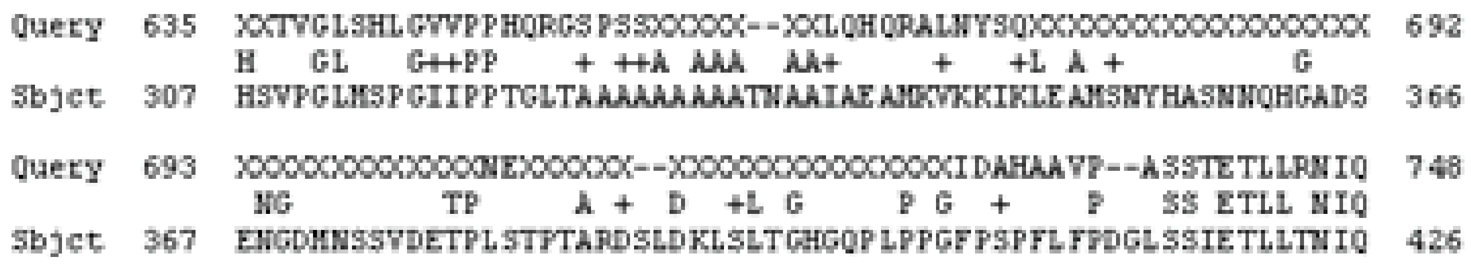
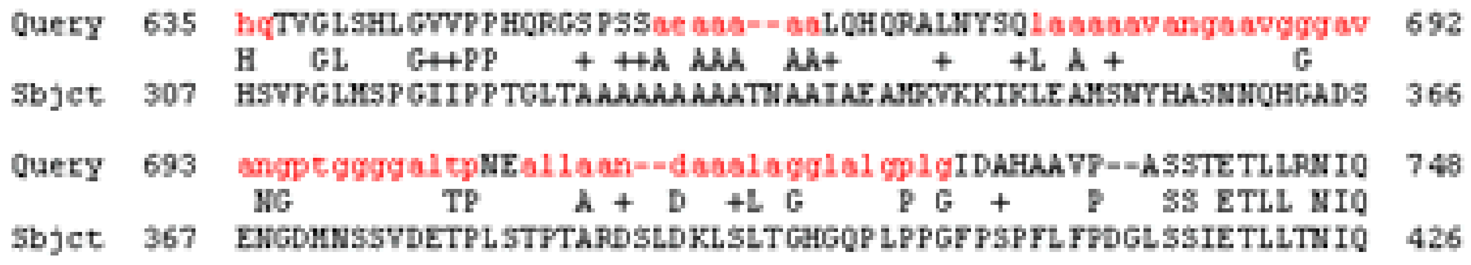
# EECS730: Introduction to Bioinformatics

## Lecture 03: Edit distance and sequence alignment



```
Query  635  XXTVGLSHLGVVPPHQRGSPSSXXXXX--XXLQHQRALNYSQXXXXXXXXXXXXXXXXXX  692
            H   GL   G++PP    + ++A AAA  AA+     +   +L A +         G
Sbjct  307  HSVPGLMSPGIIPPTGLTAAAAAAAAATNAAIAEAMKVKKIKLEAMSNYHASNNQHGADS  366

Query  693  XXXXXXXXXXXXXNEXXXXXX--XXXXXXXXXXXXXXXIDAHAAVP--ASSTETLLRNIQ  748
             NG        TP     A +  D  +L G     P G  +    P   SS ETLL NIQ
Sbjct  367  ENGDMNSSVDETPLSTPTARDSLDKLSLTGHGQPLPPGFPSPFLFPDGLSSIETLLTNIQ  426
```



```
Query  635  hqTVGLSHLGVVPPHQRGSPSSaeaaa--aaLQHQRALNYSQlaaaaavangaavgggav  692
            H   GL   G++PP    + ++A AAA  AA+     +   +L A +         G
Sbjct  307  HSVPGLMSPGIIPPTGLTAAAAAAAAATNAAIAEAMKVKKIKLEAMSNYHASNNQHGADS  366

Query  693  angptggggaltpNEallaan--daaalagglalgplgIDAHAAVP--ASSTETLLRNIQ  748
             NG        TP     A +  D  +L G     P G  +    P   SS ETLL NIQ
Sbjct  367  ENGDMNSSVDETPLSTPTARDSLDKLSLTGHGQPLPPGFPSPFLFPDGLSSIETLLTNIQ  426
```

Slides adapted from Dr. Shaojie Zhang (University of Central Florida)

# KUMC visit

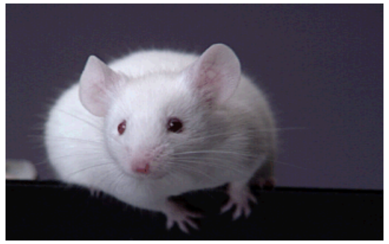
- How many of you would like to attend my talk on metagenomics?

## DNA Sequence Comparison: First Success Story

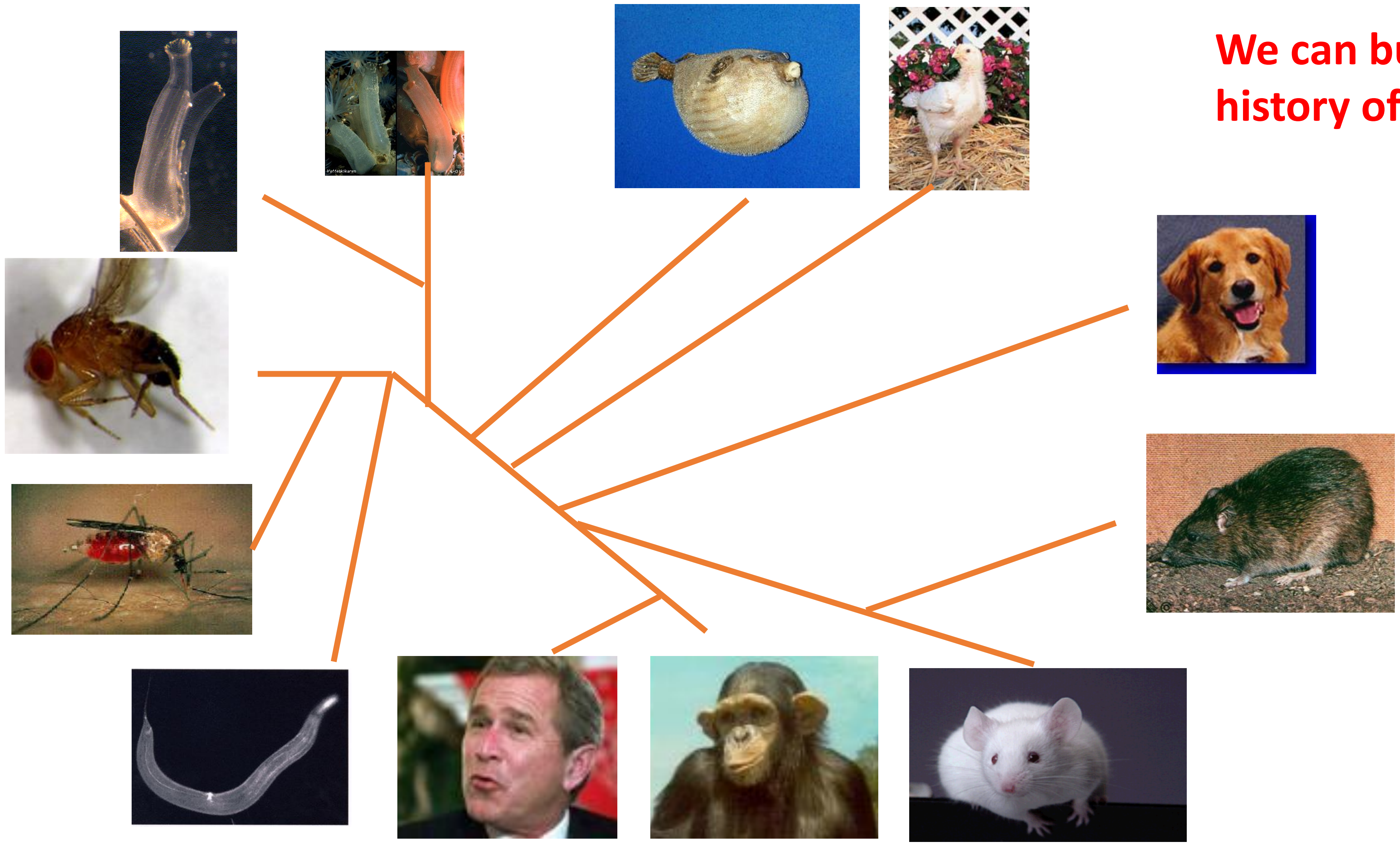
- Finding sequence similarities with genes of known function is a common approach to infer a newly sequenced gene's function
- In 1984 Russell Doolittle and colleagues found similarities between cancer-causing gene (**v-**sys in *Simian Sarcoma* Virus) and normal growth factor (PDGF) gene
- Cancer is caused by normal growth gene being switched on at a wrong time

**The human genome is not the most complex genome!!!**

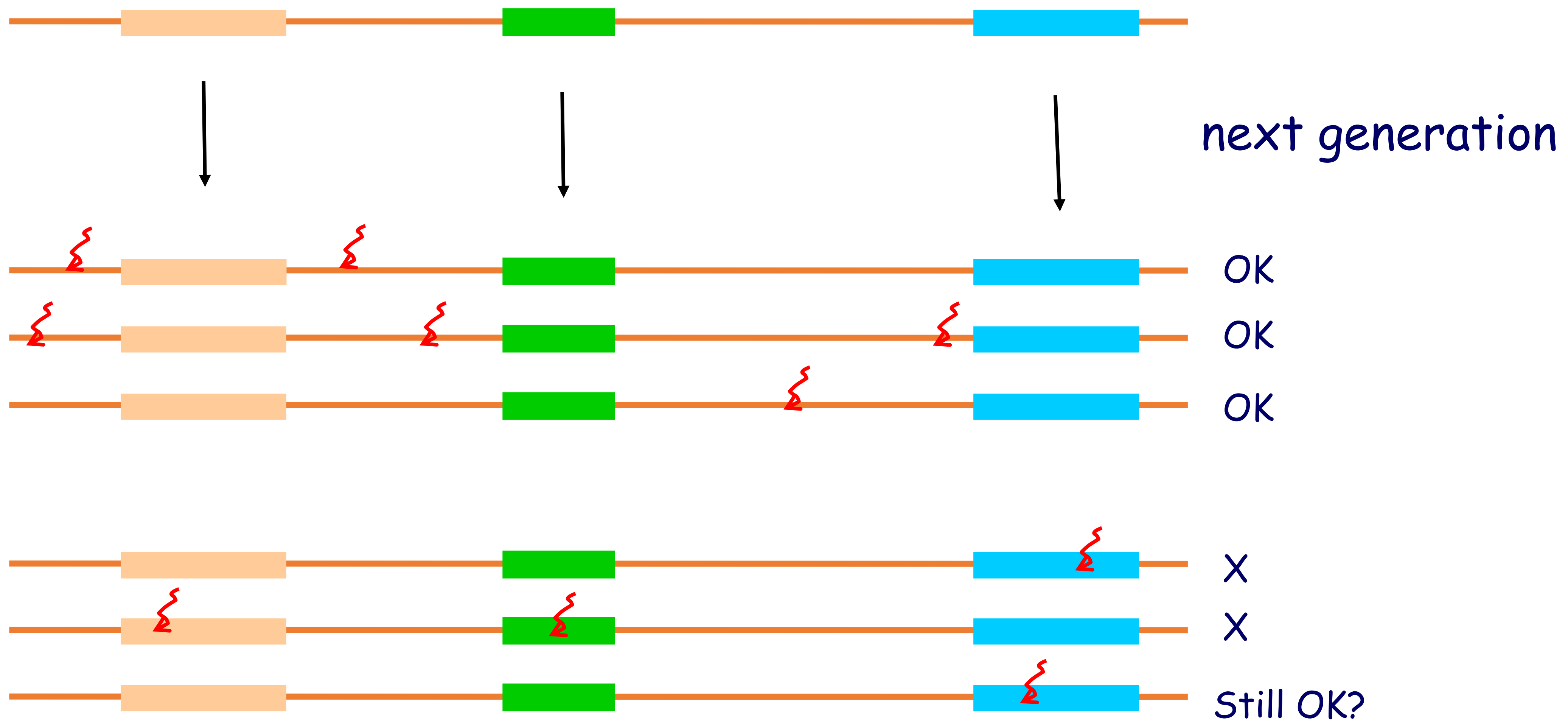
nearly 200 complete genomes have been sequenced



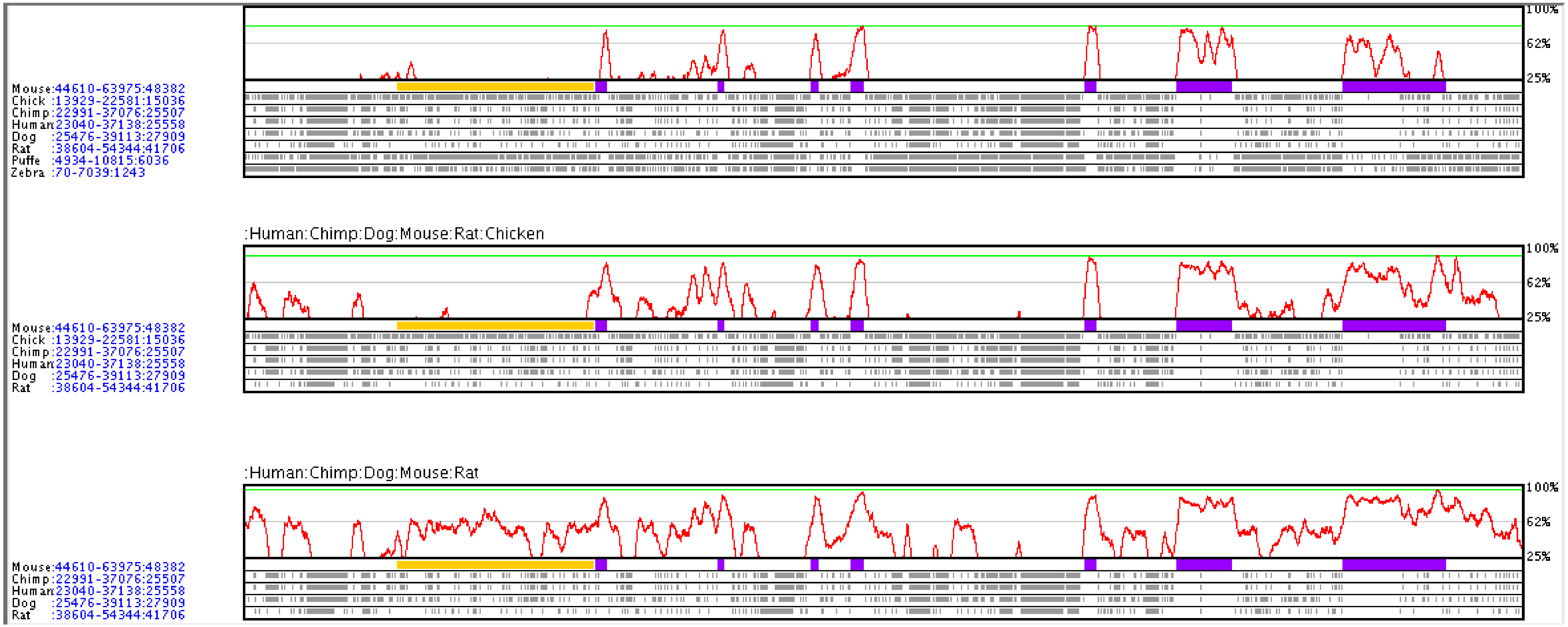
We can build the evolution history of these species



# Evolutionary Rates



# Sequence conservation implies **important** function



## Sequence similarity

- Similar genes sequences will code for similar protein sequences
- Similar protein sequences should adopt similar folds (3D structures)
- Similar 3D structures imply similar functions

- Similar gene sequences may origin from the same ancestor and can provide information in evolution inference

- How do we quantify the sequence similarity???

## Hamming distance?

```
V : ATATATAT
W : TATATATA
```

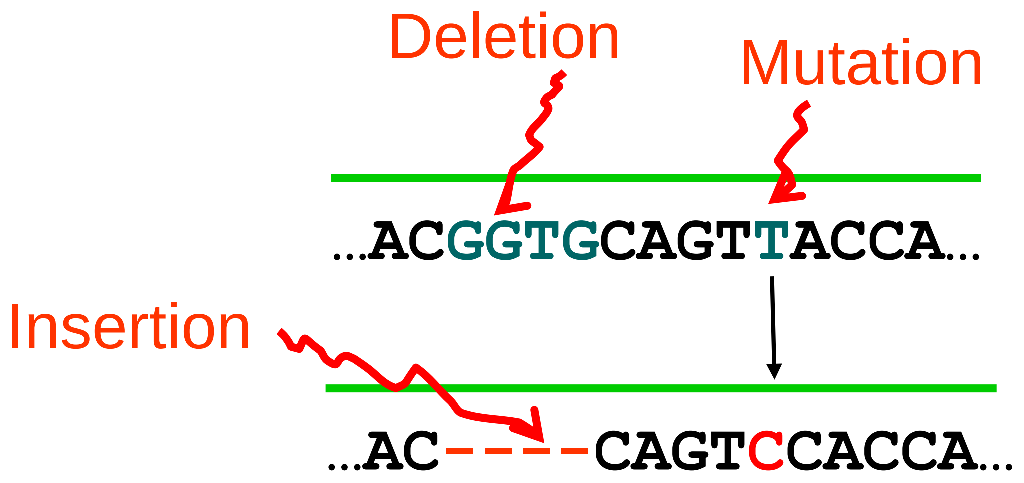
**Hamming distance**

```
V : ATATATAT--
W : --TATATATA
```

**Alignment distance**

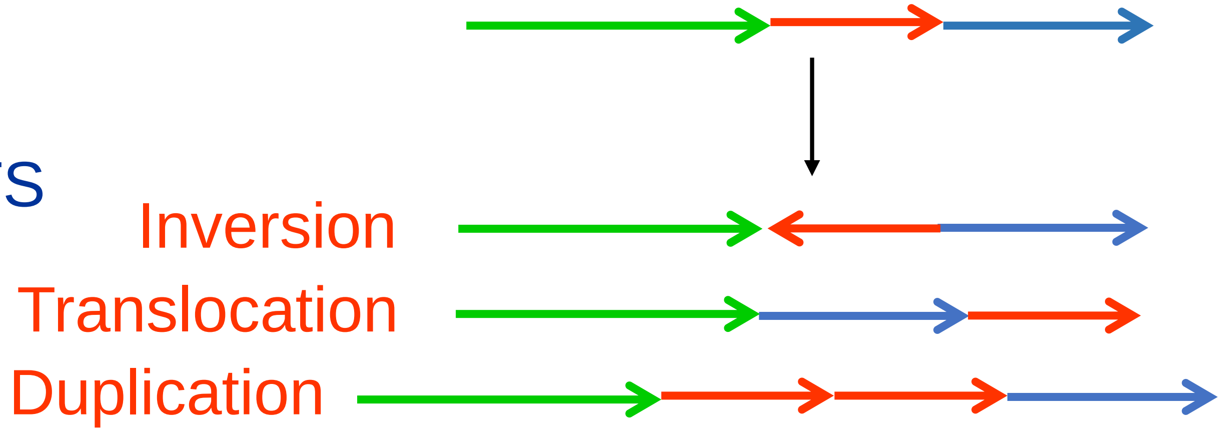
**Hamming distance underestimate the similarity of two strings, more sophisticated algorithm is needed!**

# Evolution at the DNA level



SEQUENCE EDITS

REARRANGEMENTS



# Sequence alignment

AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTTGCCCGAC

-**AG**G**CTATCAC**CT**GACC**T**C**CA**GG**C**CGA**--**TGCCC**---
T**AG**-**CTATCAC**--**GACC**G**C**--**GG**T**CGA**TT**TGCCC**GAC

**Definition**

Given two strings $x = x_1x_2...x_M$, $y = y_1y_2\cdots y_N$,

an alignment is an assignment of gaps to positions 0,···, M in x, and 0,···, N in y, so as to line up each letter in one sequence with either a letter, or a gap in the other sequence

## Sequence alignment cont.

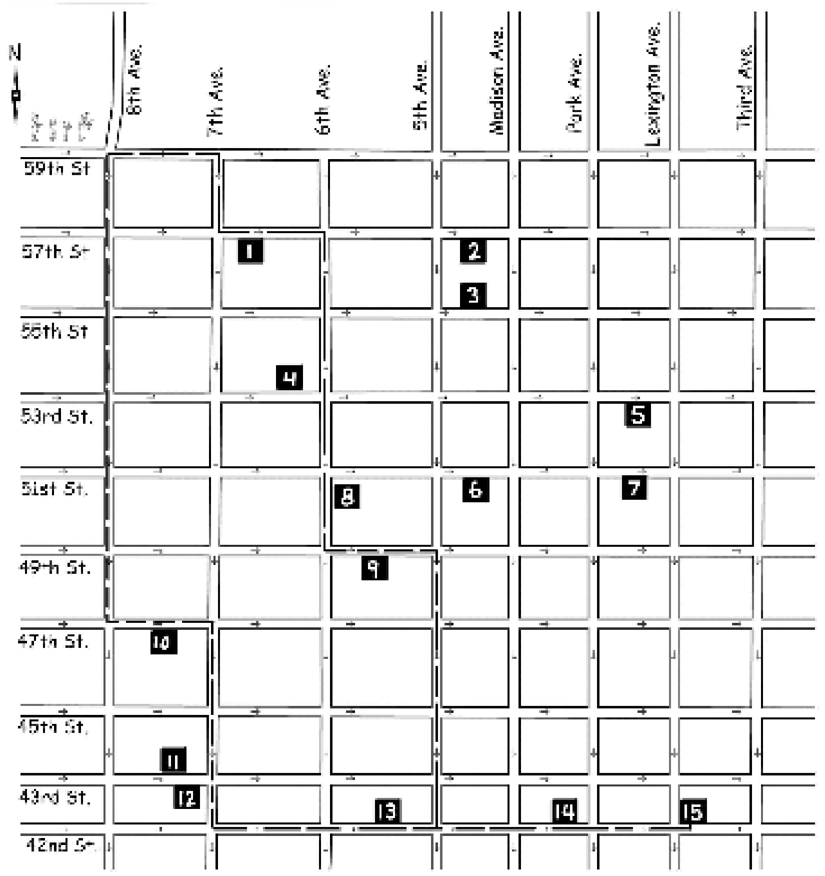
AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTTGCCCGAC

-**AG**G**CTATCAC**CT**GACC**T**C**CA**GG**C**CGA**--**TGCCC**---
T**AG**-**CTATCAC**--**GACC**G**C**--**GG**T**CGA**TT**TGCCC**GAC

**What is the object function??? (and quantitative measure)**

# The Manhattan Tourist problem

- Computing similarity is detail-oriented, and we need to do some preliminary work first:
  - The Manhattan Tourist Problem introduces grids, graphs and edit graphs

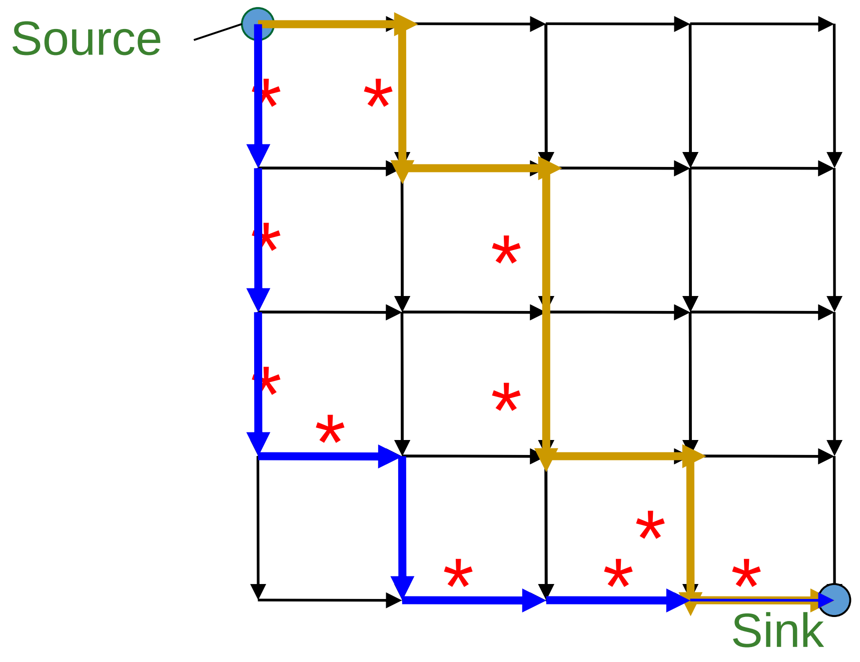


1 Carnegie Hall
2 Tiffany & Co.
3 Sony Building
4 Museum of Modern Art
5 Four Seasons
6 St. Patrick's Cathedral
7 General Electric Building
8 Radio City Music Hall
9 The Today Show
10 Paramount Building
11 NY Times Building
12 Times Square
13 General Society of Mechanics and Tradesmen (a must see!)
14 Grand Central Terminal
15 Chrysler Building

**See the most stuff in the least time.**

# Manhattan Tourist Problem (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

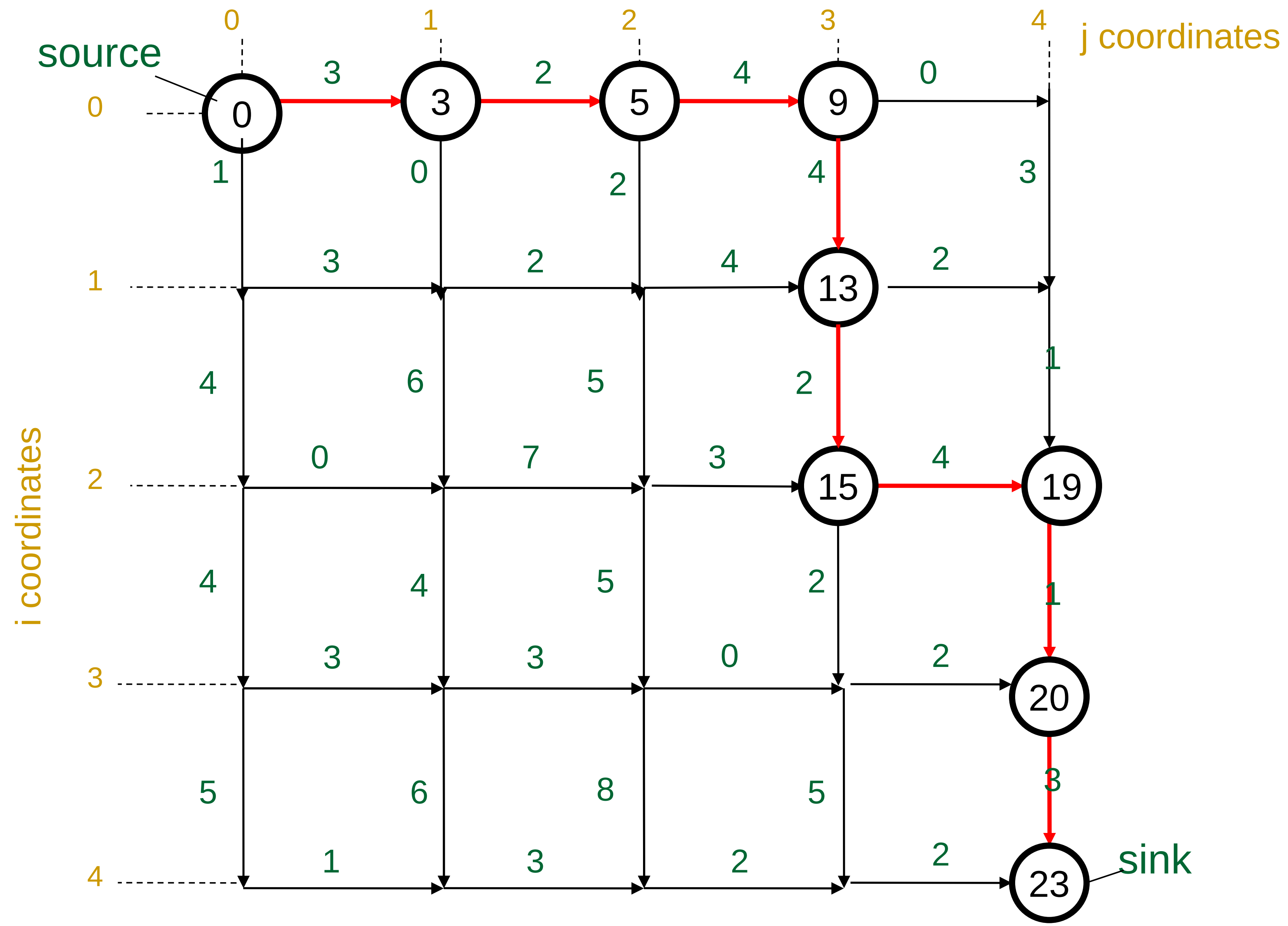


## MTP formulation

Goal: Find the longest path in a weighted grid.

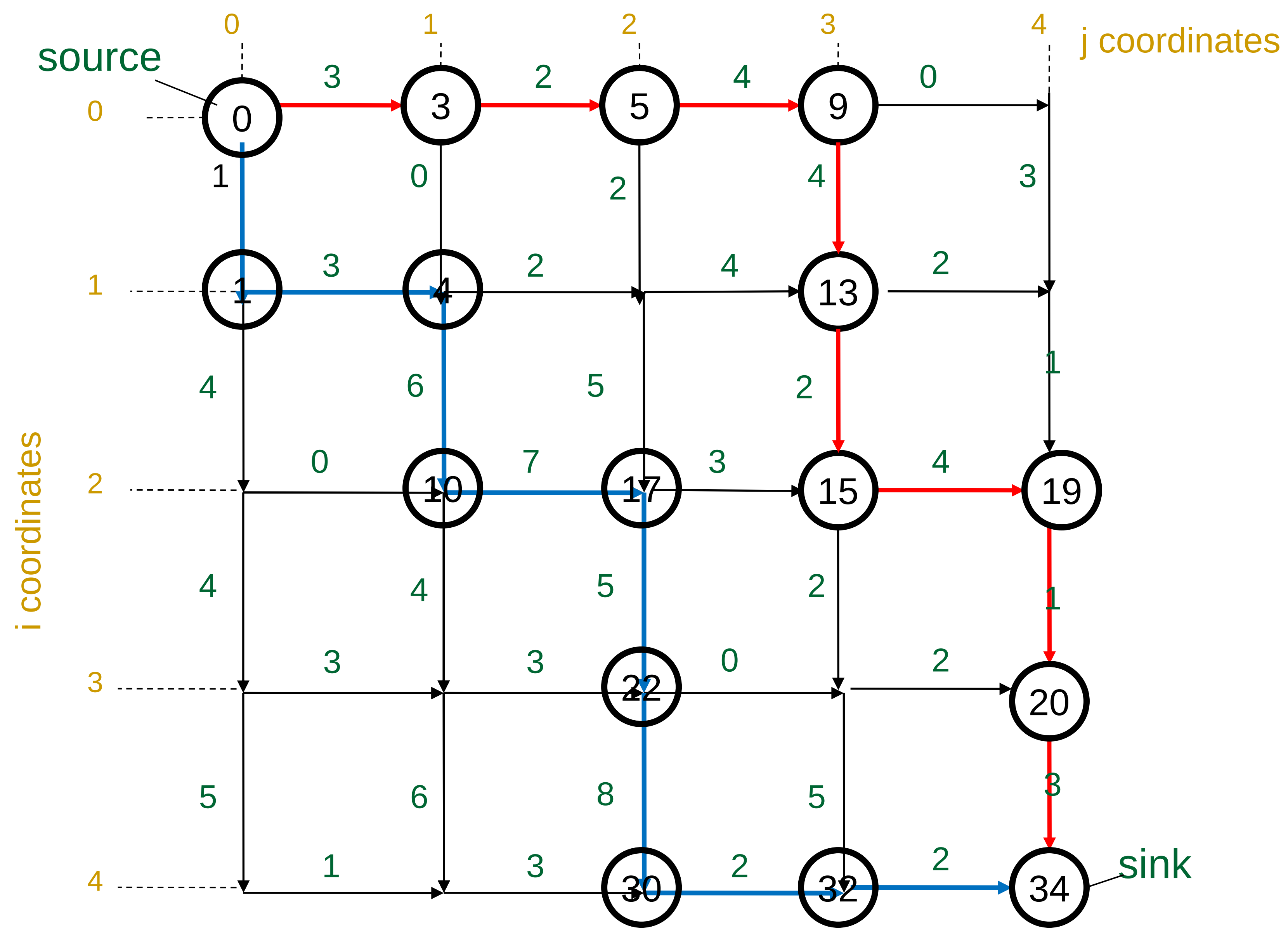
Input: A weighted grid **G** with two distinct vertices, one labeled “source” and the other labeled “sink”

Output: A longest path in **G** from “source” to “sink”



# MTP example 1

# MTP example 2

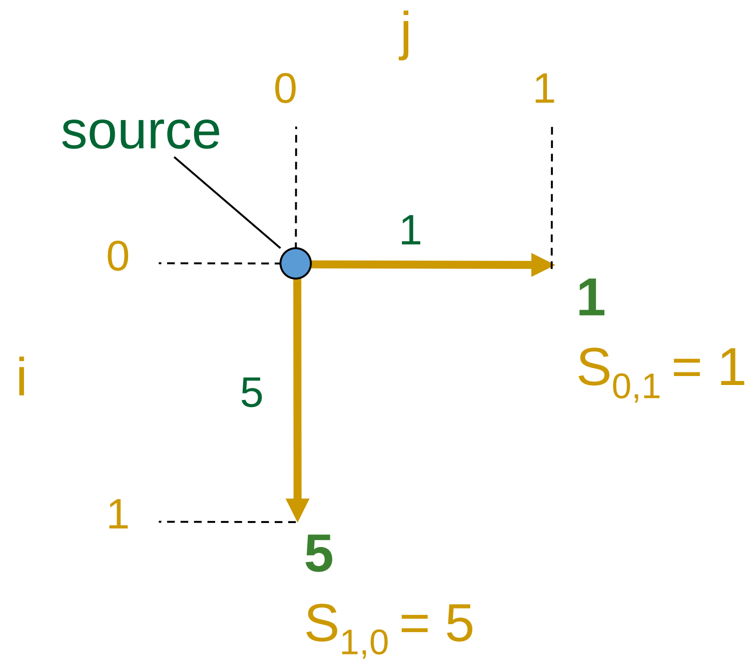


## Simple recursion

```
MT(n,m)
   x ← MT(n-1,m)+
              length of the edge from (n- 1,m) to (n,m)
   y ← MT(n,m-1)+
              length of the edge from (n,m-1) to (n,m)
   return max{x,y}
```

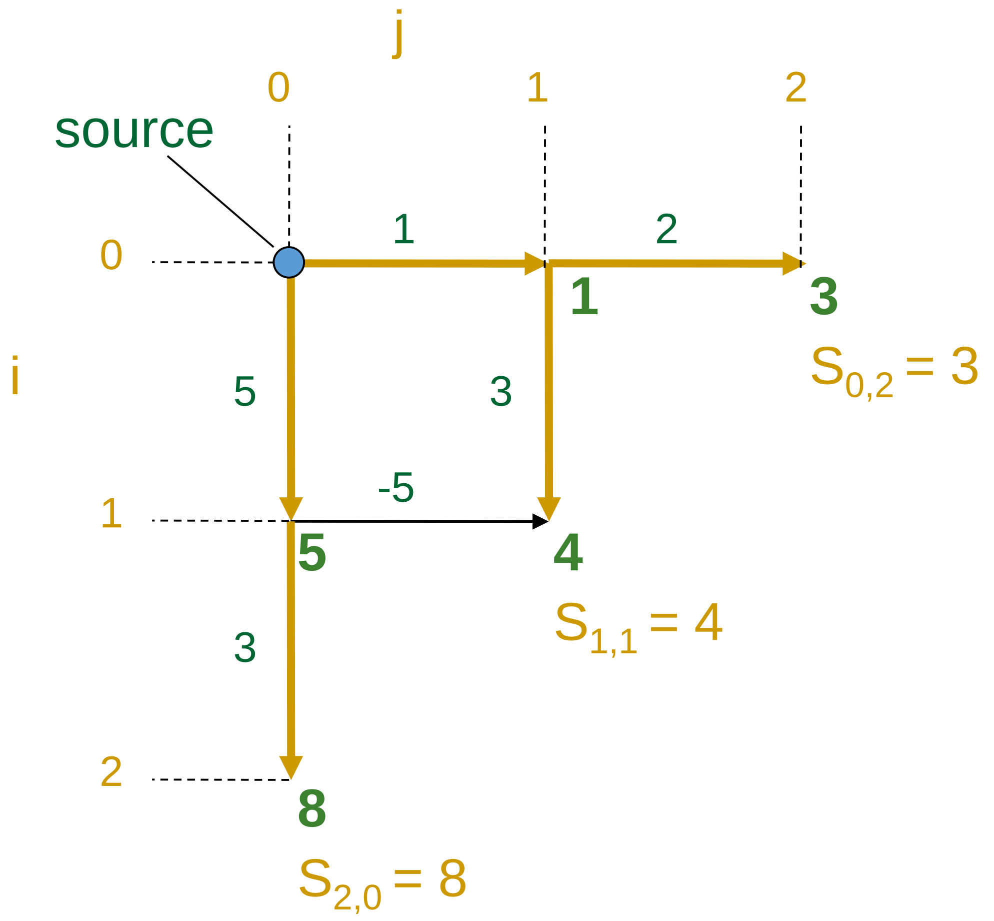
**Slow!!! For the same reason that RecursiveChange is slow**

## MTP: Dynamic Programming

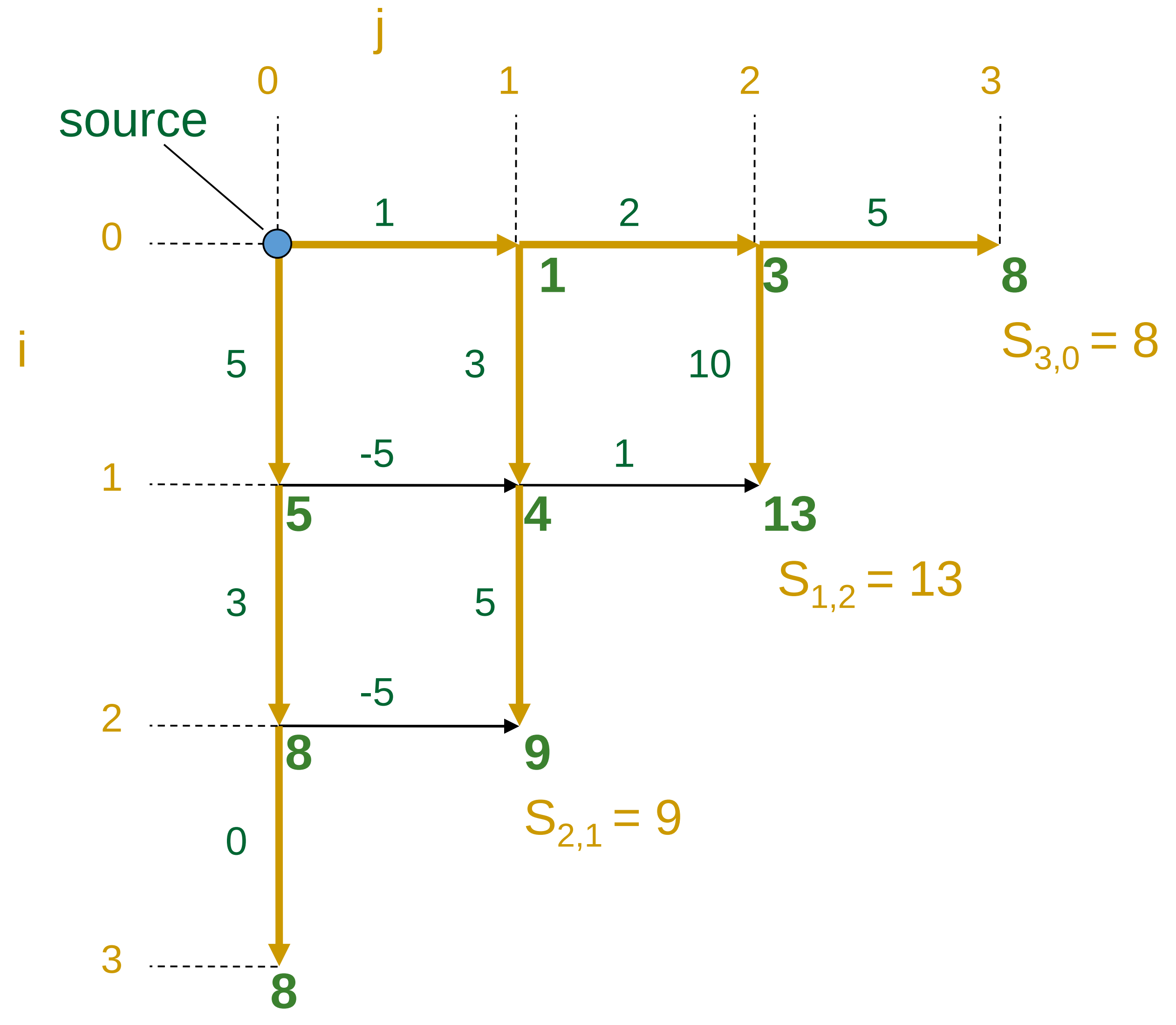


- Instead of recursion, store the result in an array **S**

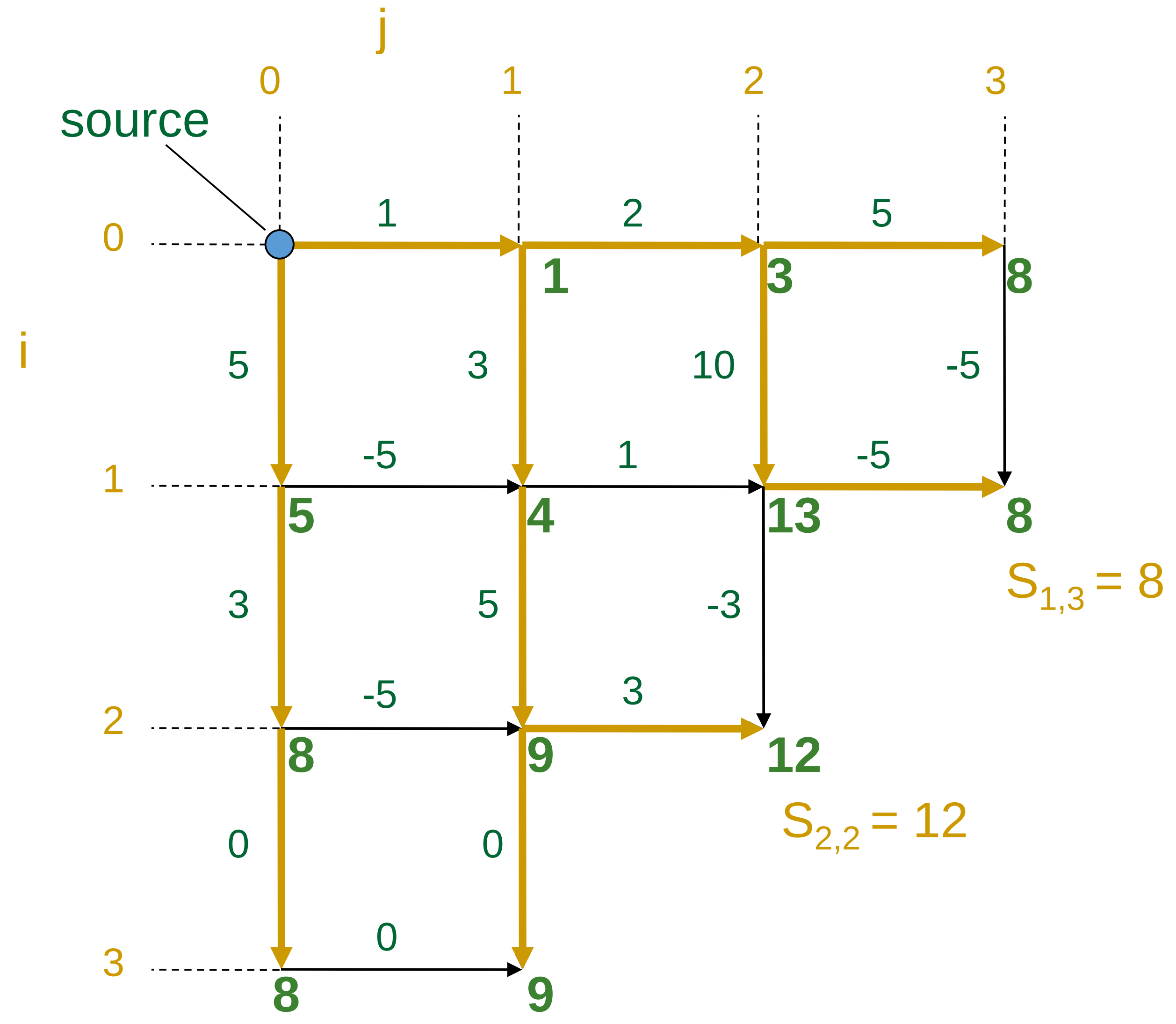
# MTP: Dynamic Programming cont.



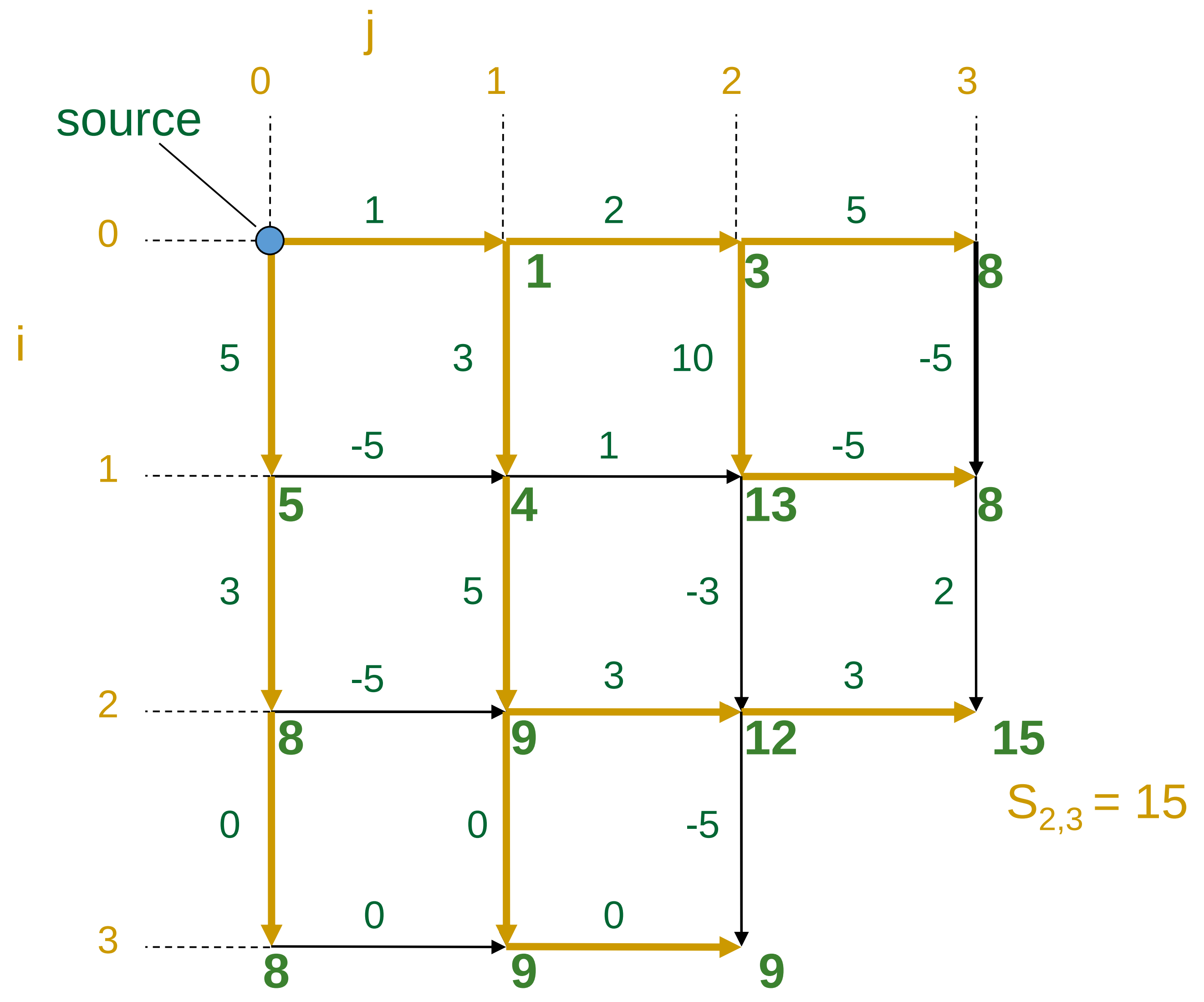
# MTP: Dynamic Programming cont.



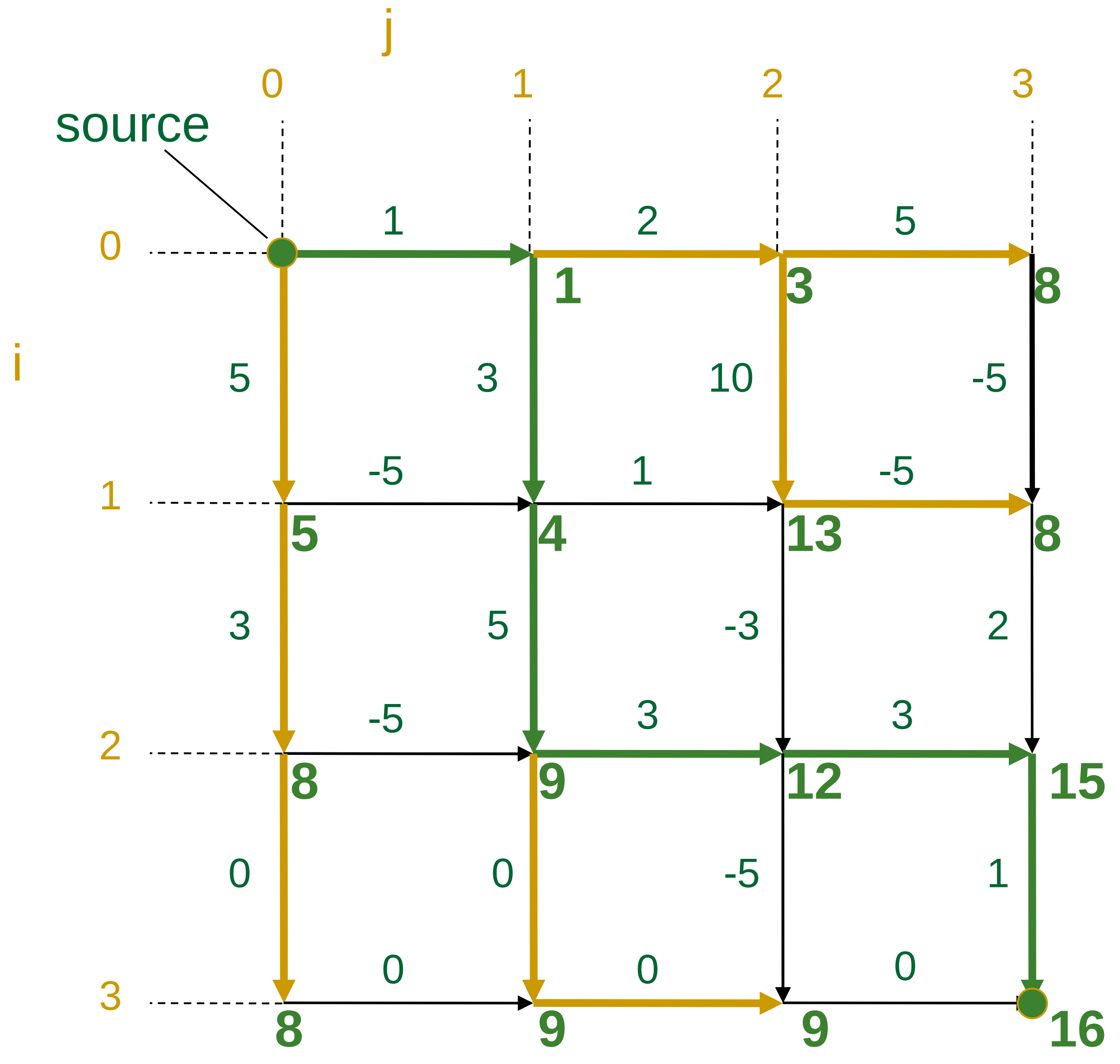
# MTP: Dynamic Programming cont.



# MTP: Dynamic Programming cont.



## MTP: Dynamic Programming cont.



**Done!**

**(showing all back-traces)**

# MTP: recurrence function

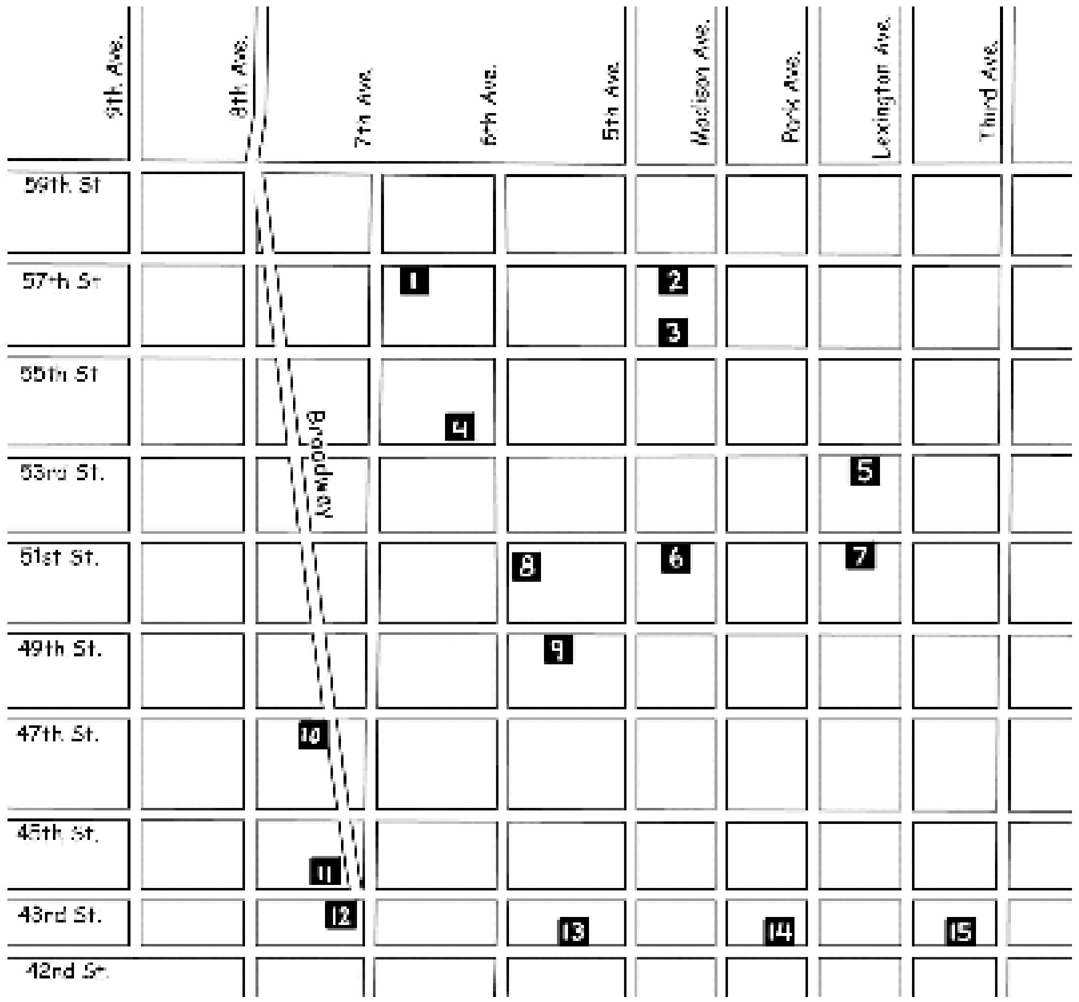
Computing the score for a point (i,j) by the recurrence relation:

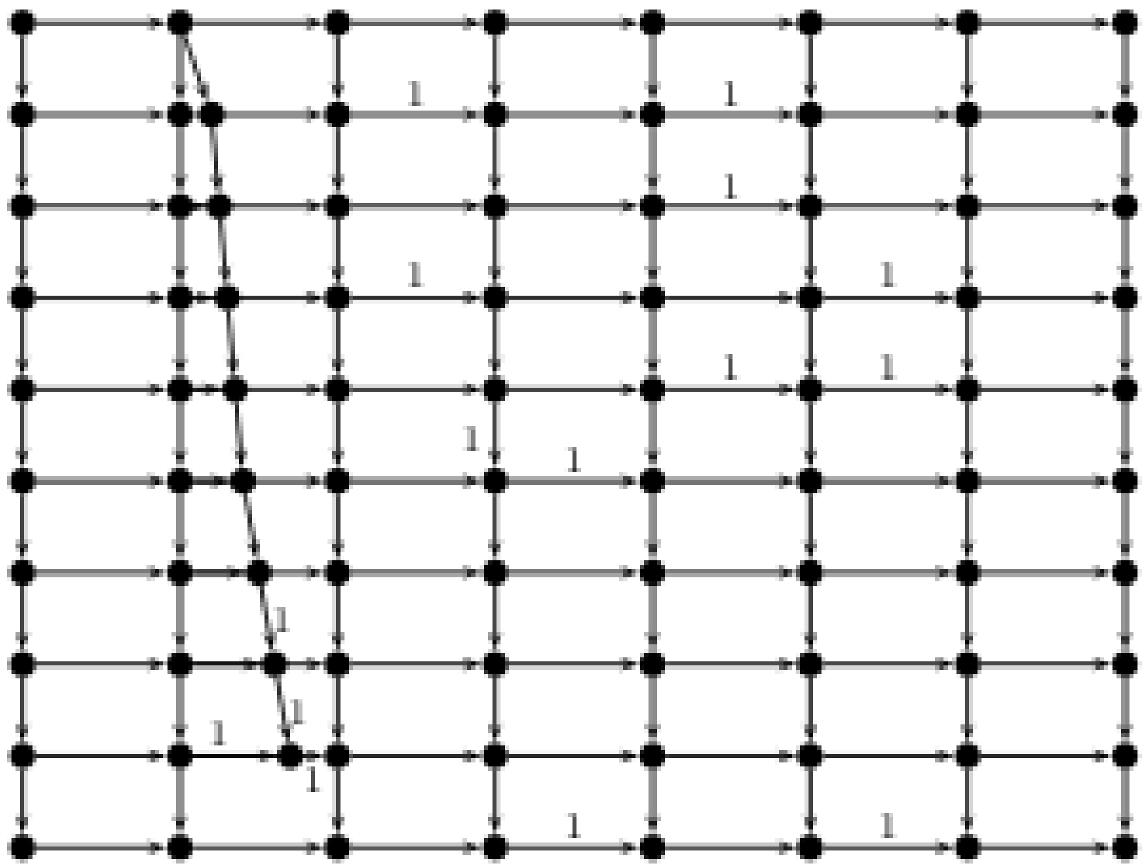
$$s_{i,\,j} = \textbf{max} \begin{cases} s_{i-1,\,j} + \text{weight of the edge between } (i\text{-}1,\, j) \text{ and } (i,\, j) \\ s_{i,\,j-1} + \text{weight of the edge between } (i,\, j\text{-}1) \text{ and } (i,\, j) \end{cases}$$

the running time is **n x m** for a **n** by **m** grid

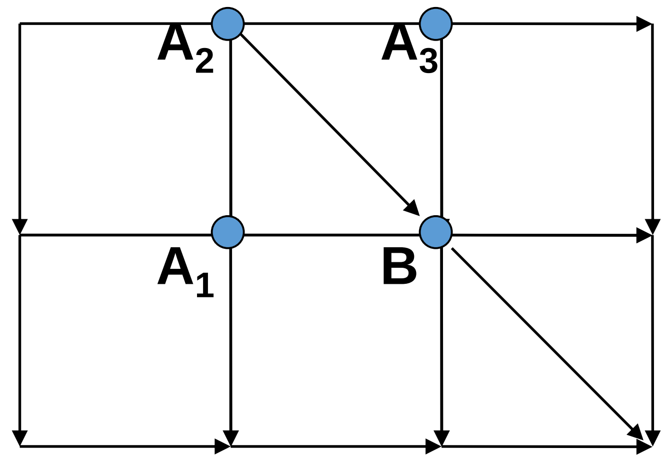
(**n** = # of rows, **m** = # of columns)

# Manhattan is not a perfect grid





## Manhattan is not a perfect grid cont.



What about diagonals?

- The score at point B is given by:

$$s_B = \textbf{max of} \begin{cases} s_{A1} + \text{weight of the edge } (A_1, B) \\ s_{A2} + \text{weight of the edge } (A_2, B) \\ s_{A3} + \text{weight of the edge } (A_3, B) \end{cases}$$

## Manhattan is not a perfect grid cont.

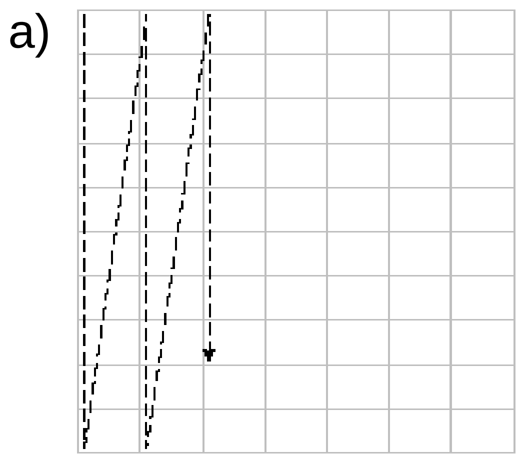
Computing the score for point **x** is given by the recurrence relation:

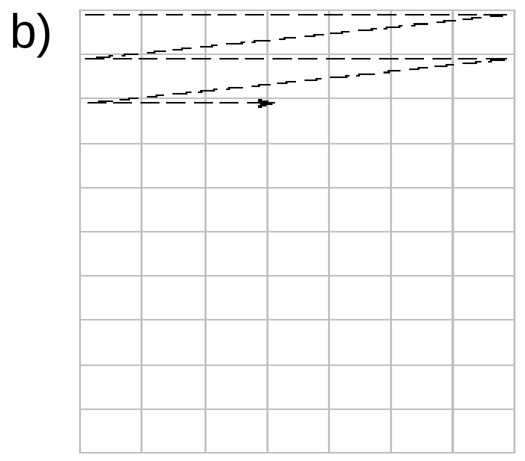
$$s_x = \text{max of} \begin{cases} s_y + \text{weight of vertex } (\mathbf{y}, \mathbf{x}) \text{ where } \mathbf{y} \in \text{Predecessors}(\mathbf{x}) \end{cases}$$

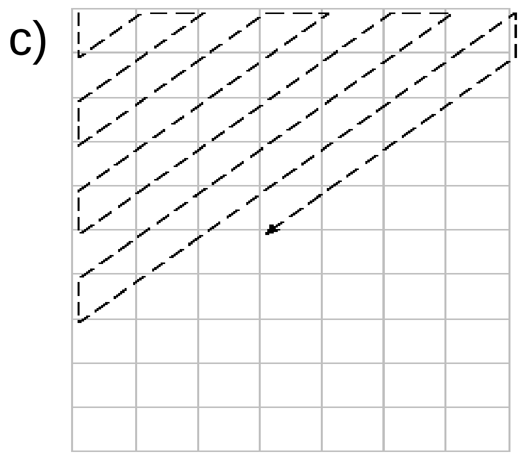
- Predecessors (**x**) – set of vertices that have edges leading to **x**
- The running time for a graph G(**V**, **E**), (**V** is the set of all vertices and **E** is the set of all edges) is O(**E**) since each edge is evaluated once

## Traversing the Manhattan grid

- 3 different strategies:
- a) Column by column
- b) Row by row
- c) Along diagonals





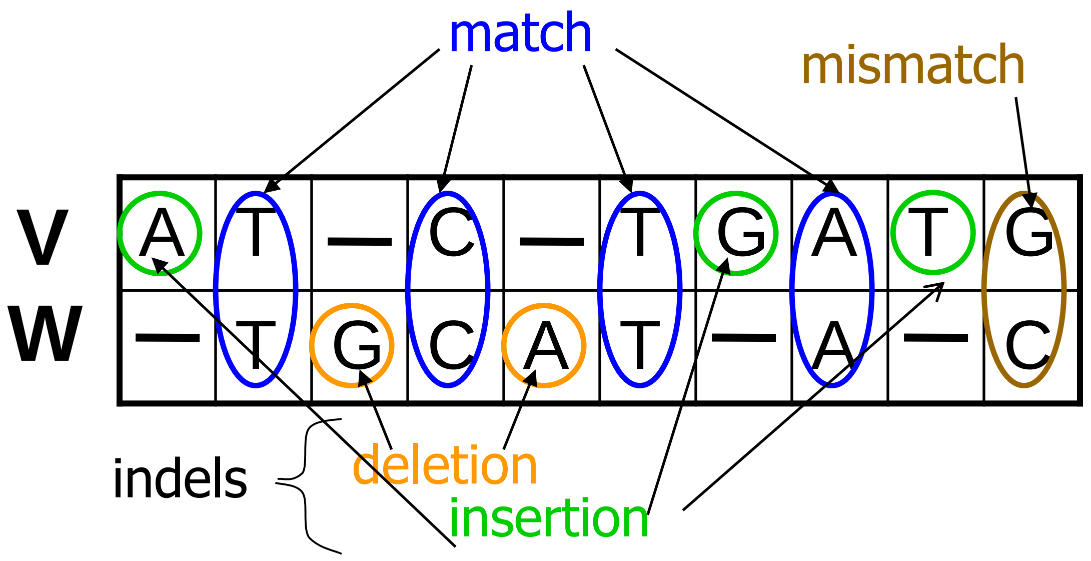


# Aligning DNA sequences

**V** = ATCTGATG n = 8

**W** = TGCATAC m = 7

4 matches
1 mismatches
2 insertions
3 deletions



## The Longest Common String (LCS) problem

- Given two sequences

  $\mathbf{v} = v_1\ v_2 \ldots v_m$ and $\mathbf{w} = w_1\ w_2 \ldots w_n$

- The LCS of **v** and **w** is a sequence of positions in

  $\mathbf{v}: 1 \leq i_1 < i_2 < \ldots < i_t \leq m$

and a sequence of positions in

  $\mathbf{w}: 1 \leq j_1 < j_2 < \ldots < j_t \leq n$

such that $i_t$ -th letter of **v** equals to $j_t$-letter of **w** and **t** is maximal

# LCS example

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| i coords: | 0 | 1 | 2 | 2 | 3 | 3 | 4 | 5 | 6 | 7 | 8 |
| elements of v | | A | T | -- | C | -- | T | G | A | T | G |
| elements of w | | -- | T | G | C | A | T | -- | A | -- | C |
| j coords: | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 5 | 6 | 6 | 7 |

(0,0)→(1,0)→(2,1)→(2,2)→(3,3)→(3,4)→(4,5)→(5,5)→(6,6)→(7,6)→(8,7)

Matches shown in red

positions in v: 2 < 3 < 4 < 6

positions in w: 1 < 3 < 5 < 6

The LCS Problem can be expressed using the grid similar to MTP grid… Finding the heaviest path from the source to sink!!!

```
        0  1  2  2  3  4  5  6  7  7
v  =       A  T  -  G  T  T  A  T  -
           |  |     |  |     |
w  =       A  T  C  G  T  -  A  -  C
        0  1  2  3  4  5  5  6  6  7
```

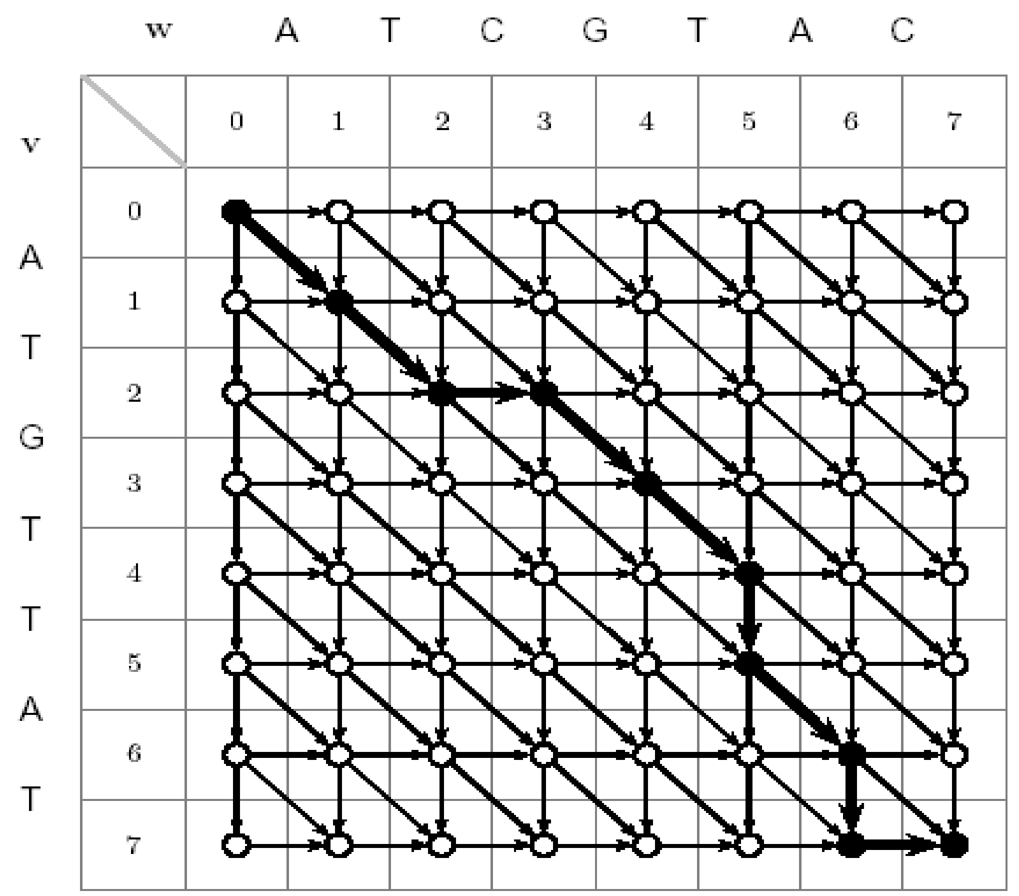
# LCS: dynamic programming

- Find the LCS of two strings

  <u>Input</u>: A weighted graph G with two distinct vertices, one labeled “source” one labeled “sink”

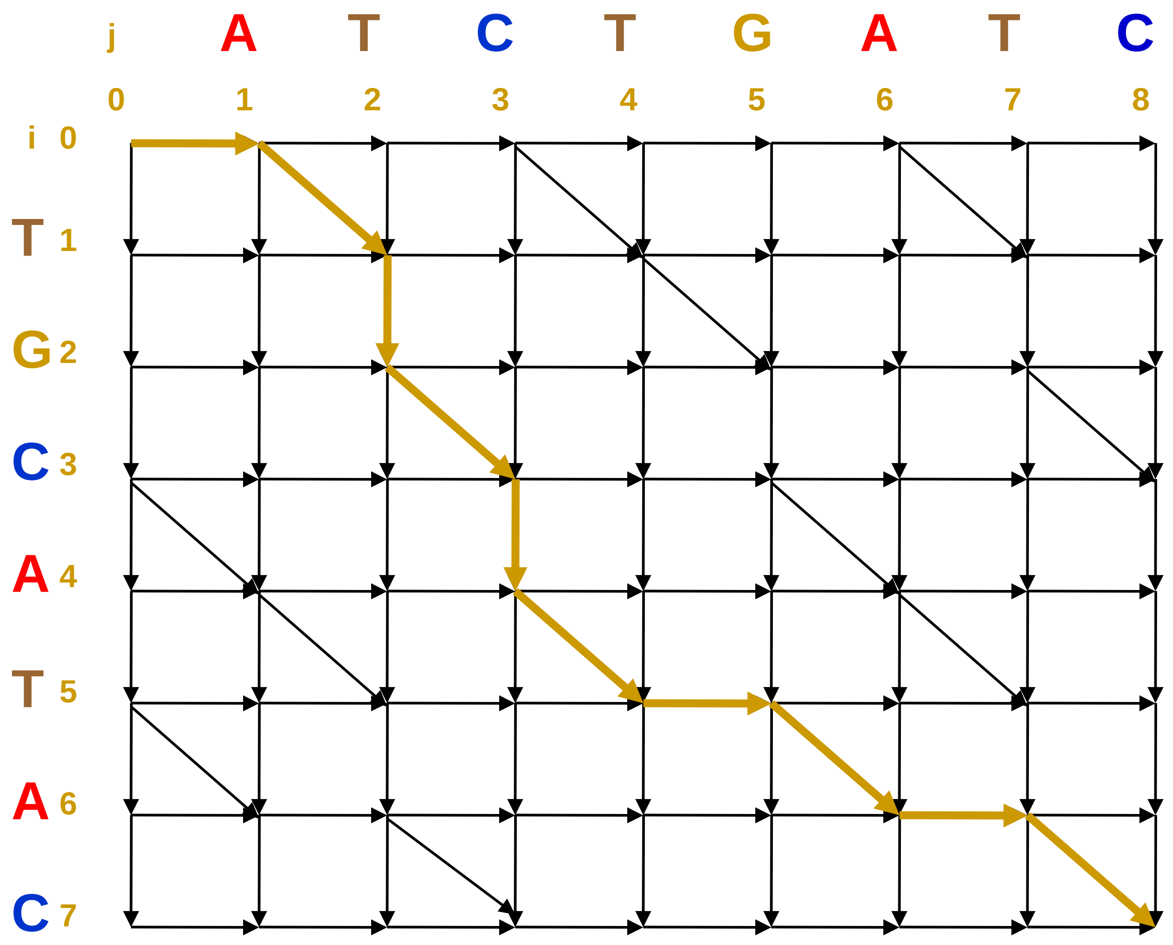
  <u>Output</u>: A longest path in G from “source” to “sink”

- Solve using an LCS edit graph with diagonals replaced with +1 edges



```
↘  ↘  →  ↘  ↘  ↓  ↘  ↓  →
A  T  -  G  T  T  A  T  -
A  T  C  G  T  -  A  -  C
```

# Edit graph for the LCS problem



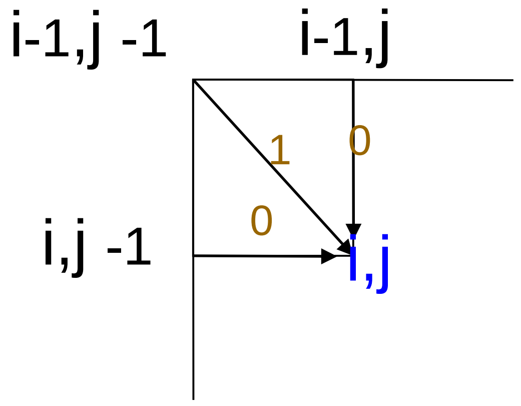
# LCS recursive function

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

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

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

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



## An issue of the LCS

```
ATGTTAT
ATCGTAC
```

```
        AT-GTTAT
LCS=5   ||  ||  |*
        ATCGT-AC
```

```
AT-GTTAT-
||  ||  |       LCS=5
ATCGT-A-C
```

- Does it mean that both alignments are equally good?
- The second one is more “gappy”, which is not a good sign in alignment as it indicates frameshift
- Need a better object function for sequence similarity, suggestions?

## The edit distance problem

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

$$d(\mathbf{v},\mathbf{w}) = \text{MIN no. of elementary operations to transform } \mathbf{v} \rightarrow \mathbf{w}$$

- **The edit distance is considered as the evolution distance, as the edit is made by evolutionary force**

## Edit distance: example

- 5 edit operations: TGCATAT → ATCCGAT
  - TGCATAT → (delete last T)
  - TGCATA → (delete last A)
  - TGCAT → (insert A at front)
  - ATGCAT → (substitute C for 3rd G)
  - ATCCAT → (insert G before last A)
  - ATCCGAT (Done)

- 4 edit operations: TGCATAT → ATCCGAT
  - TGCATAT → (insert A at front)
  - ATGCATAT → (delete 6th T)
  - ATGCATA → (substitute G for 5th A)
  - ATGCGTA → (substitute C for 3rd G)
  - ATCCGAT (Done)

# Alignment: 2 row representation

Given 2 DNA sequences **v** and **w**:

**v** : ATCTGAT  **m** = 7
**w** : TGCATA  **n** = 6

Alignment : 2 * **k** matrix ( **k** > **m**, **n** )

|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| letters of v | A | T | -- | G | T | T | A | T | -- |
| letters of w | A | T | C | G | T | -- | A | -- | C |

5 matches

2 insertions

2 deletions

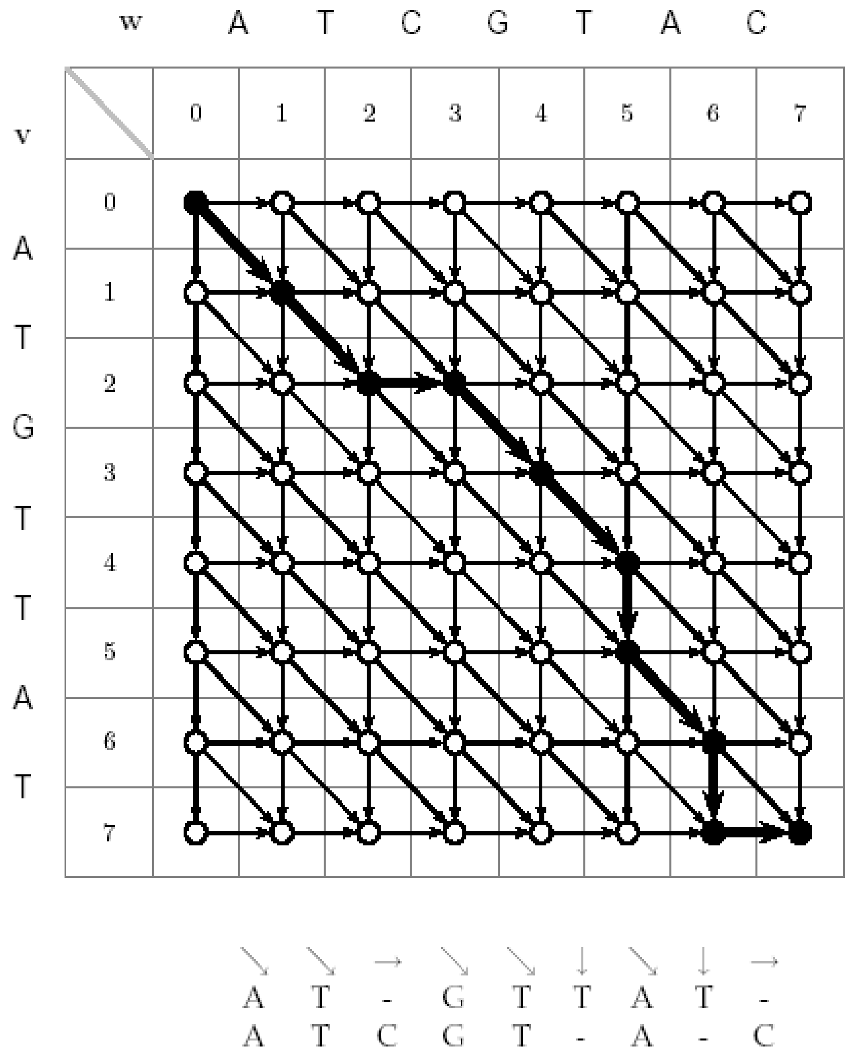
```
       0  1  2  2  3  4  5  6  7  7
v  =      A  T  -  G  T  T  A  T  -
          |  |     |  |     |
w  =      A  T  C  G  T  -  A  -  C
       0  1  2  3  4  5  5  6  6  7
```

# The Alignment Grid revisited

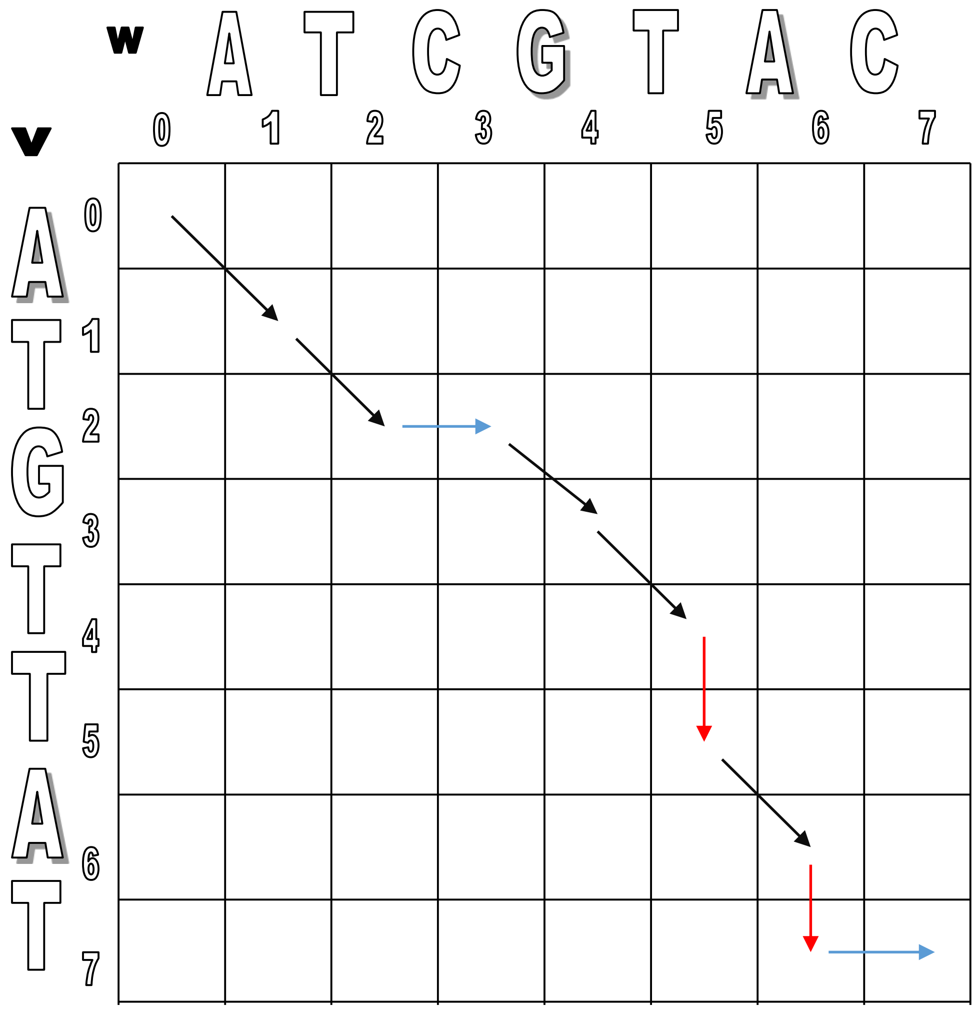
- 2 sequences used for grid

- **V** = ATGTTAT
- **W** = ATCGTAC

- Every alignment path is from source to sink



## Alignments in edit graph

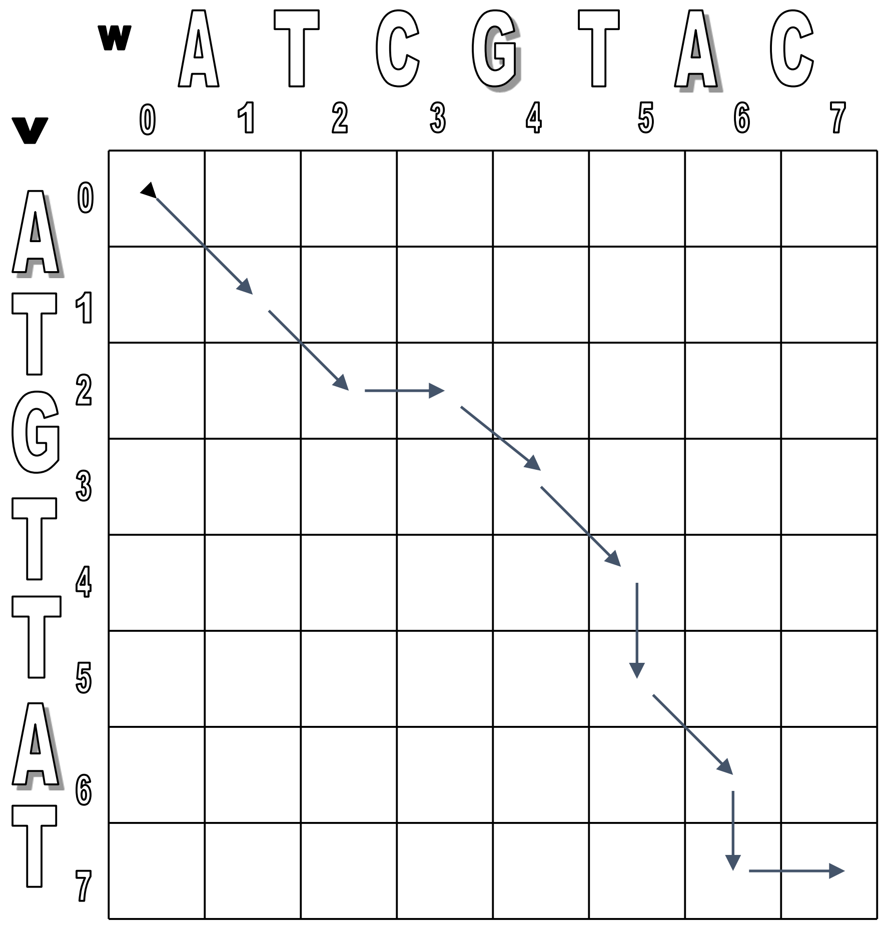


$\downarrow$ and $\rightarrow$ represent indels in **v** and **w** with edit operation 1.

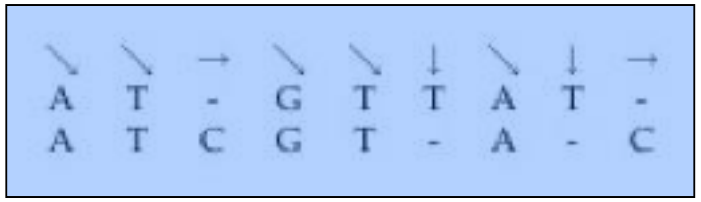
$\searrow$ represent match or mismatch with edit operation of 0 or 1.

- The total number of edit operations of the alignment path is 4.

## Alignment as path in the edit graph

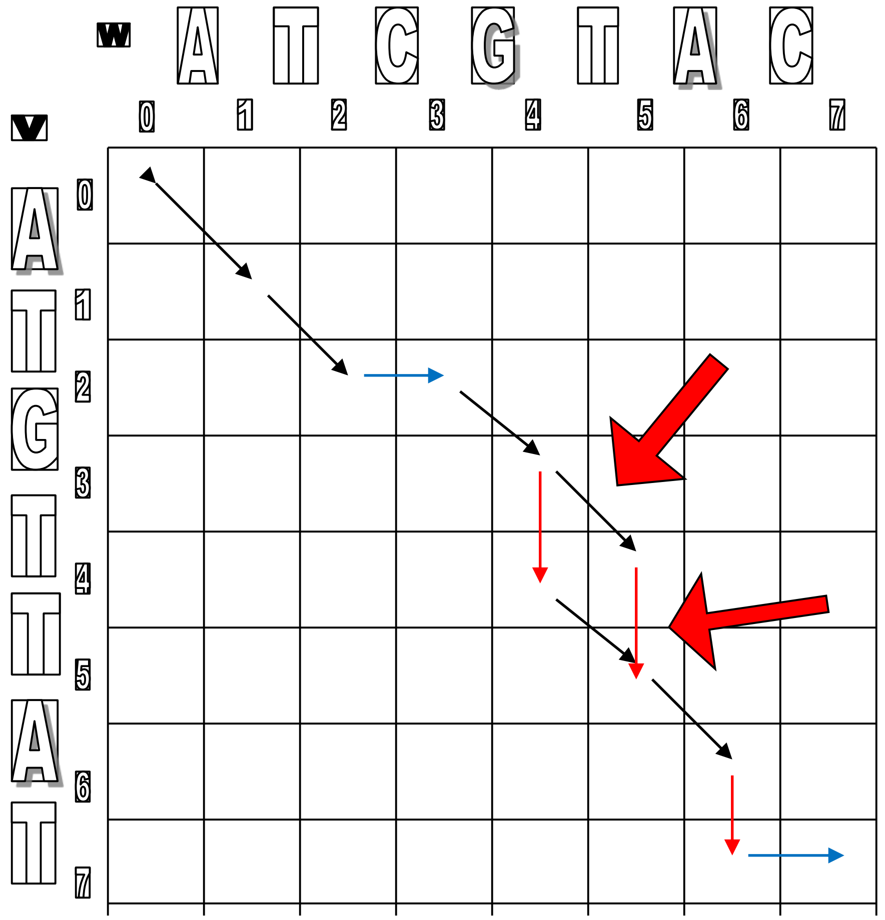


Every path in the edit graph corresponds to an alignment:



| ↘ | ↘ | → | ↘ | ↘ | ↓ | ↘ | ↓ | → |
|---|---|---|---|---|---|---|---|---|
| A | T | - | G | T | T | A | T | - |
| A | T | C | G | T | - | A | - | C |

# Equivalently good solutions may present



<u>Old Alignment</u>

```
     0122345677
v=   AT_GTTAT_
w=   ATCGT_A_C
     0123455667
```

<u>New Alignment</u>

```
     0122345677
v=   AT_GTTAT_
w=   ATCG_TA_C
     0123445667
```

# More details into edit distance solution

- Dynamic programming

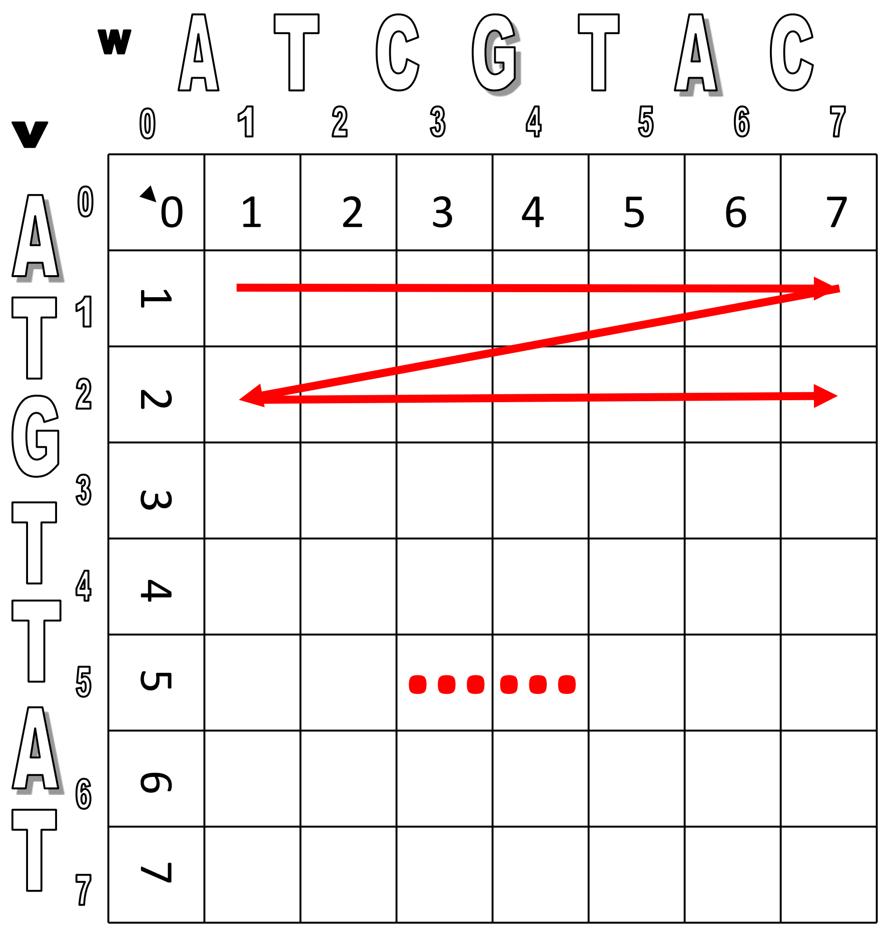
$$s_{i,j} = \max \begin{cases} s_{i-1,\, j-1} + 1 \text{ if } v_i = w_j & \searrow \\ s_{i-1,\, j} & \downarrow \\ s_{i,\, j-1} & \rightarrow \end{cases}$$

# Initializing the DP table

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

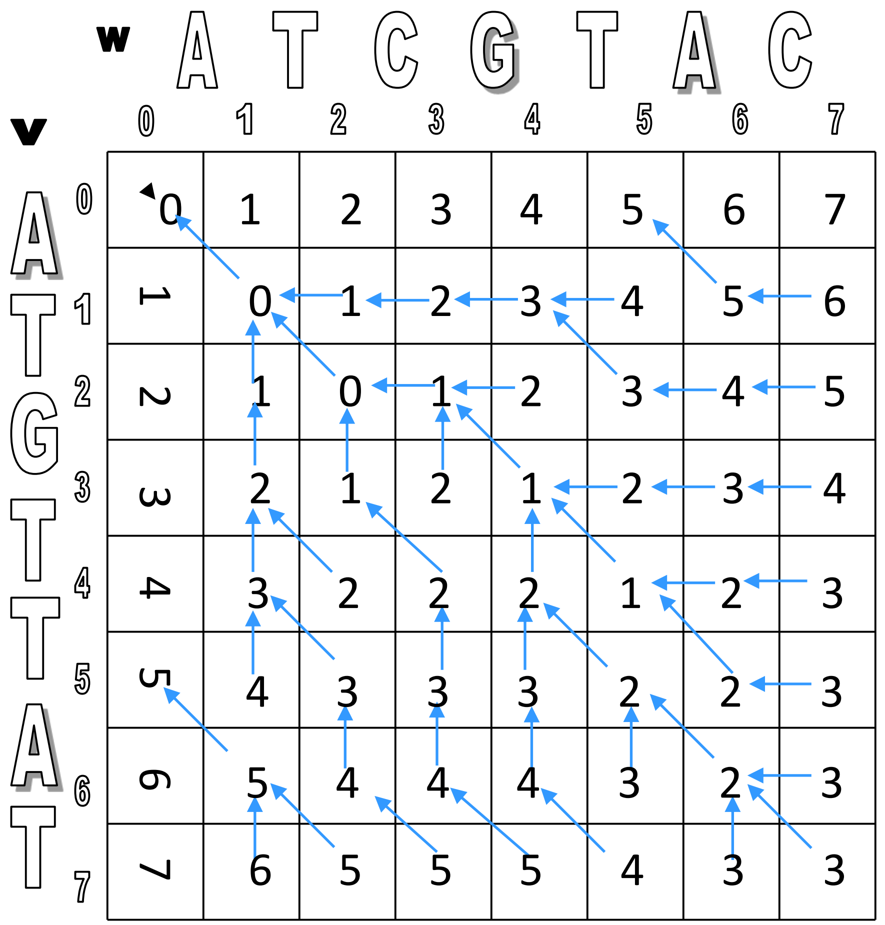
Initialize *1*st row and *1*st column to be all corresponding edit costs.

# Filling the table



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

## Filling the table cont.

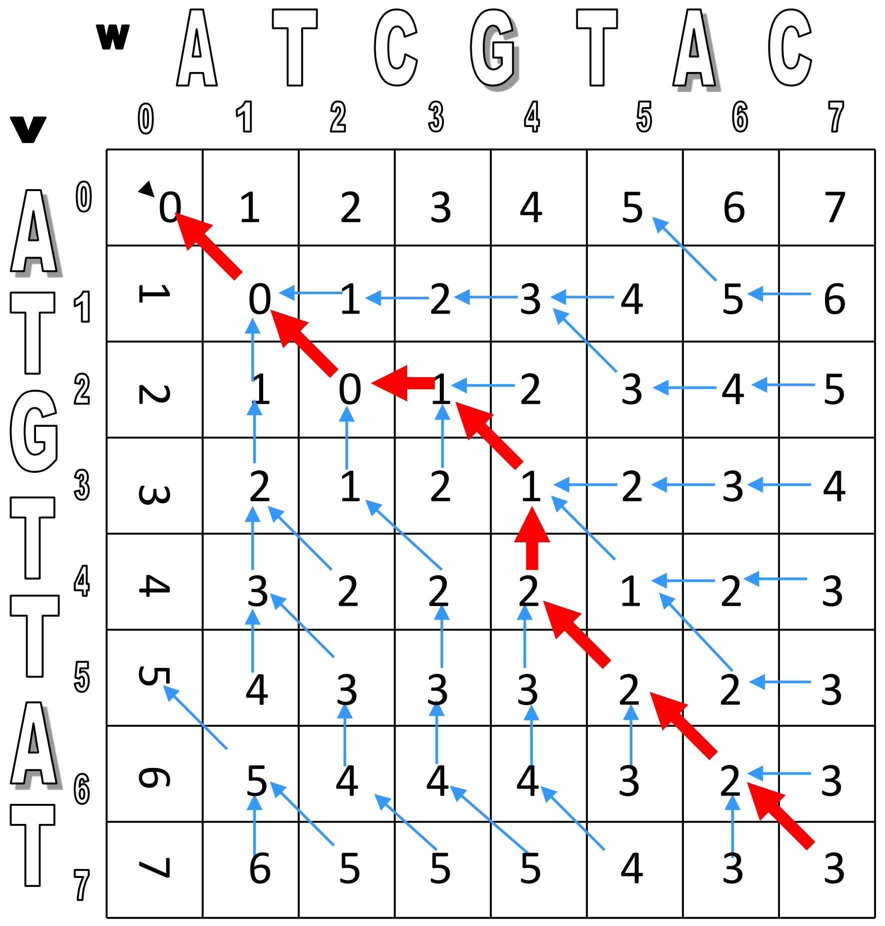


| v \ w |  |  | A | T | C | G | T | A | C |
|---|---|---|---|---|---|---|---|---|---|
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|  | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| A | 1 | 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| T | 2 | 2 | 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| G | 3 | 3 | 2 | 1 | 2 | 1 | 2 | 3 | 4 |
| T | 4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

## Trace back: find the optimal edit graph and generate the alignment

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

# Traceback



```
ATCG-TAC
|| | ||*
AT-GTTAT
```

## Running time

- It takes O(*nm*) time to fill in the *n * 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 * m* matrix.