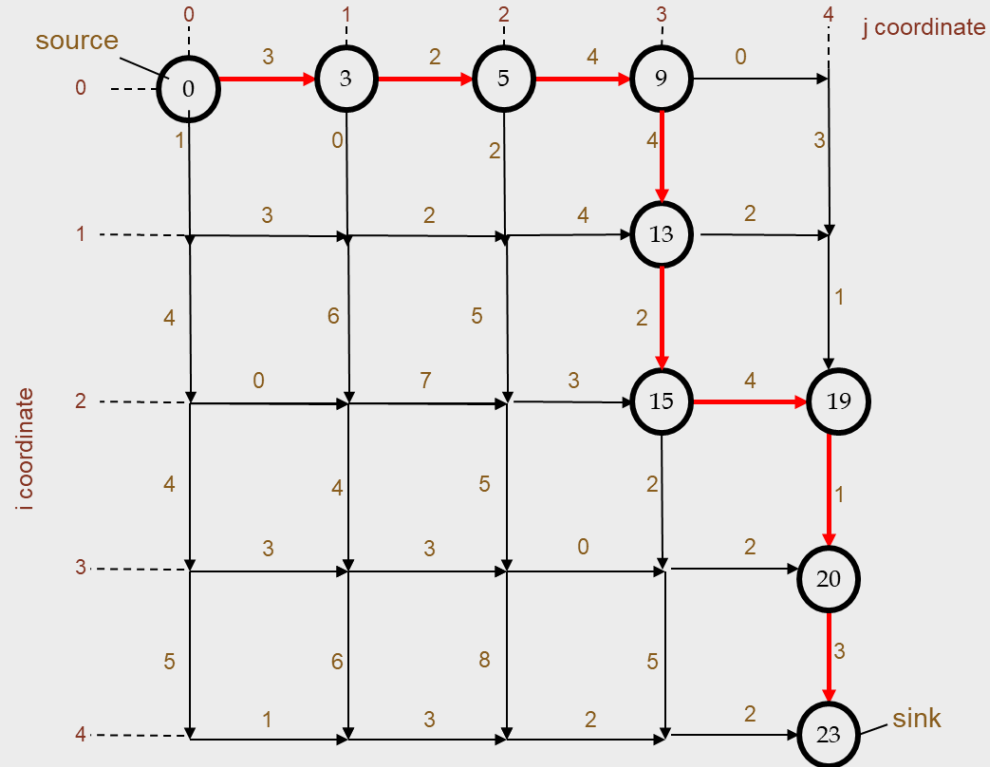
**MTP:** *Imagine seeking a path (from source to sink) to travel (only eastward and southward) with the most number of attractions (\*) in the Manhattan grid*

**Weight of edge:** number of attractions along the edge.

**Goal:** Find a longest path from source to sink.



# MTP: An Example



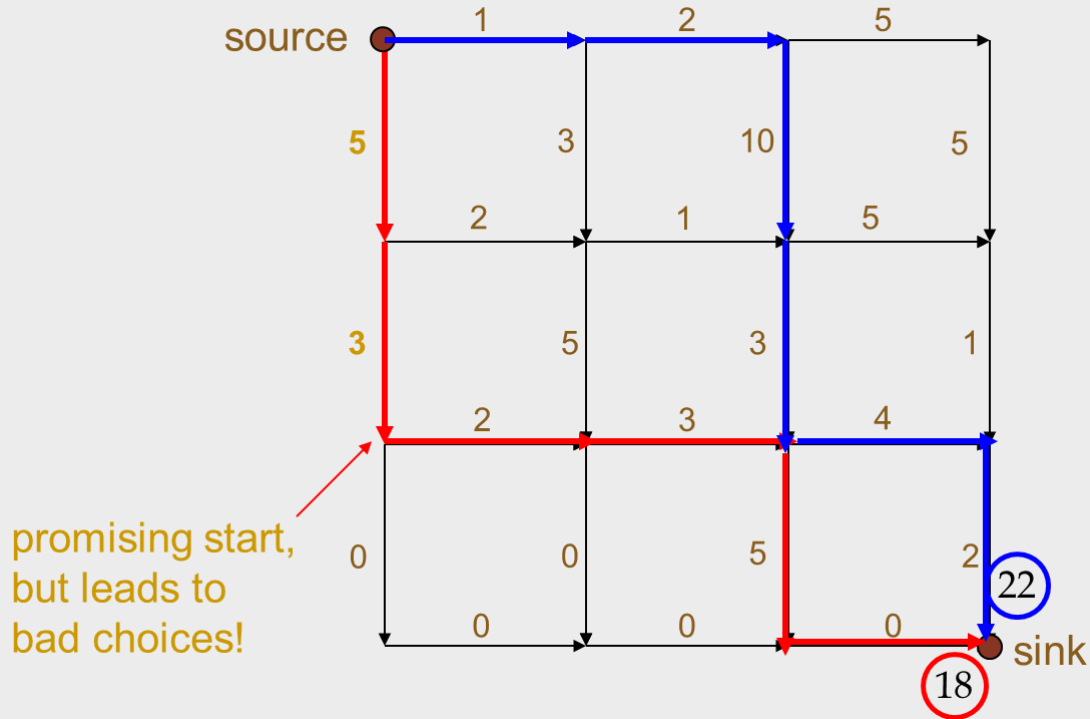
# Toward a Computational Problem

**MTP:** *Find a longest path in a rectangular city.*

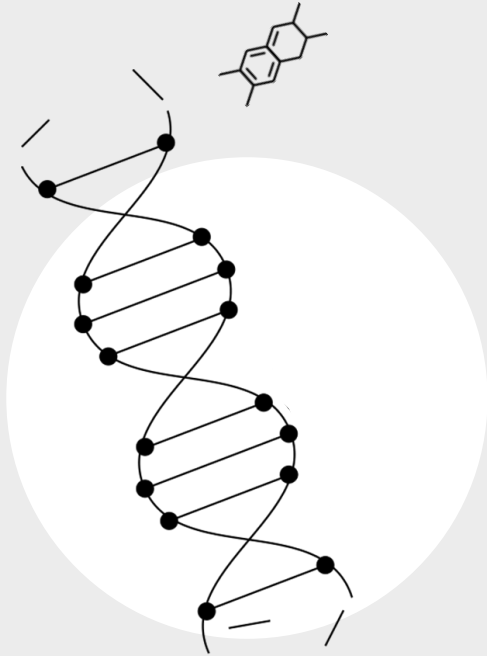
- **Input:** A weighted  $n \times m$  rectangular grid ( $n + 1$  rows and  $m + 1$  columns).
- **Output:** A longest path from source  $(0, 0)$  to sink  $(n, m)$  in the grid.

How many different paths are there from source to sink in a  $16 \times 12$  rectangular grid?

# MTP: Greedy Algorithm Is Not Optimal

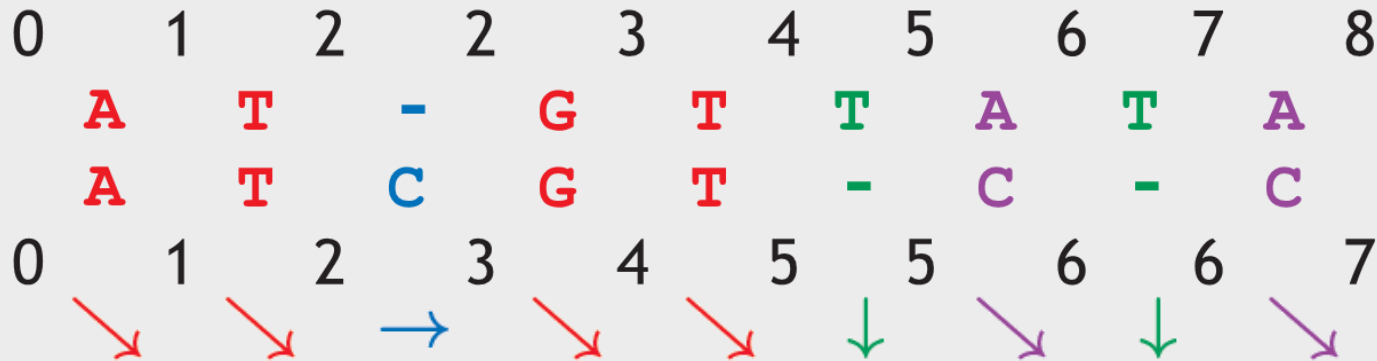


# Returning to Sequence Alignment ...



# Like a City

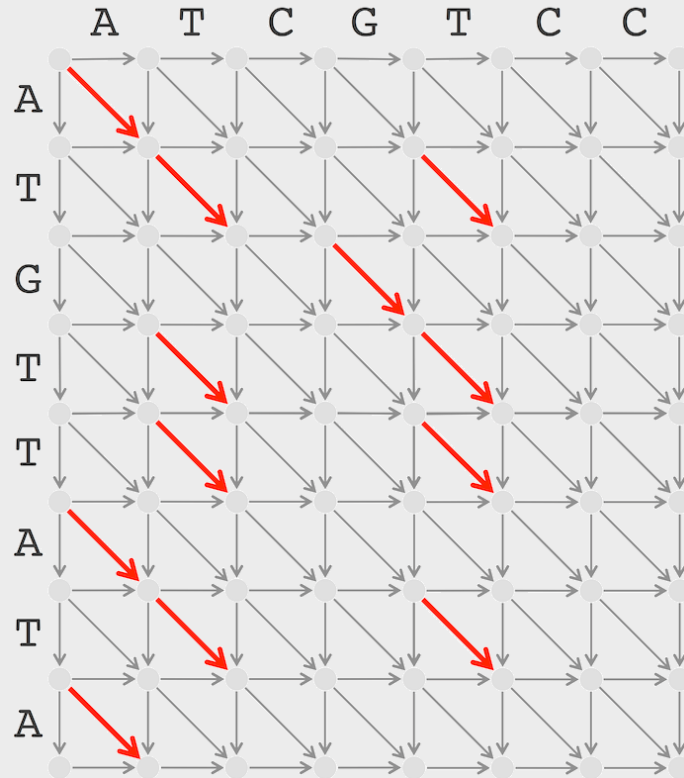
match/mismatch ( $\searrow$  /  $\swarrow$ ), insertion ( $\rightarrow$ ), or deletion ( $\downarrow$ )



$(0,0) \searrow (1,1) \searrow (2,2) \rightarrow (2,3) \searrow (3,4) \searrow (4,5) \downarrow (5,5) \swarrow (6,6) \downarrow (7,6) \swarrow (8,7)$

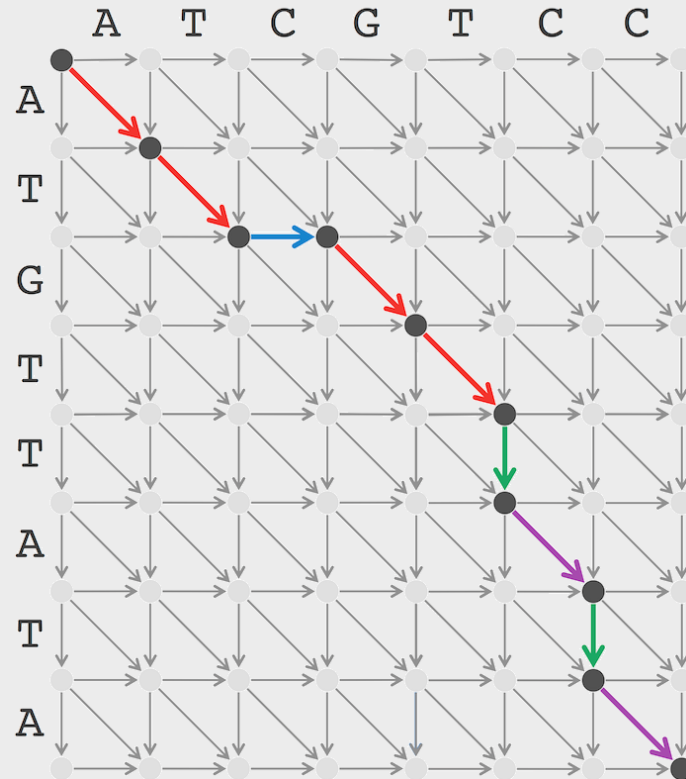


# Representing an Alignment as a Path



If both sequences are the same, what pattern is seen?

# Representing an Alignment as a Path



# Making Change

Different coin values are called **denominations**.

## USA Denominations

(100, 50, 25, 10, 5, 1)

## Roman Republic Denominations

(120, 40, 30, 24, 20, 10, 5, 4, 1)

**STOP and Think:** What algorithm would you propose to provide as few coins as possible when making change for a given set of denominations?

# Greedy Change

1. Given an amount *money*, Choose the largest coin denomination remaining that is less than or equal to *money*.
2. Subtract *coin* from *money*.
3. If *money* is equal to zero, STOP.  
Otherwise, return to step 1.

**STOP and Think:** Does **Greedy Change** always make change with the minimum number of coins?

# Greedy Change is Suboptimal!

## Roman Republic Denominations

(120, 40, 30, 24, 20, 10, 5, 4, 1)

When changing 48 Roman denarii, GreedyChange would suggest:

**five coins ( $48 = 40 + 5 + 1 + 1 + 1$ )**

**But  $48 = 24 + 24$**

# Making Change Recursively

Say that you need to change 76 denarii, and you only have three denominations: *Coins* = (5, 4, 1).

A minimum collection of coins totaling 76 denarii must be one of the following:

- a minimal collection of coins totaling 75 denarii, plus a 1-denarius coin;
- a minimal collection of coins totaling 72 denarii, plus a 4-denarius coin;
- a minimal collection of coins totaling 71 denarii, plus a 5-denarius coin;

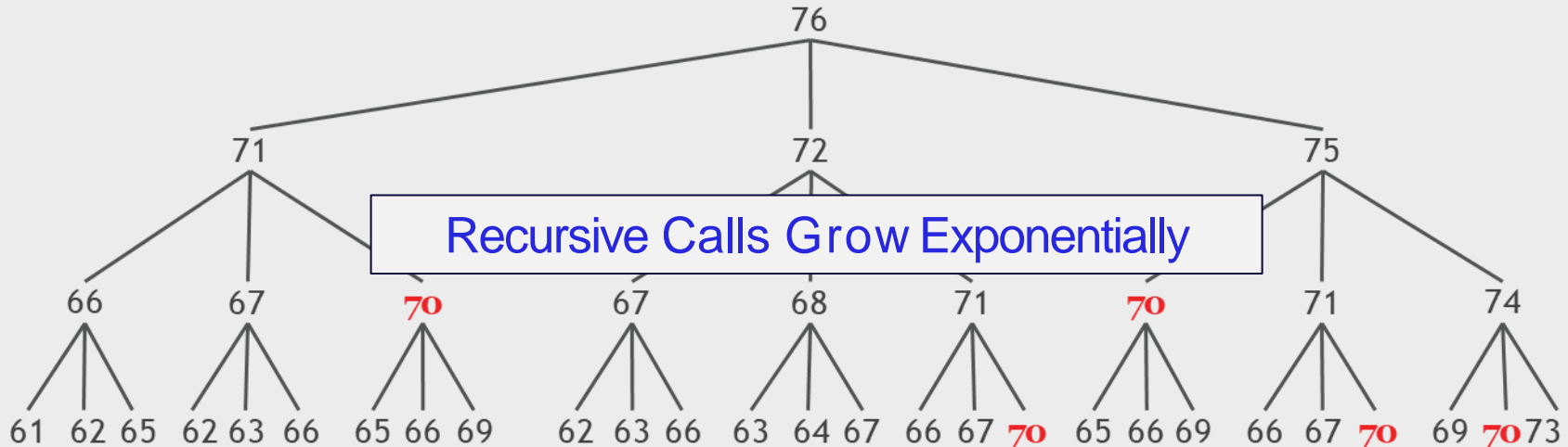
# Making Change Recursively

Let  $\text{MinNumCoins}(\text{money})$  denote the minimum number of coins needed to change an amount  $\text{money}$  for a collection of denominations  $\text{Coins}$ .

$$\text{MinNumCoins}(\text{money}) = \min \left\{ \begin{array}{l} \text{MinNumCoins}(\text{money} - \text{coin}_1) + 1 \\ \vdots \\ \text{MinNumCoins}(\text{money} - \text{coin}_d) + 1 \end{array} \right.$$

# Making Change Recursively

Say that you need to change 76 denarii, and you only have three denominations: *Coins* = (5, 4, 1).





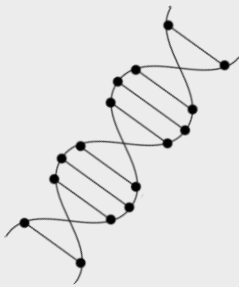
# Change with Dynamic Programming

Coins = (1, 4, 5).

$m$	0	1	2	3	4	5	6	7	8	9	10	11	12
<b>MINNUMCOINS(<math>m</math>)</b>	0	1	2	3	1	1	2	3	2	2	2	3	3

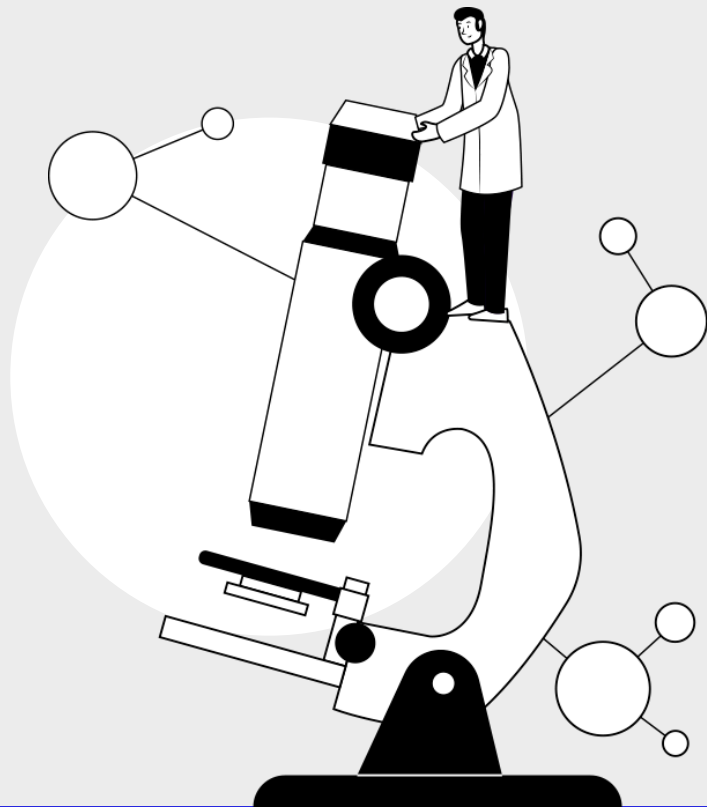
1. Set MinNumCoins(0) equal to zero.
2. For each value of  $j$  between 1 and  $money$ , set MinNumCoins( $j$ ) equal to 1 + the minimum of MinNumCoins( $money - Coin[i]$ ) over all  $i$ .
3. Return MinNumCoins( $money$ ).

**Point:** The trick behind dynamic programming is to solve each of the smaller problems once rather than billions of times.



**03**

# **Solve the Problem**



# MTP: Recursive Algorithm

```
SouthOrEast( $i, j$ )  
  if  $i = 0$  and  $j = 0$   
    return 0  
   $x \leftarrow -\infty, y \leftarrow -\infty$   
  if  $i > 0$   
     $x \leftarrow \text{SouthOrEast}(i - 1, j) + \text{weight of the vertical edge into } (i, j)$   
  if  $j > 0$   
     $y \leftarrow \text{SouthOrEast}(i, j - 1) + \text{weight of the horizontal edge into } (i, j)$   
  return  $\max\{x, y\}$ 
```

**Exercise Break:** How many times is **SoutOrEast(3,2)** called in the computation of **SouthOrEast(9, 7)**?

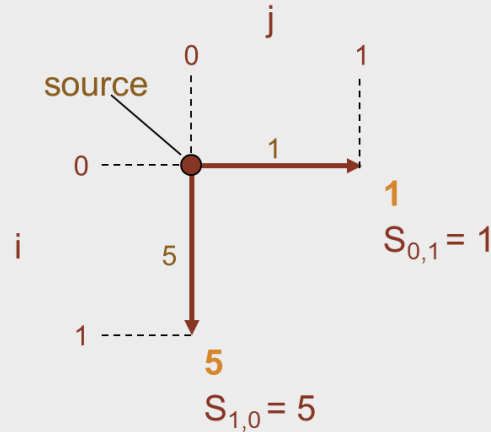
# MTP: Dynamic Programming

Computing the score for a point  $(i, j)$  by the recurrence relation:

$$s_{i,j} = \max \left\{ \begin{array}{l} s_{i-1,j} + \text{weight of the edge between } (i-1, j) \text{ and } (i, j) \\ s_{i,j-1} + \text{weight of the edge between } (i, j-1) \text{ and } (i, j) \end{array} \right.$$

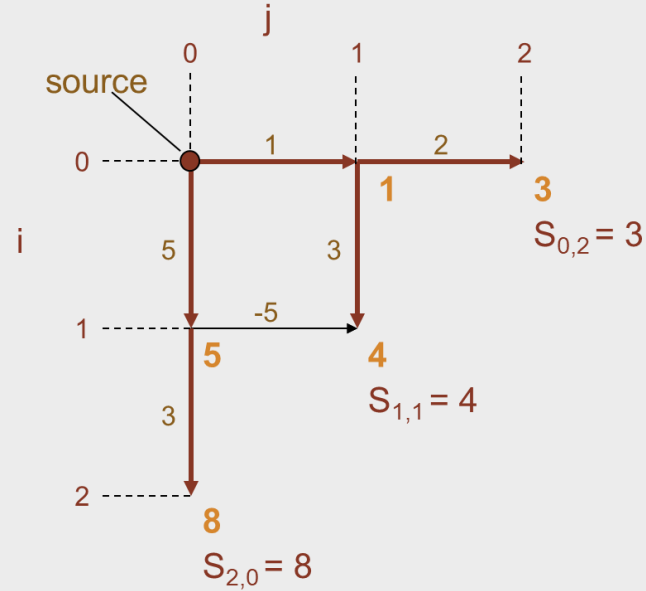
The running time is  $n \times m$  for a  $n$  by  $m$  grid  
( $n$  = # of rows,  $m$  = # of columns)

# MTP: Dynamic Programming

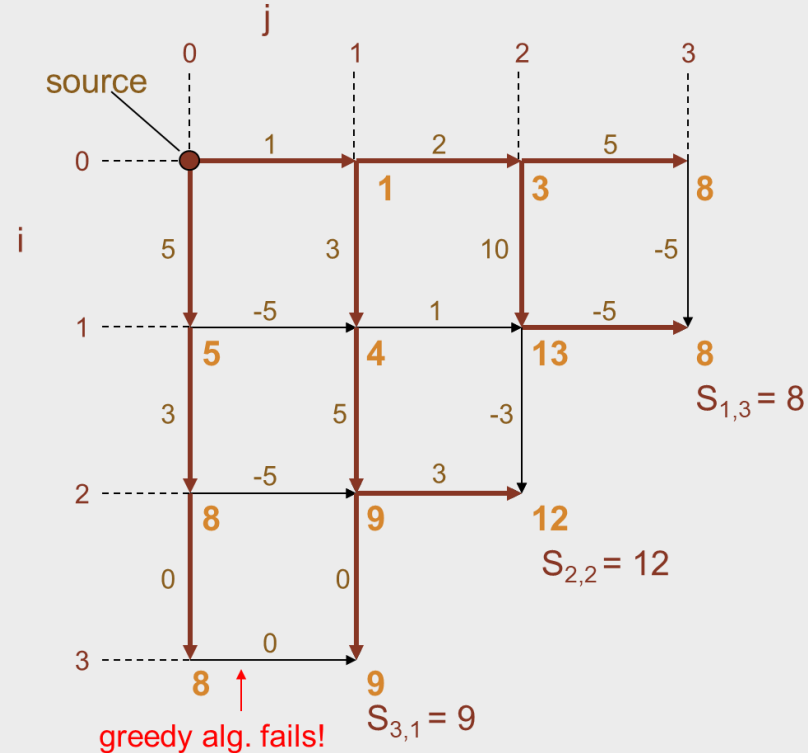


- Calculate optimal path score for each vertex in the graph
- Each vertex's score is the maximum of the prior vertices score plus the weight of the respective edge in between

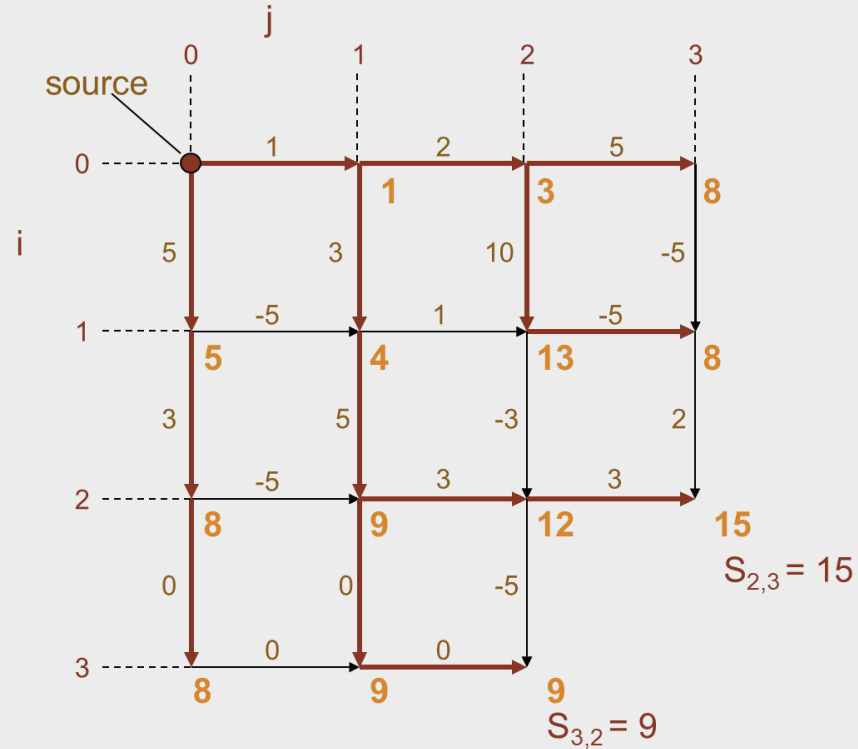
# MTP: Dynamic Programming



# MTP: Dynamic Programming

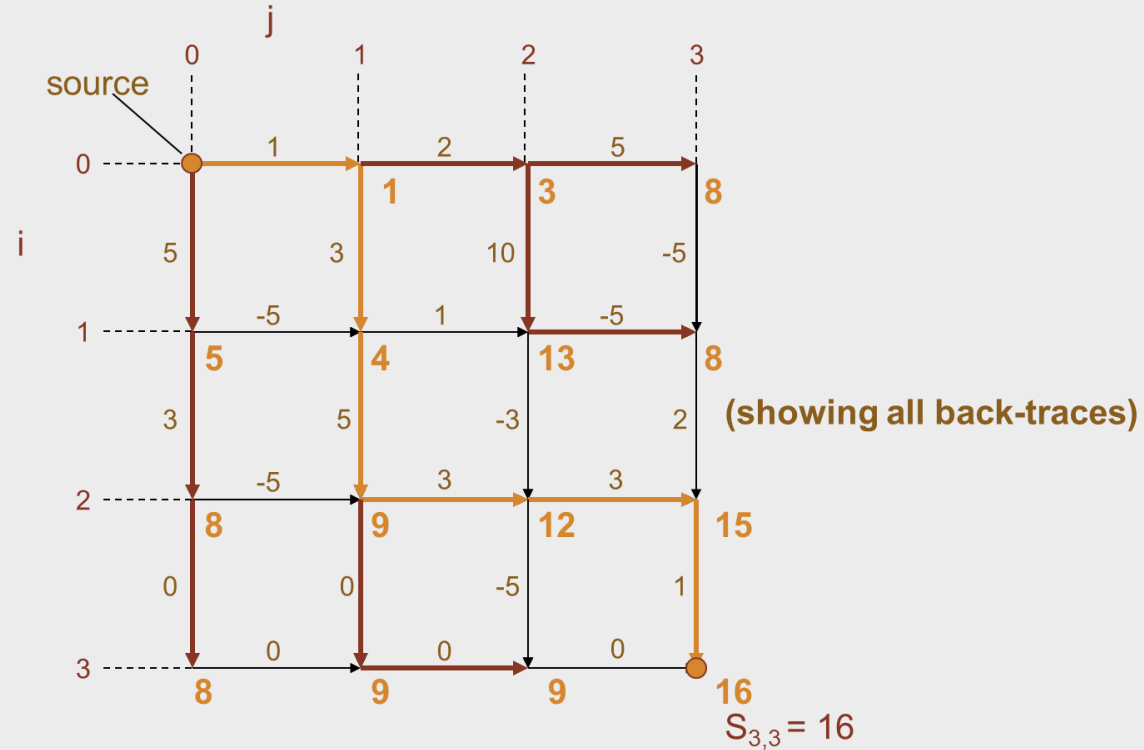


# MTP: Dynamic Programming

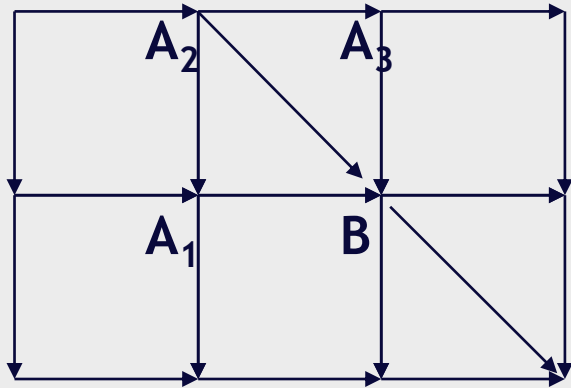




# MTP: Dynamic Programming



# Add other edge...

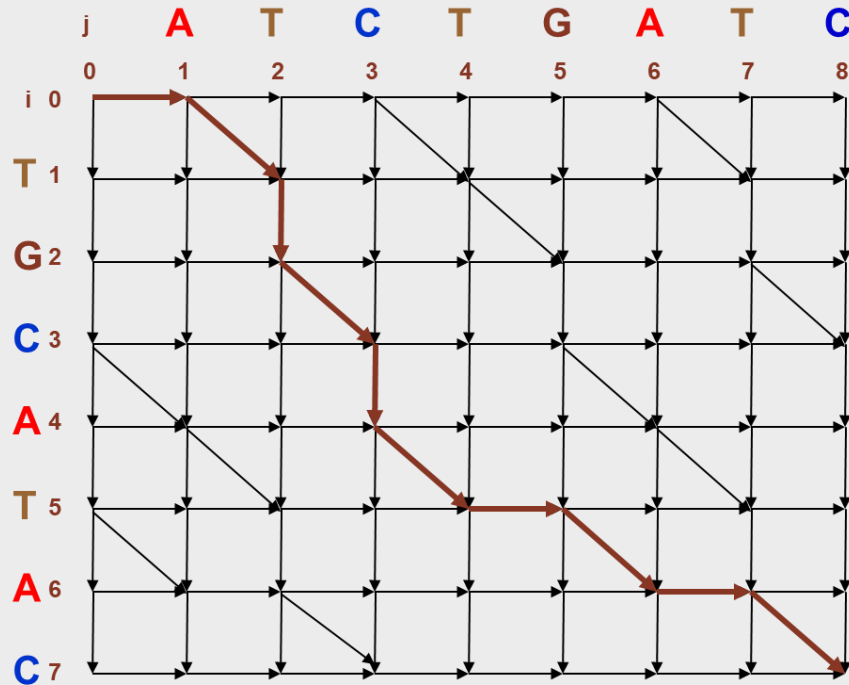


What about diagonals?

The score at point B is given by:

$$s_B = \max \text{ of } \begin{cases} s_{A_1} + \text{weight of the edge } (A_1, B) \\ s_{A_2} + \text{weight of the edge } (A_2, B) \\ s_{A_3} + \text{weight of the edge } (A_3, B) \end{cases}$$

# Add other edge...



Every path is a common subsequence.

Every diagonal edge adds an extra element to common subsequence

**LCS Problem:** Find a path with maximum number of diagonal edges

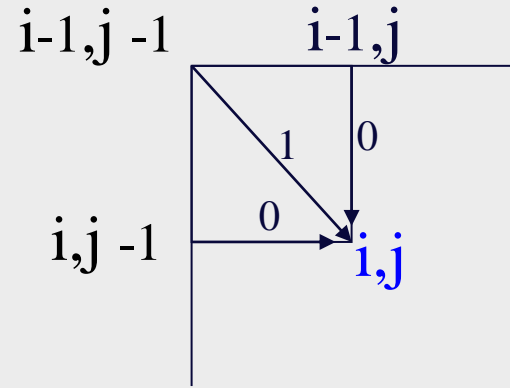
# Computing LCS

Let  $\mathbf{v}_i$  = prefix of  $\mathbf{v}$  of length  $i$ :  $v_1 \dots v_i$

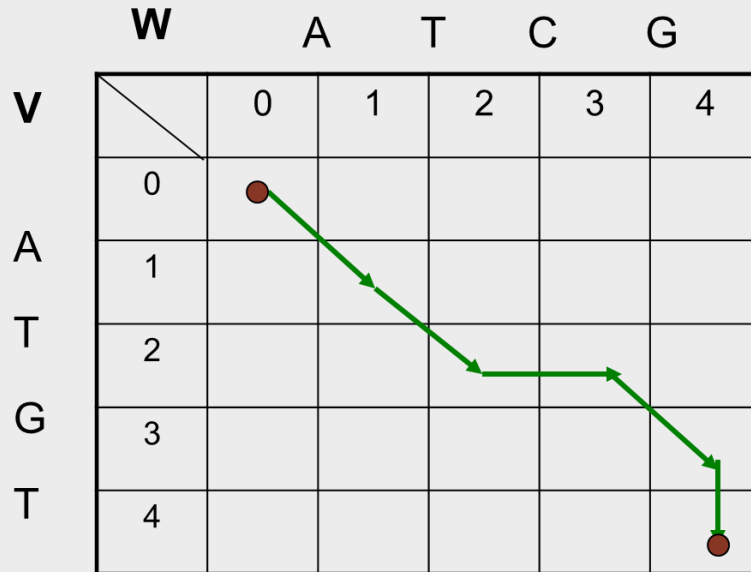
and  $\mathbf{w}_j$  = prefix of  $\mathbf{w}$  of length  $j$ :  $w_1 \dots w_j$

The length of  $\text{LCS}(\mathbf{v}_i, \mathbf{w}_j)$  is computed by:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1 \text{ if } v_i = w_j \end{cases}$$



# Every Path in the Grid Corresponds to an Alignment



0 1 2 2 3 4
   
 V = A T - G T
   
   | |   |
   
 W = A T C G -
   
   0 1 2 3 4 4

# Edit Distance

Levenshtein (1966) introduced **edit distance** between two strings as the minimum number of elementary operations (insertions, deletions, and substitutions) to transform one string into the other

$d(\mathbf{v}, \mathbf{w})$  = MIN number of elementary operations  
to transform  $\mathbf{v} \rightarrow \mathbf{w}$

# Edit Distance vs Hamming Distance

Hamming distance  
always compares

$i$ -th letter of  $v$  with  
 $i$ -th letter of  $w$

Edit distance  
may compare

$i$ -th letter of  $v$  with  
 $j$ -th letter of  $w$

How to find what  $j$  goes with what  $i$  ???

$v =$   
 $w = \text{TATATATA}$

Make it all line up

$w = \text{TATATATA} -$

Hamming distance:

$$d(v, w) = 8$$

Computing Hamming distance  
is a **trivial** task

Edit distance:

$$d(v, w) = 2$$

Computing edit distance  
is a **non-trivial** task

# Edit Distance: Example

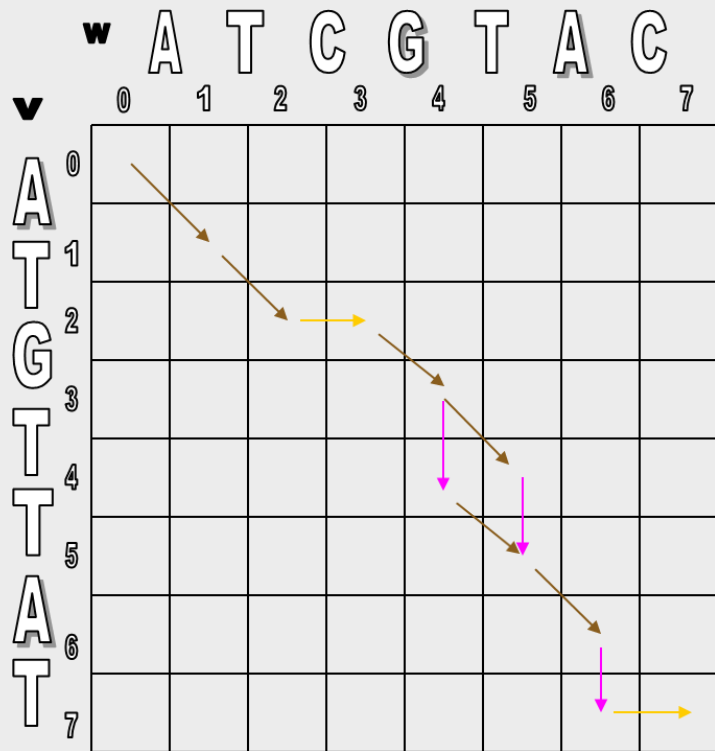
TGCATAT → ATCCGAT in ? steps

TGCATAT → ATCCGAT Max. Score? How?

LCS Problem



# Alignment as a Path in the Edit Graph



012345677	012345677
v= AT_GTTAT_	v= AT_GTTAT_
w= ATCGT_A_C	w= ATCG_TA_C
01234567	01234567

(0,0) , (1,1) , (2,2), (2,3), (3,4),  
 (4,5), (5,5), (6,6), (7,6), (7,7)

# Dynamic Programming Example

		w							
		A	T	C	G	T	A	C	
v	0	0	0	0	0	0	0	0	0
	1	0							
	2	0							
	3	0							
	4	0							
	5	0							
	6	0							
	7	0							

Initialize 1<sup>st</sup> row and 1<sup>st</sup> column to be all zeroes.

Or, to be more precise, initialize 0<sup>th</sup> row and 0<sup>th</sup> column to be all zeroes.

# Dynamic Programming Example

		w							
		A	T	C	G	T	A	C	
v		0	1	2	3	4	5	6	7
A	0	0	0	0	0	0	0	0	0
T	1	0	1	1	1	1	1	1	1
G	2	0	1						
T	3	0	1						
T	4	0	1						
A	5	0	1						
T	6	0	1						
	7	0	1						

$$S_{i,j} = \max \begin{cases} S_{i-1,j-1} & \leftarrow \text{value from NW} + 1, \text{ if } v_i = w_j \\ S_{i-1,j} & \leftarrow \text{value from North (top)} \\ S_{i,j-1} & \leftarrow \text{value from West (left)} \end{cases}$$



Initialize 1<sup>st</sup> row and 1<sup>st</sup> column to be all zeroes.

# Dynamic Programming Example

		w							
			A	T	C	G	T	A	C
v		0	1	2	3	4	5	6	7
	A	0	0	0	0	0	0	0	0
	T	1	0	1	1	1	1	1	1
	G	2	0	1	2	2	2	2	2
	T	3	0	1	2				
	T	4	0	1	2				
	A	5	0	1	2				
	T	6	0	1	2				

Find a match in row and column 2.  
 $i=2, j=2,5$  is a match (T).  
 $j=2, i=2,4,5,7$  is a match (T).

Since  $v_i = w_j$ ,  $s_{i,j} = s_{i-1,j-1} + 1$

$$s_{2,2} = [s_{1,1} = 1] + 1$$

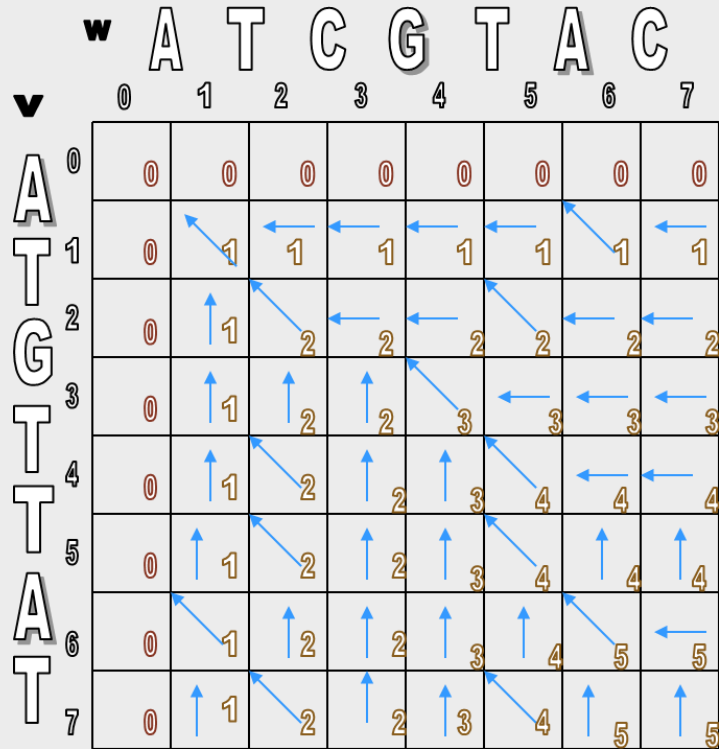
$$s_{2,5} = [s_{1,4} = 1] + 1$$

$$s_{4,2} = [s_{3,1} = 1] + 1$$

$$s_{5,2} = [s_{4,1} = 1] + 1$$

$$s_{7,2} = [s_{6,1} = 1] + 1$$

# Dynamic Programming Example

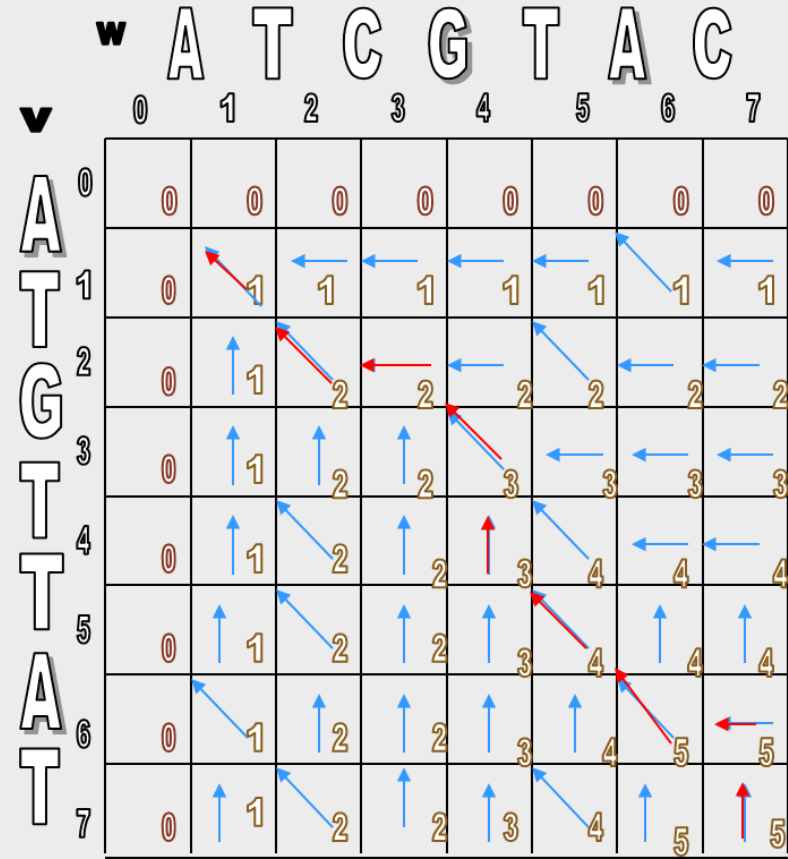


# LCS Algorithm

```
1. LCS(v,w)
2.   for  $i \leftarrow 1$  to  $n$ 
3.      $s_{i,0} \leftarrow 0$ 
4.   for  $j \leftarrow 1$  to  $m$ 
5.      $s_{0,j} \leftarrow 0$ 
6.   for  $i \leftarrow 1$  to  $n$ 
7.     for  $j \leftarrow 1$  to  $m$ 
8.        $s_{i,j} \leftarrow \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1, \text{ if } v_i = w_j \end{cases}$ 
9.        $b_{i,j} \leftarrow \begin{cases} \uparrow & \text{if } s_{i,j} = s_{i-1,j} \\ \leftarrow & \text{if } s_{i,j} = s_{i,j-1} \\ \swarrow & \text{if } s_{i,j} = s_{i-1,j-1} + 1 \end{cases}$ 
10.
11.   return ( $s_{n,m}, b$ )
```

# Now What?

- $LCS(v, w)$  created the alignment grid
- Now we need a way to read the best alignment of  $v$  and  $w$
- Follow the arrows backwards from sink



# Printing LCS: Backtracking

```
1. PrintLCS(b,v,i,j)
2.   if  $i = 0$  or  $j = 0$ 
3.     return
4.   if  $b_{i,j} = "$  ↖  $"$ 
5.     PrintLCS(b,v,i-1,j-1)
6.     print  $v_i$ 
7.   else
8.     if  $b_{i,j} = "$  ↑  $"$ 
9.       PrintLCS(b,v,i-1,j)
10.    else
11.      PrintLCS(b,v,i,j-1)
```



# LCS Runtime

- It takes  $O(nm)$  time to fill in the  $n \times m$  dynamic programming matrix.
- Why  $O(nm)$ ? The pseudocode consists of a nested “for” loop inside of another “for” loop to set up a  $n \times m$  matrix.

# Local vs. Global Alignment

- The Global Alignment Problem tries to find the longest path between vertices  $(0,0)$  and  $(n,m)$  in the edit graph.
- The Local Alignment Problem tries to find the longest path among paths between arbitrary vertices  $(i,j)$  and  $(i', j')$  in the edit graph.
- For example, positions 20-40 of sequence A might be aligned with positions 50-70 of sequence B

Global

GTAGGCTTAAGGTTA  
-TAG----A---T-A

Local

GTAGGCTTAAGGTTA  
TAGATA

# Local vs. Global Alignment

- Global Alignment

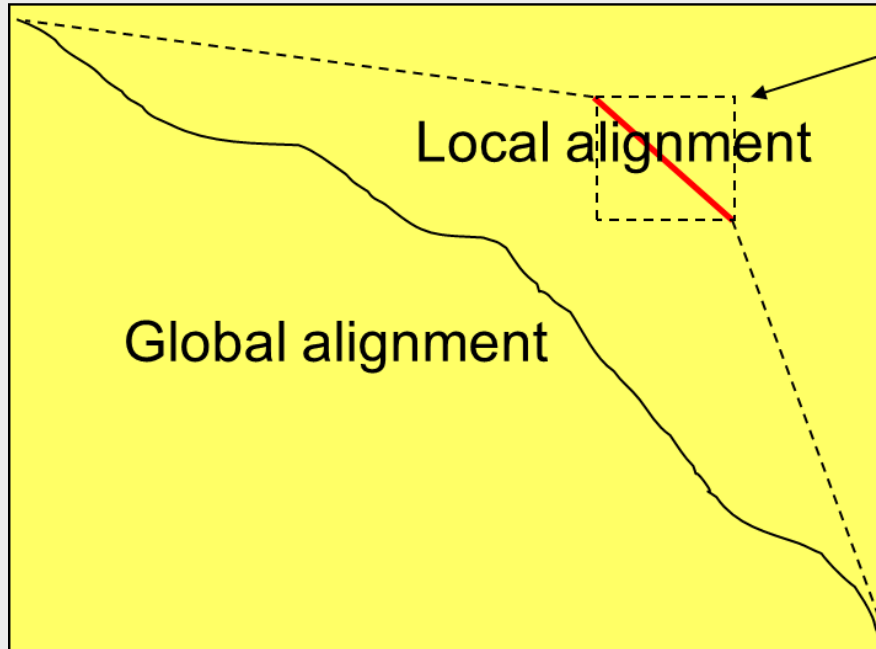
```
--T--CC-C-AGT--TATGT-CAGGGGACACG-A-GCATGCAGA-GAC
|  || |  ||  |  ||  |  |  |||  ||  |  |  |  ||||  |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG-T-CAGAT--C
```

- Local Alignment—better alignment to find conserved segment

```
          tccCAGTTATGTCAGgggacacgagcatgcagagac
          |||||
aattgccgccgctcgttttcagCAGTTATGTCAGatc
```

Sample: Homeobox genes have a short region called the *homeodomain* that is highly conserved between species. A global alignment would not find them!

# Local vs. Global Alignment



Compute a "mini"  
Global Alignment to  
get Local

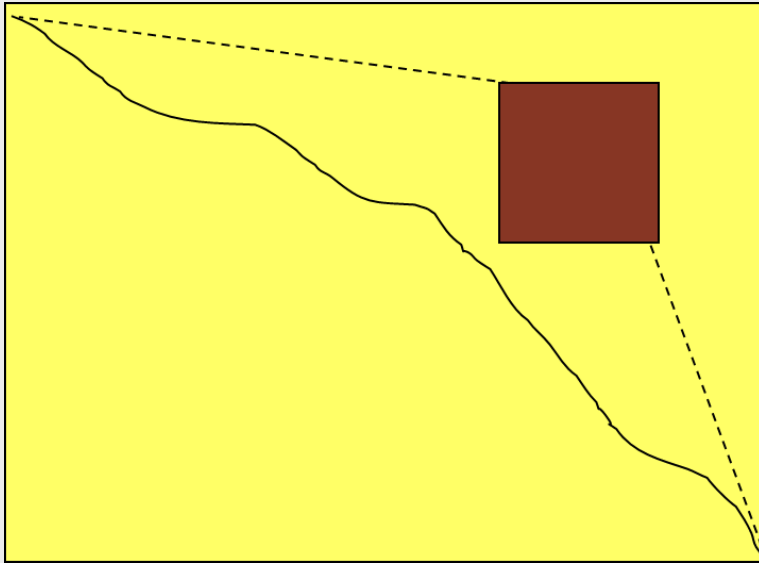
# The Local Alignment Problem

Goal: *Find the best local alignment between two strings*

- Input : Strings **v**, **w** and scoring matrix  $\delta$
- Output : Alignment of substrings of **v** and **w** whose alignment score is maximum among all possible alignment of all possible substrings

**$O(?)$**

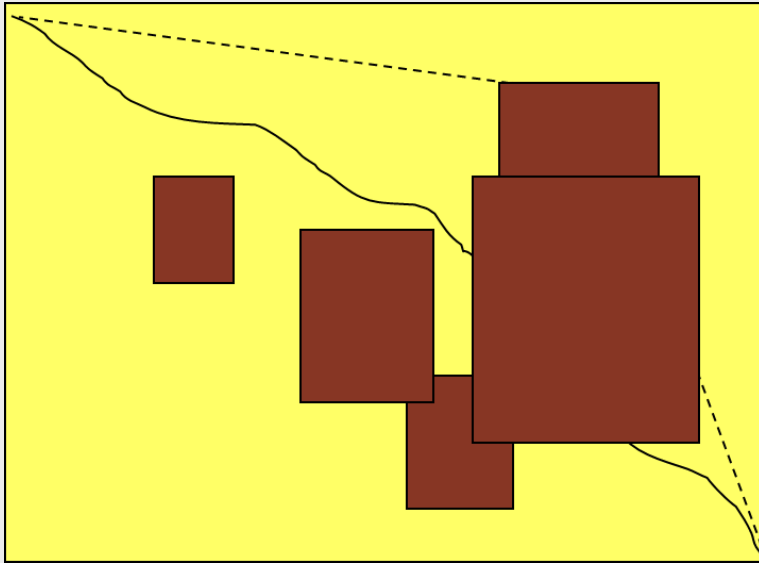
# Local Alignment: Running Time



- Long run time  $O(n^4)$ :
  - In the grid of size  $n \times n$  there are  $\sim n^2$  vertices  $(i, j)$  that may serve as a source.
  - For each such vertex computing alignments from  $(i, j)$  to  $(i', j')$  takes  $O(n^2)$  time.

How to solve this problem?

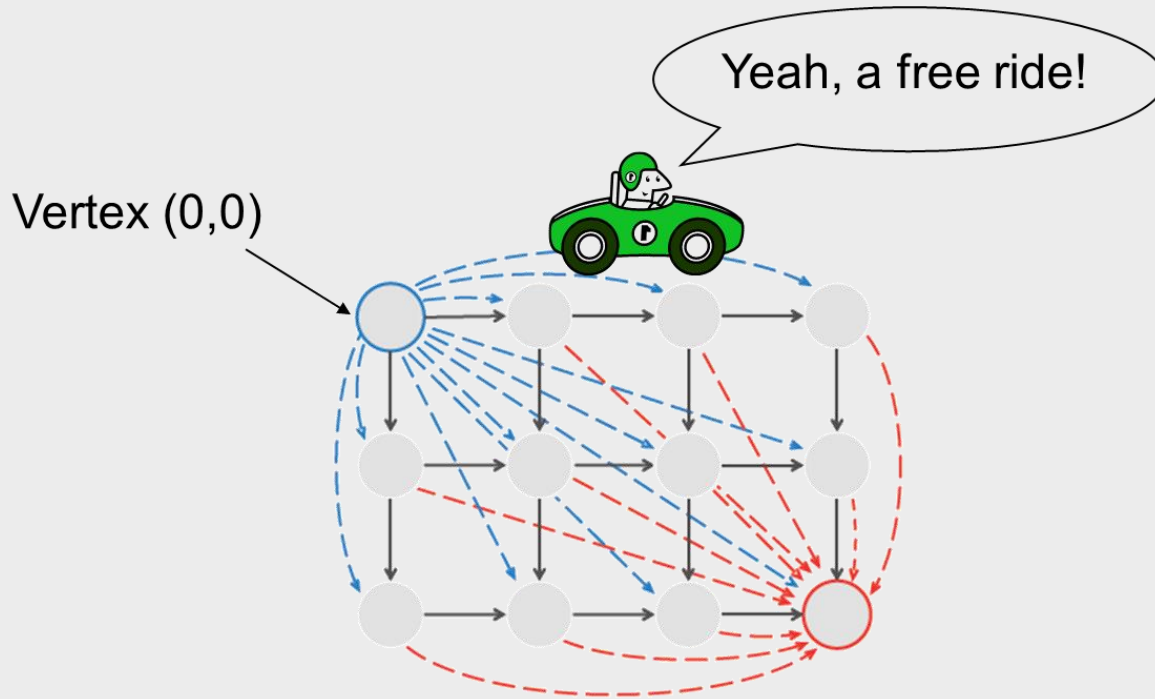
# Local Alignment: Running Time



- Long run time  $O(n^4)$ :
  - In the grid of size  $n \times n$  there are  $\sim n^2$  vertices  $(i,j)$  that may serve as a source.
  - For each such vertex computing alignments from  $(i,j)$  to  $(i',j')$  takes  $O(n^2)$  time.

How to solve this problem?

# Local Alignment: Running Time



The dashed edges represent the free rides from (0,0) to every other node.



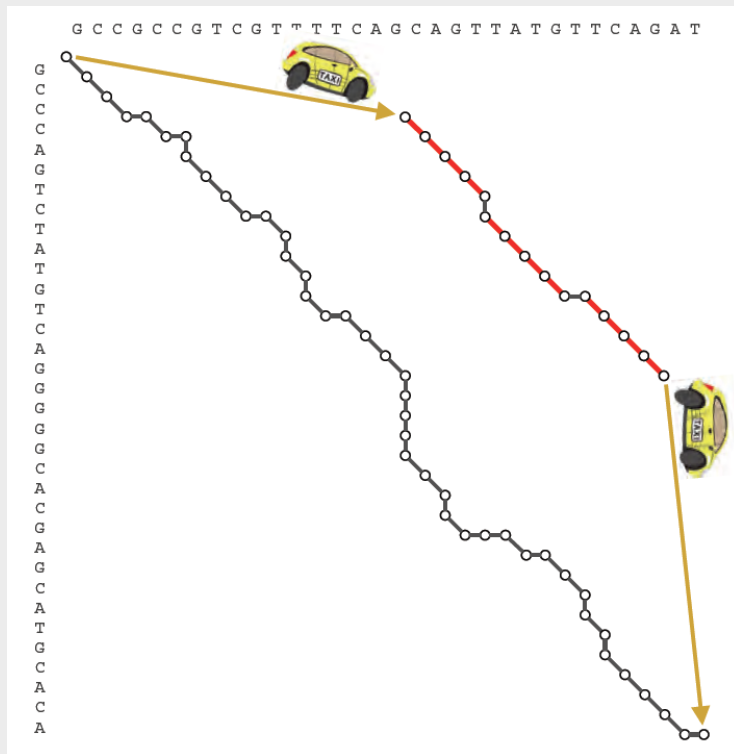
# Local Alignment: Running Time

Adding zero-weight edges from (0, 0) to every node has made the source node (0, 0) a predecessor of every node (i, j).

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j} + \text{Score}(v_i, -) \\ s_{i,j-1} + \text{Score}(-, w_j) \\ s_{i-1,j-1} + \text{Score}(v_i, w_j) \end{cases}$$

Also, because node (n,m) now has every other node as a predecessor,  $s_{n,m}$  will be equal to the largest value of  $s_{i,j}$  over the entire alignment graph.

# Local Alignment: Running Time



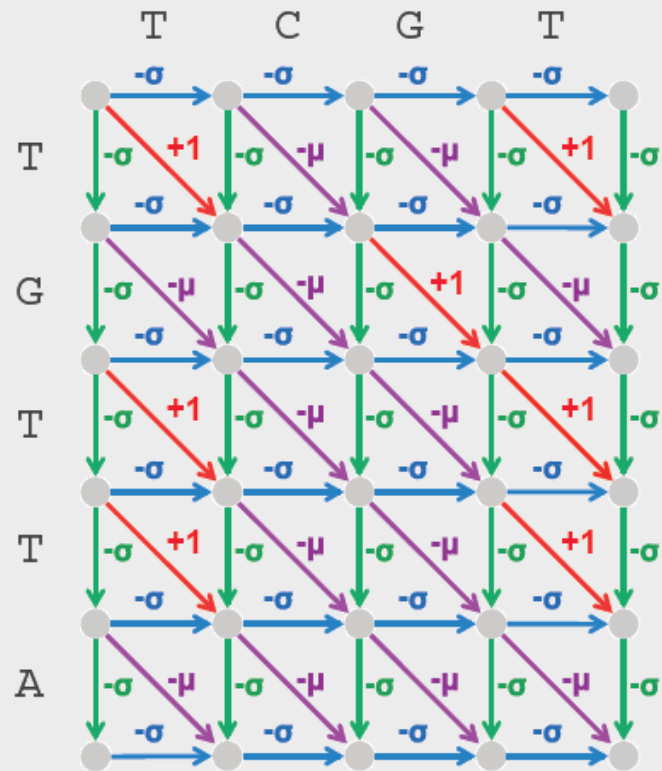
# From LCS to Alignment (Change up the Scoring)

- The Longest Common Subsequence (LCS) problem—the simplest form of sequence alignment – allows only insertions and deletions (no mismatches).
- In the LCS Problem, we scored 1 for matches and 0 for indels
- Consider penalizing indels and mismatches with negative scores
- Simplest *scoring schema*:
  - +1 : **match premium**
  - $\mu$  : **mismatch penalty**
  - $\sigma$  : **indel penalty**

# From LCS to Alignment (Change up the Scoring)

$$s_{i,j} = \max \begin{cases} 0 & \text{for local alignment} \\ s_{i-1,j} - \sigma \\ s_{i,j-1} - \sigma \\ s_{i-1,j-1} + 1, & \text{if } v_i = w_j \\ s_{i-1,j-1} - \mu, & \text{if } v_i \neq w_j. \end{cases}$$

$$\text{Score} = \# \text{matches} - \mu(\# \text{mismatches}) - \sigma(\# \text{indels})$$



# Scoring Matrices

To generalize scoring, consider a  $(4+1) \times (4+1)$  **scoring matrix**  $\delta$ .

In the case of an amino acid sequence alignment, the scoring matrix would be a  $(20+1) \times (20+1)$  size. The addition of 1 is to include the score for comparison of a gap character “-”.

This will simplify the algorithm as follows:

$$s_{i,j} = \max \begin{cases} s_{i-1,j-1} + \delta(v_i, w_j) \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \end{cases}$$

# Making a Scoring Matrix

- Scoring matrices are created based on biological evidence.
- Alignments can be thought of as two sequences that differ due to mutations.
- Some of these mutations have little effect on the protein's function, therefore some penalties,  $\delta(v_i, w_j)$ , will be less harsh than others.

**PAM**

**BLOSUM**

# Scoring Matrix: Example

	A	R	N	K
A	5	-2	-1	-1
R	-	7	-1	3
N	-	-	7	0
K	-	-	-	6

**Seq. 1** AKRANR

**Seq.2** KAAANK  
 $-1 + (-1) + (-2) + 5 + 7 + 3 = 11$

- Notice that although R and K are different amino acids, they have a positive score.
- Why? They are both positively charged amino acids → will not greatly change function of protein.

# Gap Penalties

- In nature, a series of k indels often come as a single event rather than a series of k single nucleotide events:

ATA\_\_GC  
ATATTGC

This is more likely.

Normal scoring would  
give the same score for  
both alignments

ATAG\_GC  
AT\_GTGC

This is less likely.



# Gap Penalties

- *Gaps*- contiguous sequence of spaces in one of the rows
- Score for a gap of length  $x$  is:

$$-(\rho + \sigma(x-1))$$

where  $\rho > 0$  is the penalty for introducing a gap:

gap opening penalty

$\rho$  will be large relative to  $\sigma$ :

gap extension penalty

- because you do not want to add too much of a penalty for extending the gap.

# Scoring Insertions and Deletions

match = 1  
mismatch = 0

Total Score: 4

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

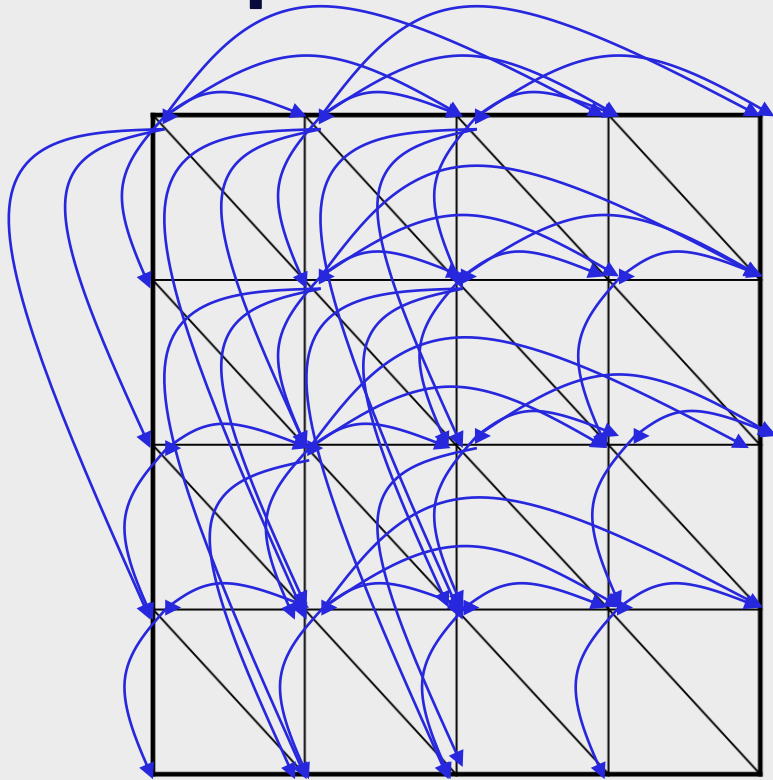
match = 1  
mismatch = 0  
 $\rho = 3$  (gap opening)  
 $\sigma = 0.1$  (gap extension)  
 $x = 3$  (gap length)  
 $\gamma(g) = -3 - (3 - 1) 0.1 = -3.2$

Total Score:  $8 - 3.2 = 4.8$

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

insertion / deletion

# Gap Penalties and Edit Graph

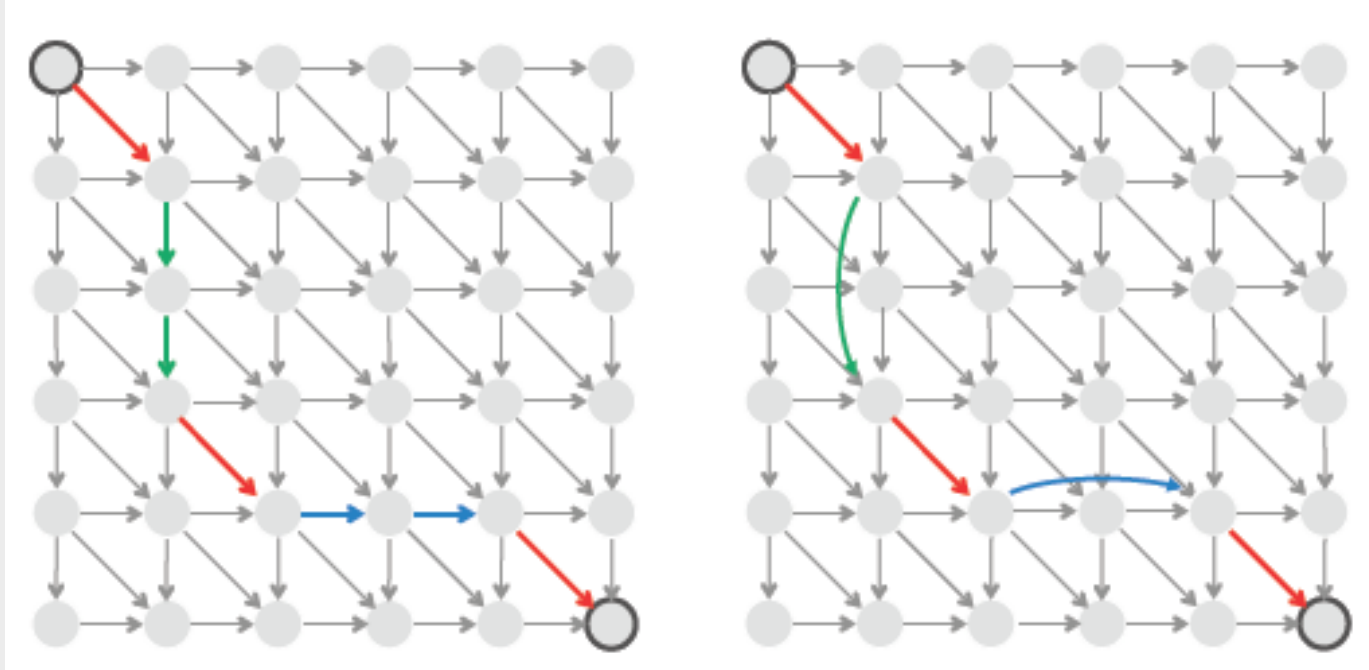


To reflect affine gap penalties we have to add “long” horizontal and vertical edges to the edit graph. Each such edge of length  $x$  should have weight

$$-\rho - (x-1) * \sigma$$

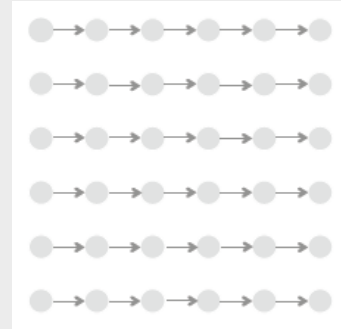
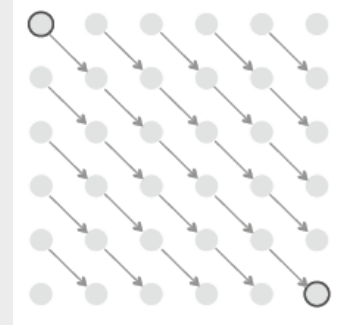
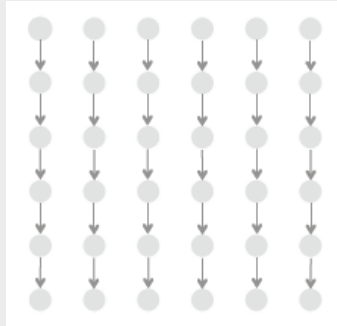
So the complexity increases from  $O(n^2)$  to  $O(n^3)$

# Gap Penalties and Edit Graph

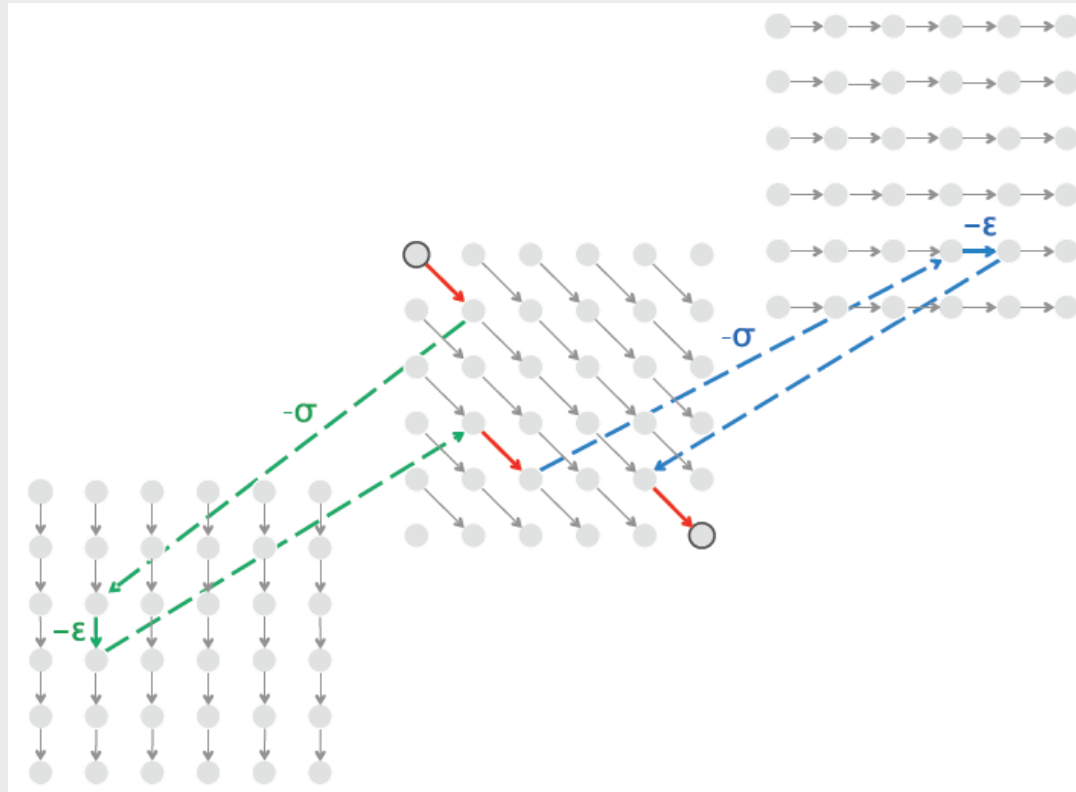


# Gap Penalties and 3 Layer Manhattan Grid

- The **top level** creates/extends gaps in the sequence  $w$ .
- The **middle level** extends matches and mismatches.
- The **bottom level** creates/extends gaps in sequence  $v$ .



# Gap Penalties and 3 Layer Manhattan Grid



# Switching between 3 Layers

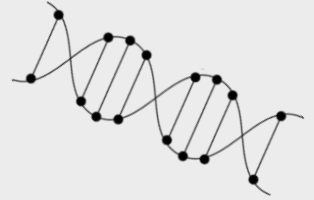
- -  $\sigma$  : Jumping penalty is assigned to moving from the main level to either the upper level or the lower level
- -  $\epsilon$  : Gap extension penalty for each continuation on a level other than the main level)

$$\text{lower}_{i,j} = \max \begin{cases} \text{lower}_{i-1,j} - \epsilon \\ \text{middle}_{i-1,j} - \sigma \end{cases}$$

$$\text{middle}_{i,j} = \max \begin{cases} \text{lower}_{i,j} \\ \text{middle}_{i-1,j-1} + \text{Score}(v_i, w_j) \\ \text{upper}_{i,j} \end{cases}$$

$$\text{upper}_{i,j} = \max \begin{cases} \text{upper}_{i,j-1} - \epsilon \\ \text{middle}_{i,j-1} - \sigma \end{cases}$$

# Resources



[1] Bioinformatics Algorithms: An Active Learning Approach, P. Compeau, and P. Pevzner. Active Learning Publishers, 2nd Ed. Vol. 2, (2015)