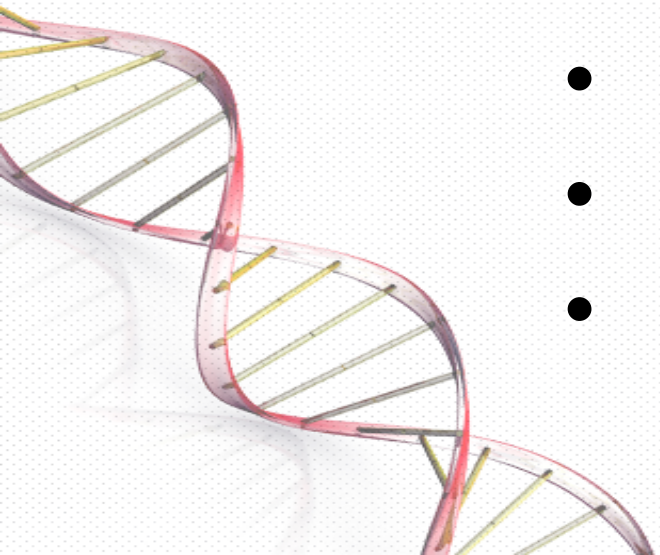
Sequence Alignments

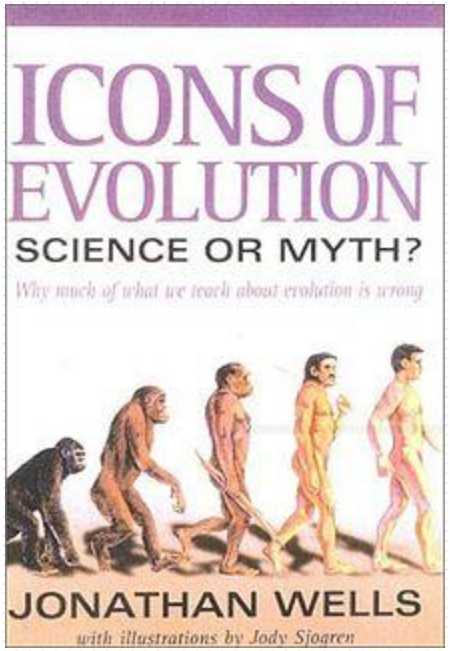
Hammad Naveed

Why alignments ?

```
AAB24882        TYHMCQFHCRYVNNHSGEKLYECNERSKAFSCPSHLQCHKRRQIGEKTHEHNQCGKAFPT 60
AAB24881        --------------------YECNQCGKAFAQHSSLKCHYRTHIGEKPYECNQCGKAFSK 40
                                    ****: .***:  * *:** * :****.:* *******..

AAB24882        PSHLQYHERTHTGEKPYECHQCGQAFKKCSLLQRHKRTHTGEKPYE-CNQCGKAFAQ- 116
AAB24881        HSHLQCHKRTHTGEKPYECNQCGKAFSQHGLLQRHKRTHTGEKPYMNVINMVKPLHNS 98
                 **** *:***********:***:**.: .***************     :  *.: :
```

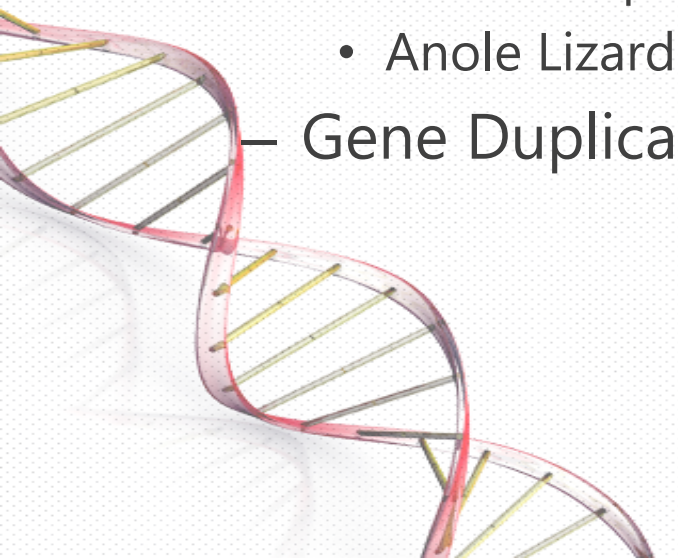
- Detect homology (similarity)
- Study evolution
- Predict functions
- Model 3D-structure

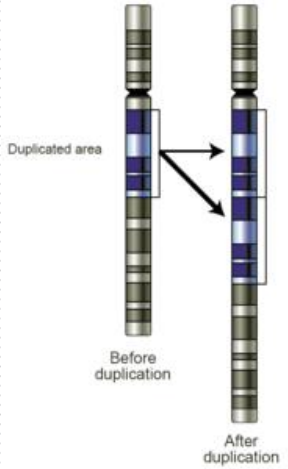


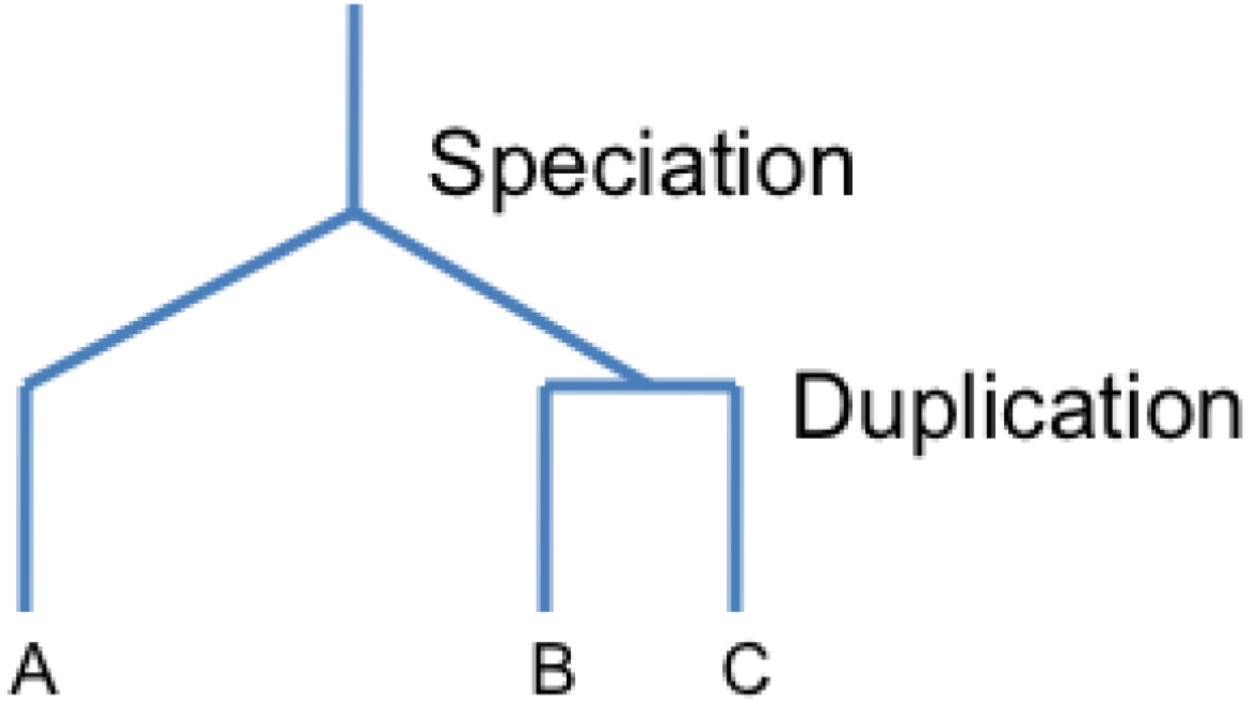


Sequence similarity

- High sequence similarity indicates homology
- Homologs have similar 3D-structure
- Homologs have a common ancestor
- Complexity arises due to
  - Speciation
    - Breed vs specie
    - Anole Lizards example video
  - Gene Duplication







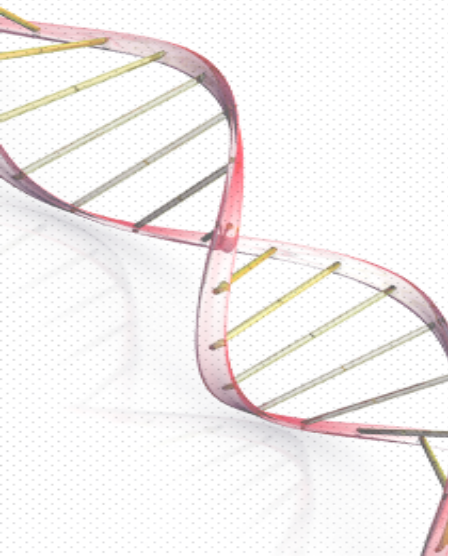
Convergent evolution

What is an alignment

```
THISSEQUENCE
||  ||||||||      10/12 Identical
THATSEQUENCE


THATSEQUENCE
||      |  |       4/12 Identical
THISISASEQUENCE


THISISA-SEQUENCE
||    | ||||||||   11/12 Identical
TH----ATSEQUENCE
```

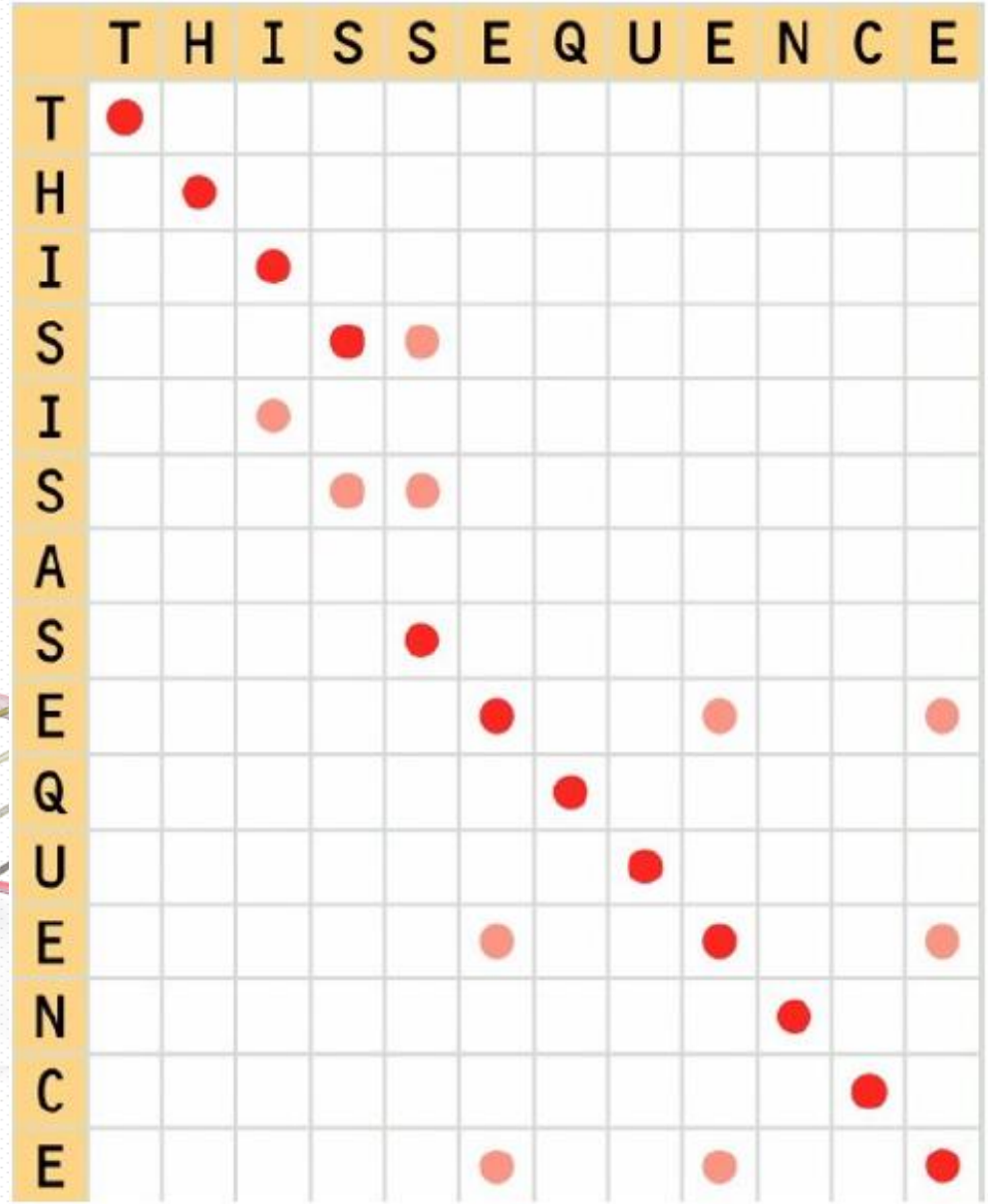
- Match
- Mismatch
- Insertions/deletions (Indels)

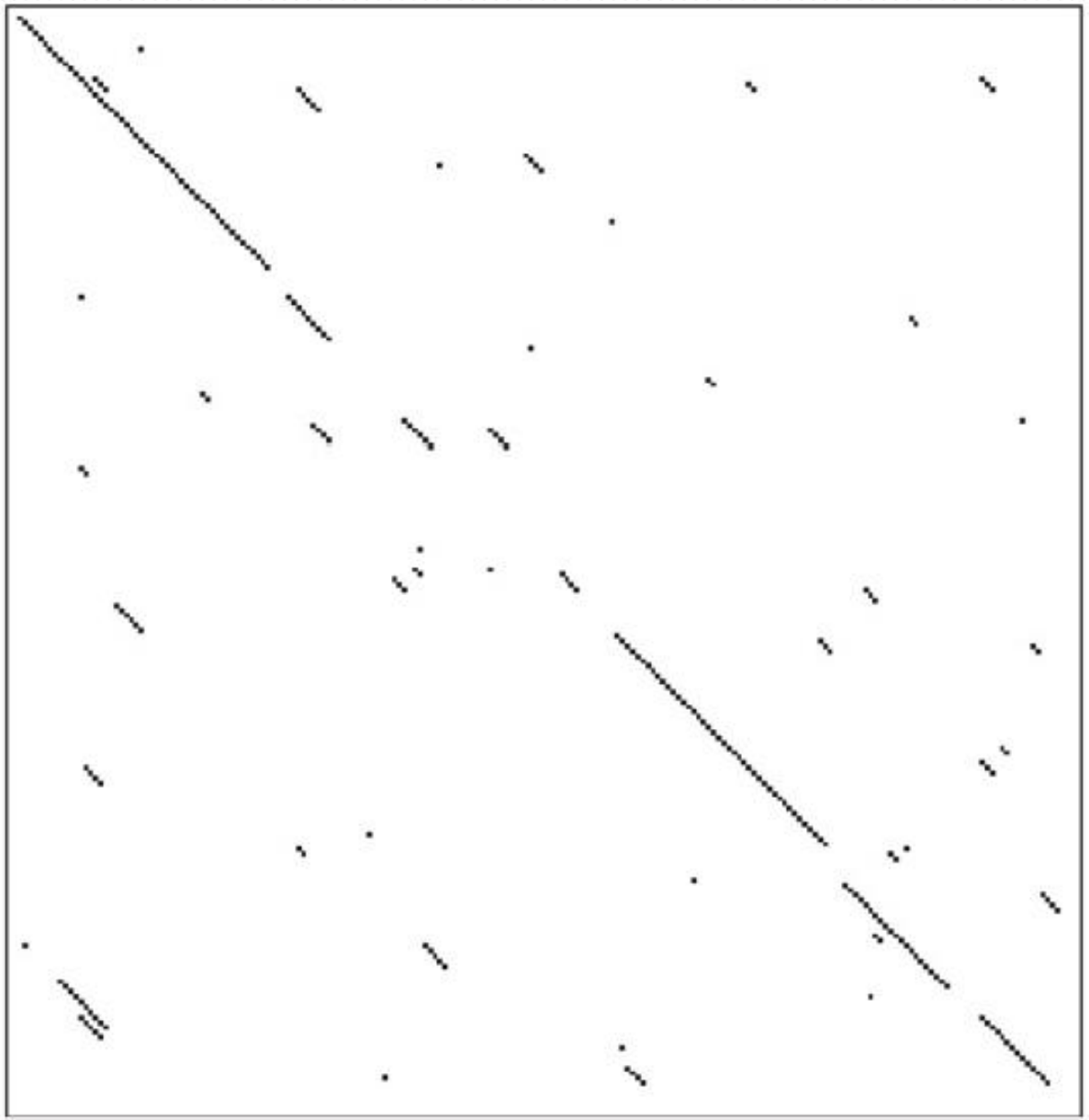
An alignment matrix



		D	O	R	O	T	H	Y	C	R	O	W	F	O	O	T	H	O	D	G	K	I	N
	D	D																	D				
	O		O		O						O			O	O			O					
	R			R						R													
	O		O		O						O			O	O			O					
	T					T										T							
	H						H										H						
	Y							Y															
	H						H		→	→	→	→	→	→	→	→	H						
	O		O		O						O			O	O			O					
	D	D																	D				
	G																			G			
	K																				K		
	I																					I	
	N																						N

Dotplots





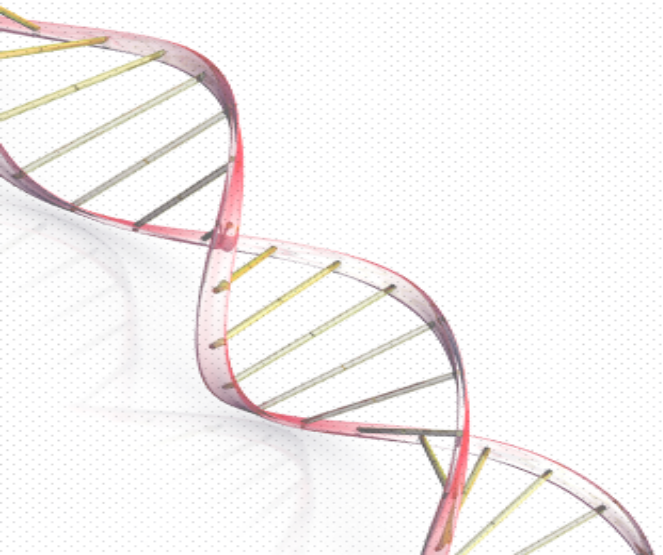
Types of alignment

- Global alignment

```
FTFTALILLAVAV
F--TAL-LLA-AV
```

- Local alignment

```
FTFTALILL-AVAV
--FTAL-LLAAV--
```



Inserting gaps

(A)

```
Bovine PI-3Kinase p110a          LNWENPDIMSELLFQNNEIIFKNGDDLRQDMLTLQIIRIMENIWQNQGLDLRMLPYGCLSIGDCVGLIEVVRNSHTIMQIQCKGGLKGAL
cAMP-dependent protein kinase --WENPAQNTAHLDQFERIKTLGTGSFGRVMLVKHMETGNHYAMKILDKQKVVKLKQIEHTLNEKRILQAVNFPFLVKLEFSFKDNSNLY

Bovine PI-3Kinase p110a          QFNSHTLHQWLKDKNKGEIYDAAIDLFTRSCAGYCVATFILGIGDRHNSNIMVKDDGQLFHIDFGHFLDHKKKFGYKRERVPFVLTQDF
cAMP-dependent protein kinase MVMEYVPGGEMFSHLRRIGRFSEPHARFYAAQIVLTFEYLHSLDLIYRDLKPENLLIDQQGYIQVTDFGFAKRVKGRTWXLCGTPEYLAP

Bovine PI-3Kinase p110a          LIVISKGAQECTKTREFERFQEMCYKAYLAIRQHANLFINLFSMMLGSGMPELQSFDDIAYIRKTLALDKTEQEALEYFMKQMNDAHHGG
cAMP-dependent protein kinase EIILSKGYNKAVDWWALGVLIYEMAAGYPPFFADQPIQIYEKIVSGKVRFPSHFSSDLKDLLRNLLQVDLTKRFGNLKNGVNDIKNHKWF

Bovine PI-3Kinase p110a          WTTKMDWIFHTIKQHALN-----------------------------------
cAMP-dependent protein kinase ATTDWIAIYQRKVEAPFIPKFKGPGDTSNFDDYEEEEIRVXINEKCGKEFSEF
```

(B)

```
Bovine PI-3Kinase p110a          LNWENPDIMSELLFQNNEIIFKNGDDLRQDMLTLQIIRIMENIWQNQGLDLRMLPYGCLSIGDCVGLIEVVRNSHTIMQIQCKGGLKGAL
cAMP-dependent protein kinase ?-WENPAQNTAHLDQFERIKTLGTGSFGRVMLVKHM--ETGNHYAMKILDKQKV-VKLKQIEHTLNEKRILQAVNFPFLVKLEFSFKDN-

Bovine PI-3Kinase p110a          QFNSHTLHQWLKDKNKGEIYDAAIDLFTRSCAGYCVATFILGIGDRHNSNIMVKD-DGQLFHIDFGHFLDHKKKKFGYKRERVPFVL--T
cAMP-dependent protein kinase -SNLYMVMEYVPGGEMFSHLRR-IGRFSEPHARFYAAQIVLTFEYLHSLDLIYRDLKPENLLIDQQGYIQVTDFGFAKRVKGRTWXLCGT

Bovine PI-3Kinase p110a          QDFL---IVISKGAQECTKTREFERF-QEMC--YKAYLAIRQHANLFINLFSMMLGSGMPELQSFDDIAYIRKTLALDKTEQEALEYFMK
cAMP-dependent protein kinase PEYLAPEIILSKGYNKAVDWWALGVLIYEMAAGYPPFFA-DQPIQIYEKIVSGKVRF--PSHFSSDLKDLLRNLLQVDLTKR--FGNLKN

Bovine PI-3Kinase p110a          QMNDAHHGGWTTKMDWI----------------------FHTIKQHAL----N----------
cAMP-dependent protein kinase GVNDIKNHKWFATTDWIAIYQRKVEAPFIPKFKGPGDTSNFDDYEEEEIRVXINEKCGKEFSEF
```

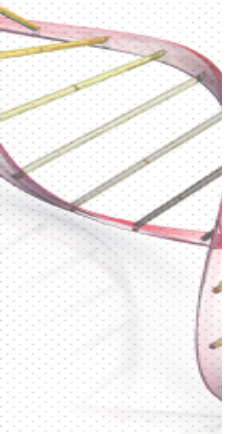
What is an optimal alignment?

```
THISSEQUENCE
||  ||||||||      10/12 Identical
THATSEQUENCE


THATSEQUENCE
||      |  |       4/12 Identical
THISISASEQUENCE


THISISA-SEQUENCE
||    | ||||||||   11/12 Identical
TH----ATSEQUENCE
```

Different scoring

```
T H I S S E Q U E N C E
5 8-1 1 4 5 6 0 5 6 9 5        Score = 52
T H A T S E Q U E N C E


T H A T S E Q U E N C E
5 8-1-1-2 0-1 0 5 0 0 5        Score = 18
T H I S I S A S E Q U E N C E


T H I S I S A - S E Q U E N C E
5 8 0 0 0 0 4 0 4 5 6 0 5 6 9 5        Score = 56
T H - - - - A T S E Q U E N C E
```

T-T = 5
H-H= 8
S-S = 4
E-E = 5
Q-Q= 6
U-U= 0
N-N= 6
C-C = 9
Gap=0

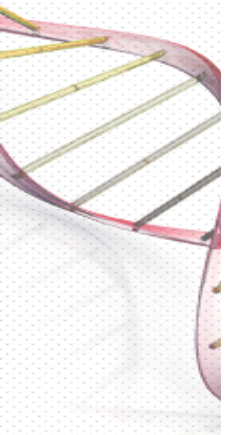


With Gap cost

```
T H I S S E Q U E N C E
5 8-1 1 4 5 6 0 5 6 9 5        Score = 52
T H A T S E Q U E N C E
```

T-T = 5
H-H= 8
S-S = 4
E-E = 5
Q-Q= 6
U-U= 0
N-N= 6
C-C = 9
Gap=-1

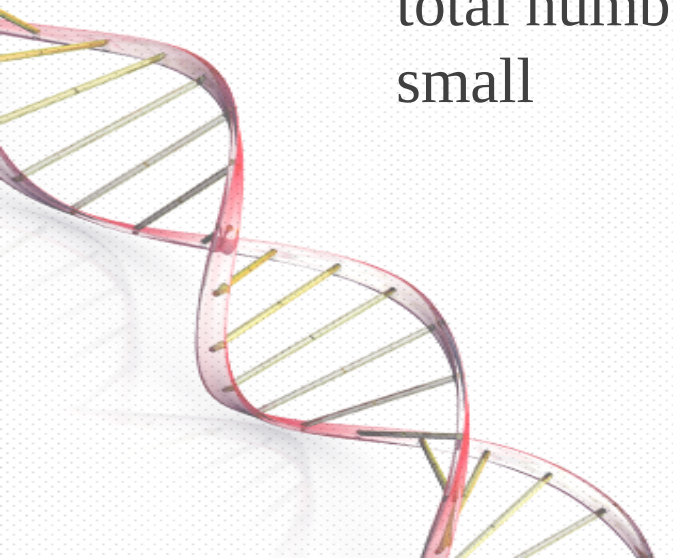
```
T H A T S E Q U E N C E
5 8-1-1-2 0-1 0 5 0 0 5        Score = 18
T H I S I S A S E Q U E N C E
```



```
T H I S I S A - S E Q U E N C E
5 8-1-1-1-1 4-1 4 5 6 0 5 6 9 5        Score = 51
T H - - - A T S E Q U E N C E
```

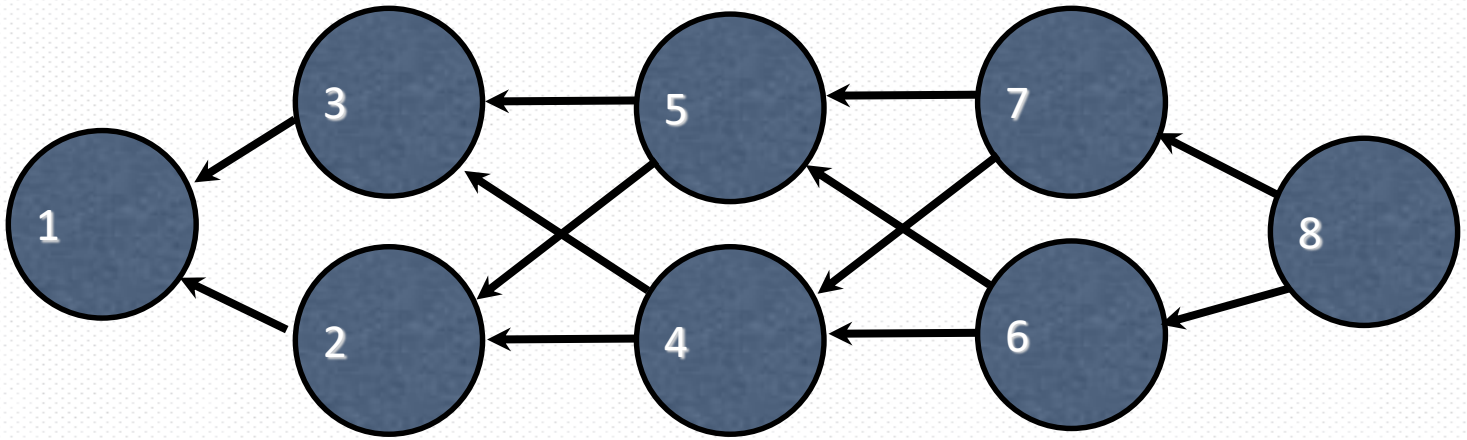
Dynamic Programming

- Algorithmic technique for optimization problems that have two properties:
  - *Optimal substructure*: Optimal solution can be computed from optimal solutions to sub-problems
  - *Overlapping sub-problems*: Sub-problems overlap such that the total number of distinct sub-problems to be solved is relatively small

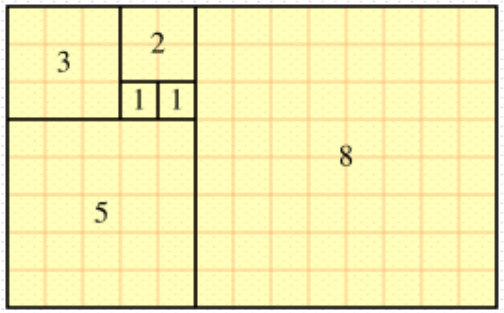


Dynamic Programming

- Break problem into overlapping subproblems
- use *memoization*: remember solutions to subproblems that we have already seen



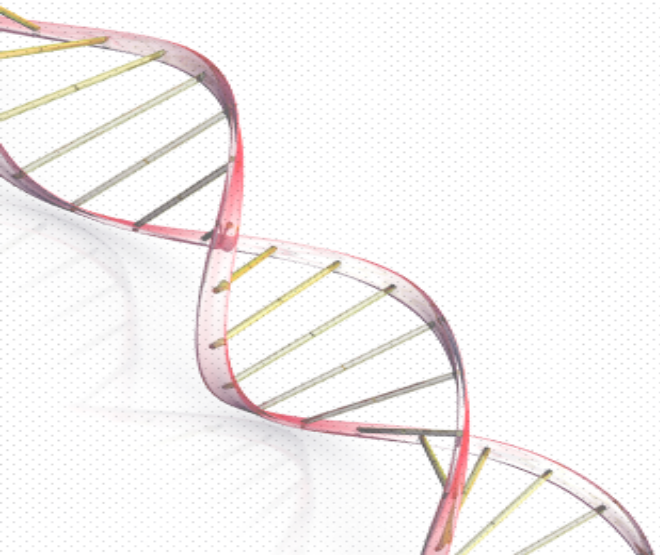
Fibonacci example



- 1,1,2,3,5,8,13,21,...
- *fib(n) = fib(n - 2) + fib(n - 1)*
- Could implement as a simple recursive function
- However, complexity of simple recursive function is exponential in *n*

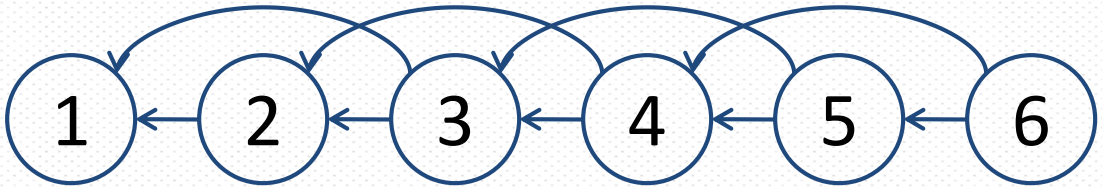
Fibonacci dynamic programming

- Two approaches
  1. *Memoization:* Store results from previous calls of function in a table (top down approach)
  2. Solve subproblems from smallest to largest, storing results in table (bottom up approach)
- Both require evaluating all (n-1) subproblems only once: O(n)



Dynamic Programming Graphs

- Dynamic programming algorithms can be represented by a directed acyclic graph
  - Each subproblem is a vertex
  - Direct dependencies between subproblems are edges



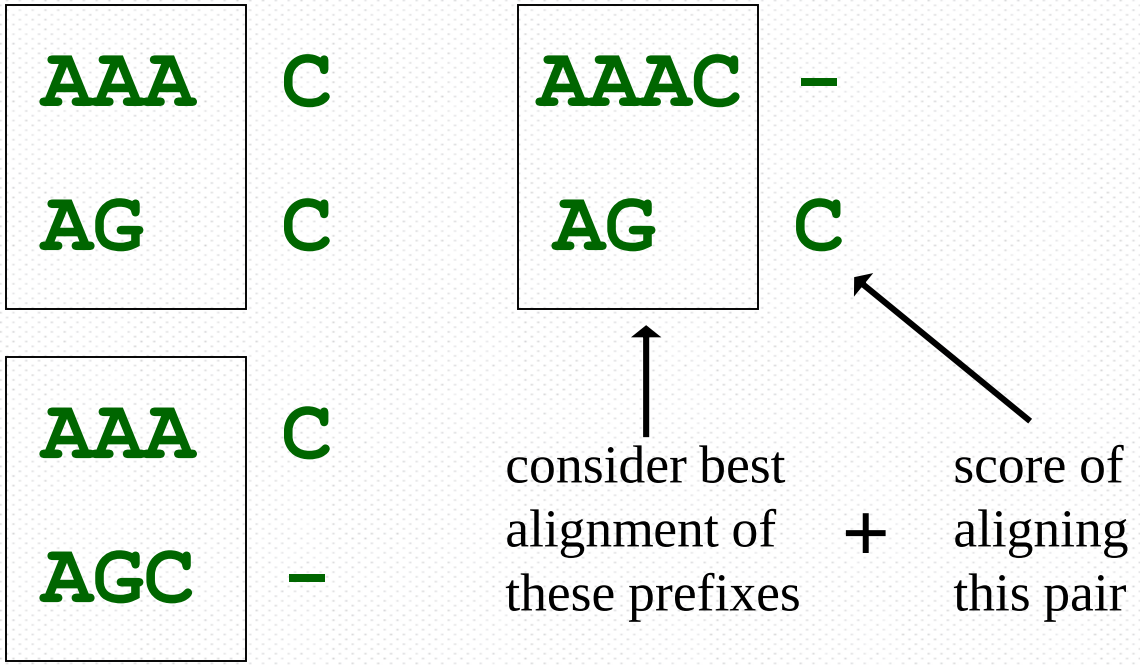
graph for fib(6)

Pairwise Alignment Via Dynamic Programming

- first algorithm by Needleman & Wunsch, *Journal of Molecular Biology*, 1970
- *dynamic programming algorithm:* determine best alignment of two sequences by determining best alignment of all prefixes of the sequences

Dynamic Programming Idea

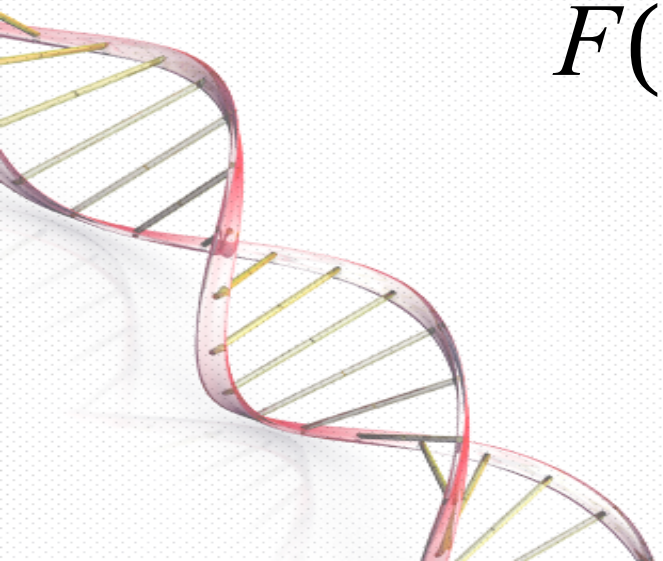
- consider last step in computing alignment of **AAAC** with **AGC**
- three possible options; in each we'll choose a different pairing for end of alignment, and add this to the best alignment of previous characters



DP Algorithm for Global Alignment with Linear Gap Penalty

- Subproblem: *F(i,j)* = score of best alignment of the length *i* prefix of *x* and the length *j* prefix of *y*.

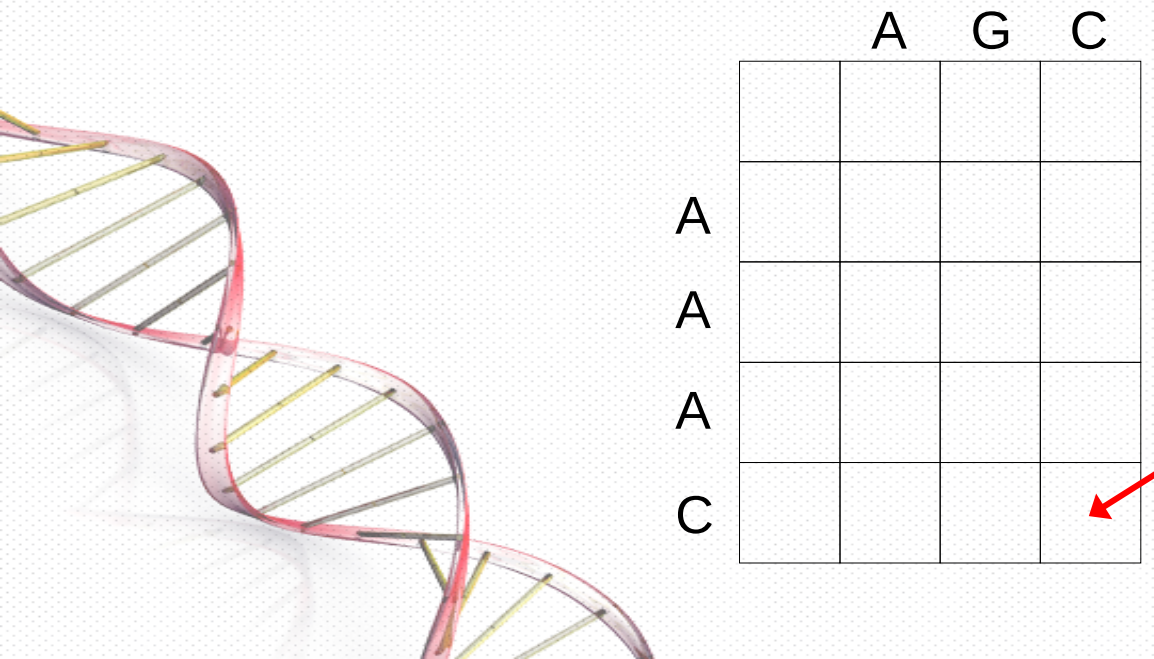
$$F(i,j) = \max \begin{cases} F(i-1, j-1) + S(x_i, y_j) \\ F(i-1, j) + s \\ F(i, j-1) + s \end{cases}$$



Dynamic Programming Implementation

- given an n-character sequence x, and an m-character sequence y
- construct an $(n+1) \times (m+1)$ matrix F
- $F(i, j)$ = score of the best alignment of

$$x[1 \ldots i] \text{ with } y[1 \ldots j]$$



Global Alignment Example

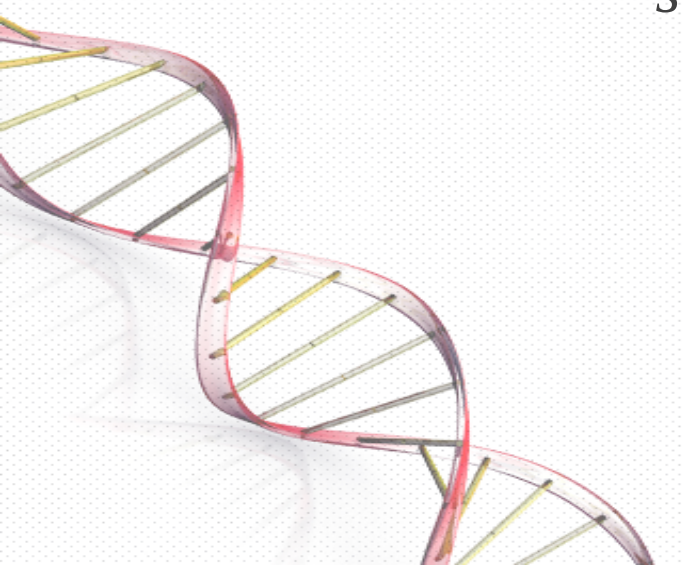
- suppose we choose the following scoring scheme:

$$S(x_i, y_i) =$$

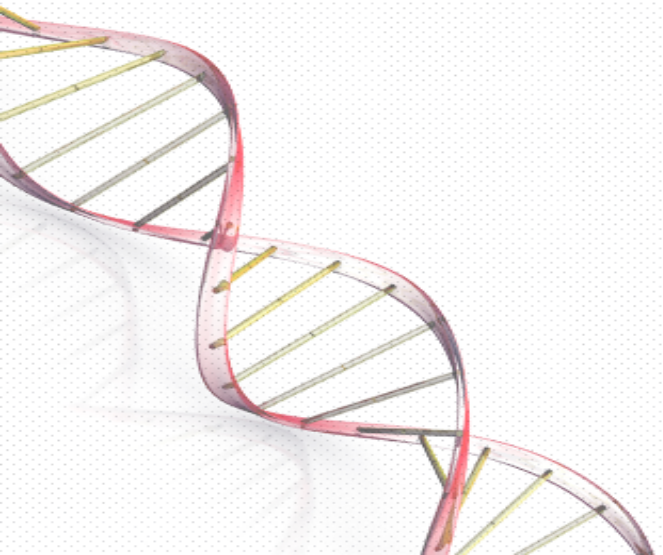
+1 when $x_i = y_i$

-1 when $x_i \neq y_i$

s (penalty for aligning with a space) = -2

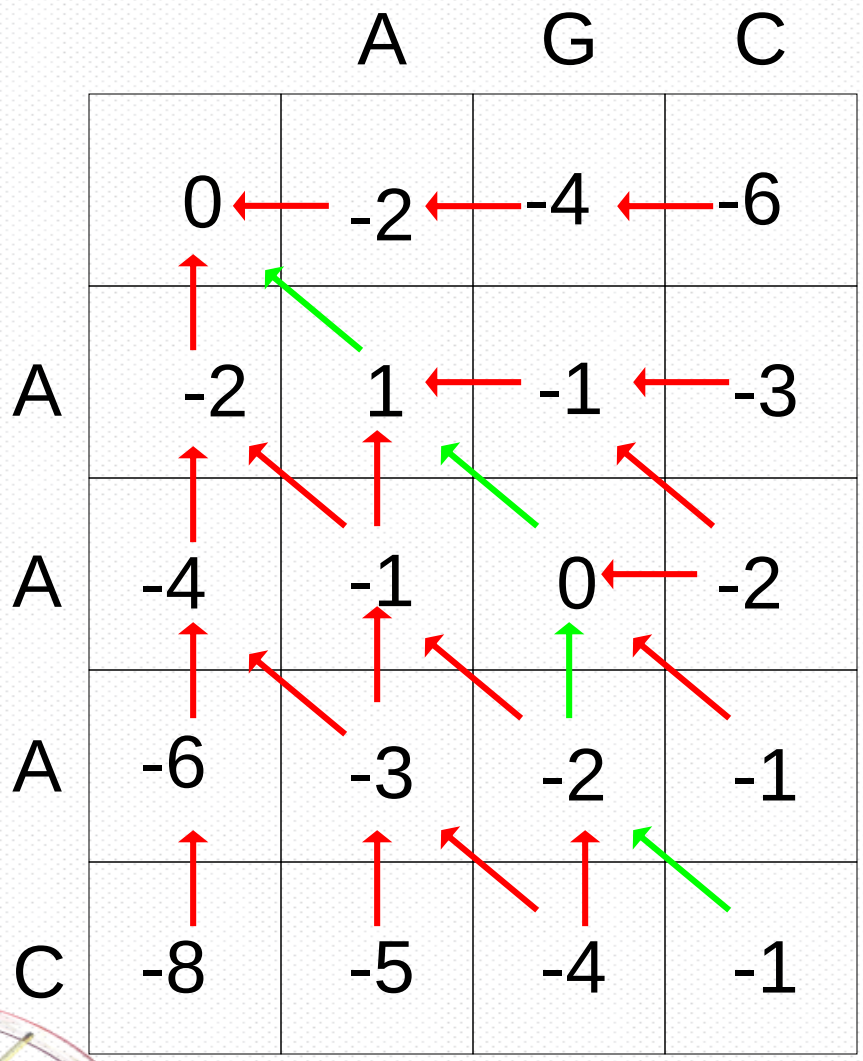


Initializing Matrix: Global Alignment with Linear Gap Penalty



		A	G	C
	0 ←	s ←	$2s$ ←	$3s$
A	s			
A	$2s$			
A	$3s$			
C	$4s$			

Global Alignment Example



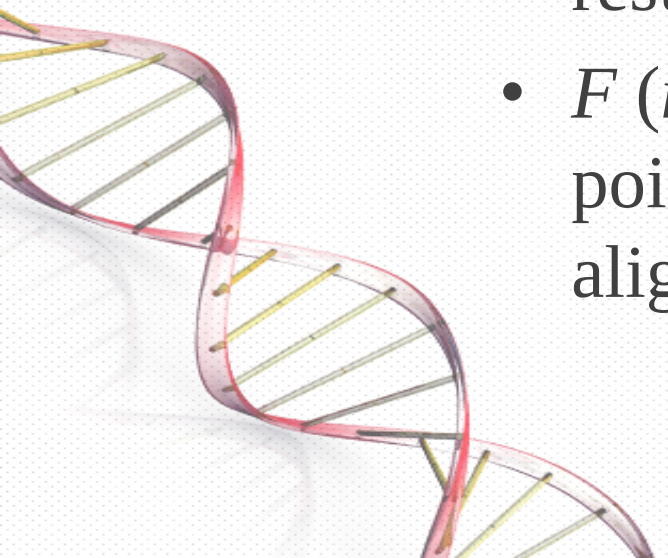
one optimal alignment

x: A A A C
y: A G - C

but there are three optimal alignments here (can you find them?)

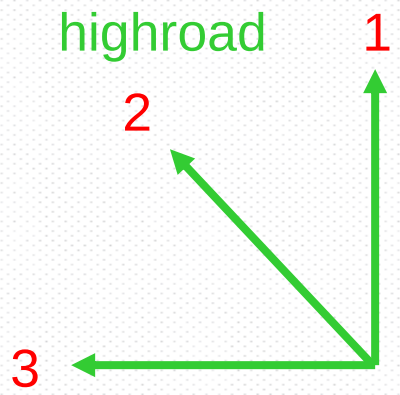
DP Algorithm Sketch: Global Alignment

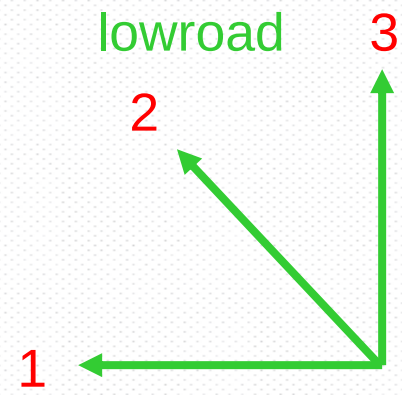
- initialize first row and column of matrix
- fill in rest of matrix from top to bottom, left to right
- for each $F(i, j)$, save pointer(s) to cell(s) that resulted in best score
- $F(m, n)$ holds the optimal alignment score; trace pointers back from $F(m, n)$ to $F(0, 0)$ to recover alignment



Equally Optimal Alignments

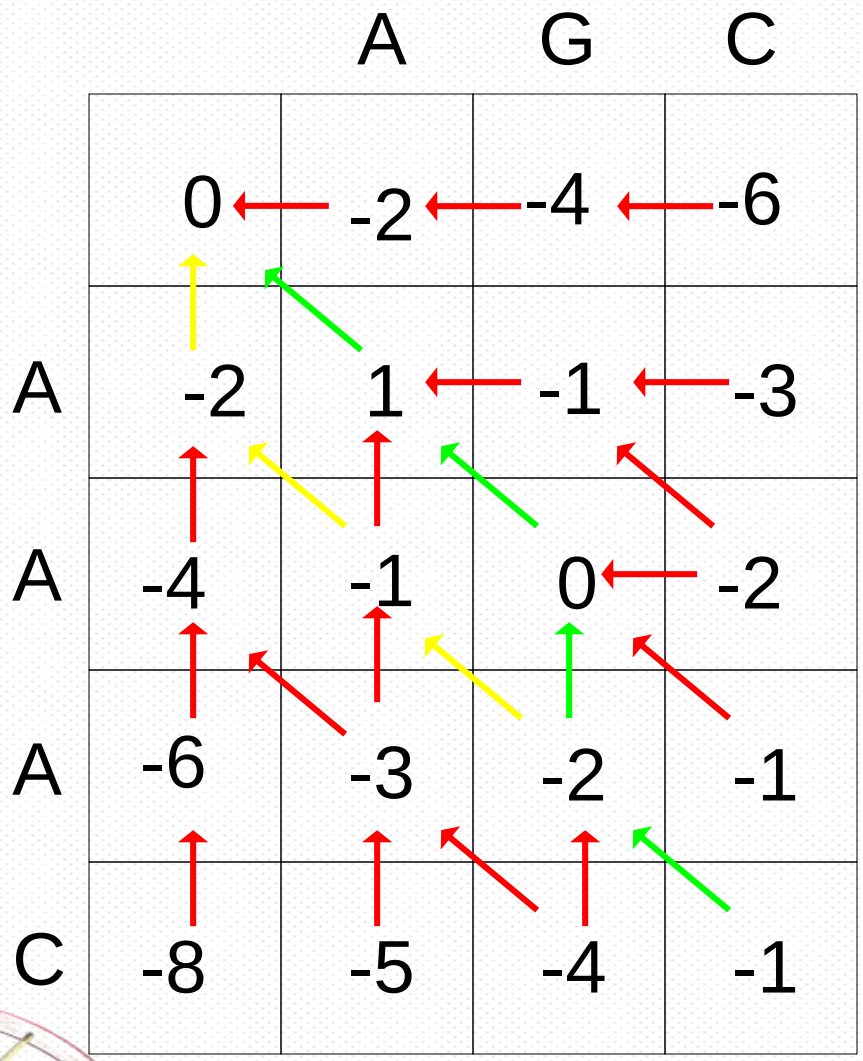
- many optimal alignments may exist for a given pair of sequences
- can use preference ordering over paths when doing traceback

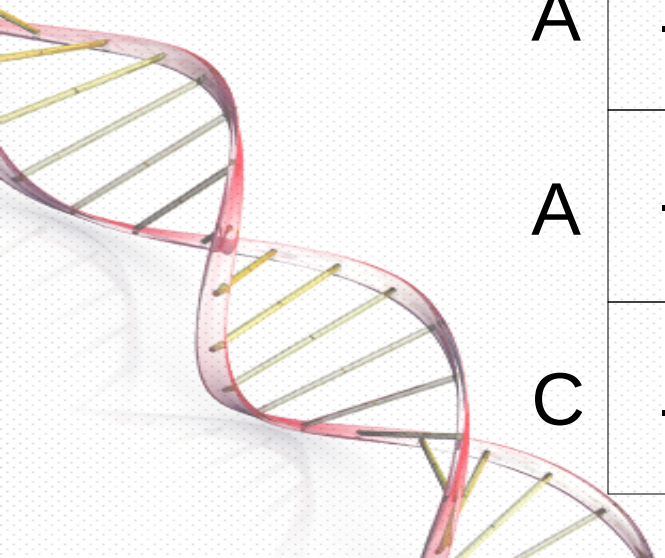




- *High road* and *low road* alignments show the two most different optimal alignments

High road & Low road Alignments





High road alignment

x: A A A C

y: A G - C

Low road alignment

x: A A A C

y: - A G C

Another Example

- Initialization step: Create Matrix with M + 1 columns
- and N + 1 rows. First row and column filled with 0.

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0											
G	0											
A	0											
T	0											
C	0											
G	0											
A	0											

Match = +1
Mismatch = 0
Space = 0

Another Example

Matrix fill step: Each position Mi,j is defined to be the MAXIMUM score at position i,j

$M_{i,j}$ = MAXIMUM **[**

$M_{i-1, j-1} + s_{i,j}$ (match/mismatch in the diagonal

$M_{i, j-1}$ + w (gap in sequence #1)

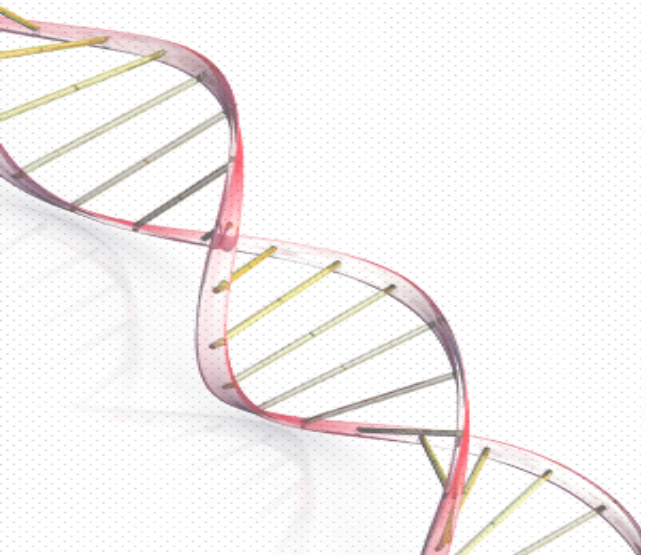
$M_{i-1, j}$ + w (gap in sequence #2)**]**

Match = +1
Mismatch = 0
Space = 0

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	1										
G	0											
A	0											
T	0											
C	0											
G	0											
A	0											

Another Example

- Fill in rest of row 1 and column 1



		G	A	A	T	T	T	C	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1	1	1
G	0	1										
A	0	1										
T	0	1										
C	0	1										
G	0	1										
A	0	1										

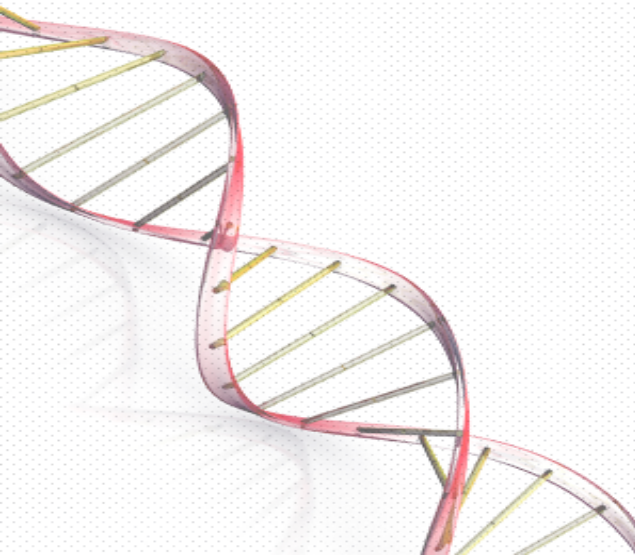
Match = +1
Mismatch = 0
Space = 0

Another Example

- Fill in column 2

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1	1	1
G	0	1	1									
A	0	1	2									
T	0	1	2									
C	0	1	2									
G	0	1	2									
A	0	1	2									

Match = +1
Mismatch = 0
Space = 0

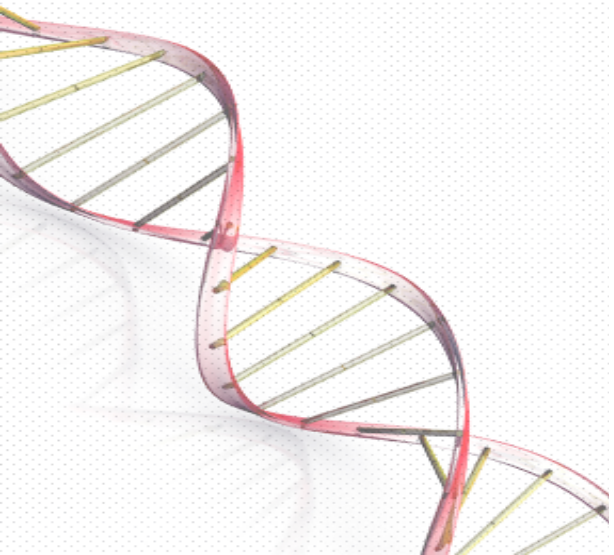


Another Example

- Fill in column 3

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1	2	2	2	2
A	0	1	2	2	2	2	2	2	2	2	2	3
T	0	1	2	2	3	3	3	3	3	3	3	3
C	0	1	2	2	3	3	3	4	4	4	4	4
G	0	1	2	2	3	3	3	4	4	5	5	5
A	0	1	2	3	3	3	3	4	5	5	5	**6**

Match = +1
Mismatch = 0
Space = 0



Another Example

- Traceback step:
  - Position at current cell and look at direct predecessors

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	
G	0	1	1	1	1	1	1	1	1	1	1	
G	0	1	1	1	1	1	1	1	2	2	2	
A	0	1	1	2	2	2	2	2	2	2	2	
T	0	1	2	2	3	3	3	3	3	3	3	
C	0	1	2	2	3	3	4	4	4	4	4	
G	0	1	2	2	3	3	4	4	5	5	5	
A												6

Seq # 1 A
|
Seq # 2 A

Another Example

- Traceback step:
  - Position at current cell and look at direct predecessors

```
Seq # 1    T A
              |
Seq # 2    -  A
Seq # 1    A
              |
Seq # 2    A
```

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0		
G	0	1	1	1	1	1	1	1	1	1		
G	0	1	1	1	1	1	1	1	2	2		
A	0	1	1	2	2	2	2	2	2	2		
T	0	1	2	2	3	3	3	3	3	3		
C	0	1	2	2	3	3	4	4	4	4		
G	0	1	2	2	3	3	4	4	5	5	5	
A												6

Another Example

- Traceback step:
  - Position at current cell and look at direct predecessors

```
Seq # 1   T  T A
                |
Seq # 2   -  -  A
Seq # 1    T A
               |
Seq # 2    - A
```

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0			
G	0	1	1	1	1	1	1	1	1			
G	0	1	1	1	1	1	1	1	2			
A	0	1	1	2	2	2	2	2	2			
T	0	1	2	2	3	3	3	3	3			
C	0	1	2	2	3	3	4	4	4			
G	0	1	2	2	3	3	4	4	5	5	5	
A												6

Another Example

- Traceback step:
  - Position at current cell and look at direct predecessors

```
Seq # 1   G T  T A
          |      |
Seq # 2   G  -  - A
Seq # 1   T  T A
                |
Seq # 2   -  -  A
```

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0				
G	0	1	1	1	1	1	1	1				
G	0	1	1	1	1	1	1	1				
A	0	1	1	2	2	2	2	2				
T	0	1	2	2	3	3	3	3				
C	0	1	2	2	3	3	4	4				
G									5	5	5	
A												6

Another Example

- Traceback step:
  - Position at current cell and look at direct predecessors

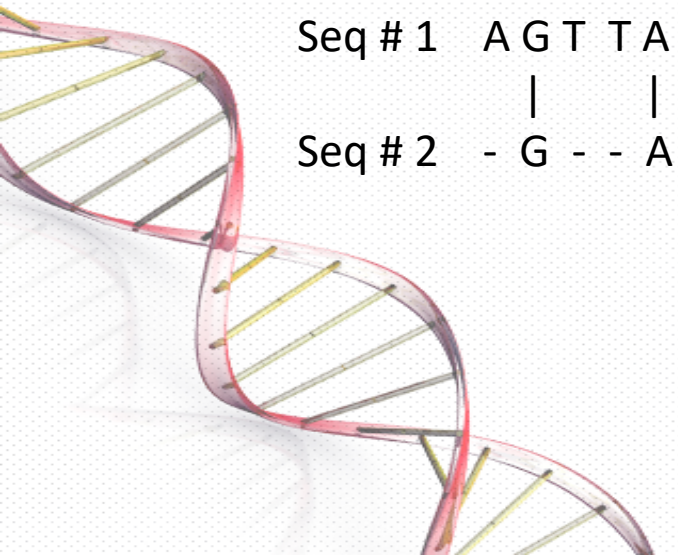
```
Seq # 1   A G T  T A
              |      |
Seq # 2   -  G  -  -  A

Seq # 1   G T  T A
              |      |
Seq # 2   G  -  -  A
```

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0					
G	0	1	1	1	1	1	1					
G	0	1	1	1	1	1	1					
A	0	1	1	2	2	2	2					
T	0	1	2	2	3	3	3					
C	0	1	2	2	3	3	4	4				
G									5	5	5	
A												6

Another Example

- Traceback step:
  - Position at current cell and look at direct predecessors

```
Seq # 1   C A G T  T A
          |   |      |
Seq # 2   C -  G  -  -  A
Seq # 1   A G T  T A
            |      |
Seq # 2   -  G  -  -  A
```

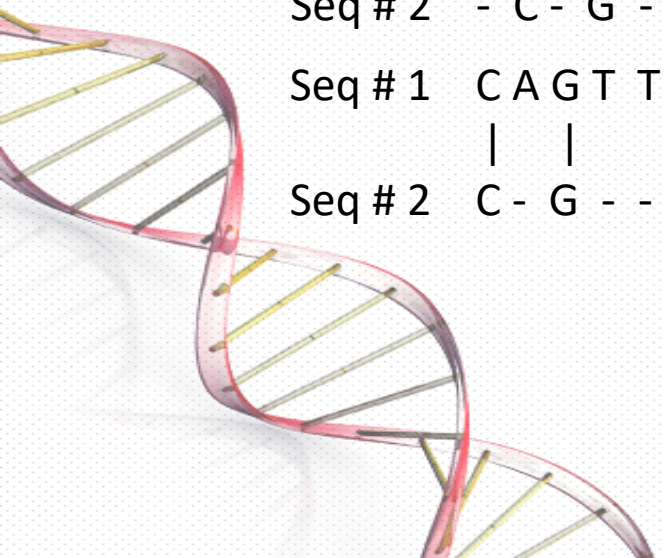


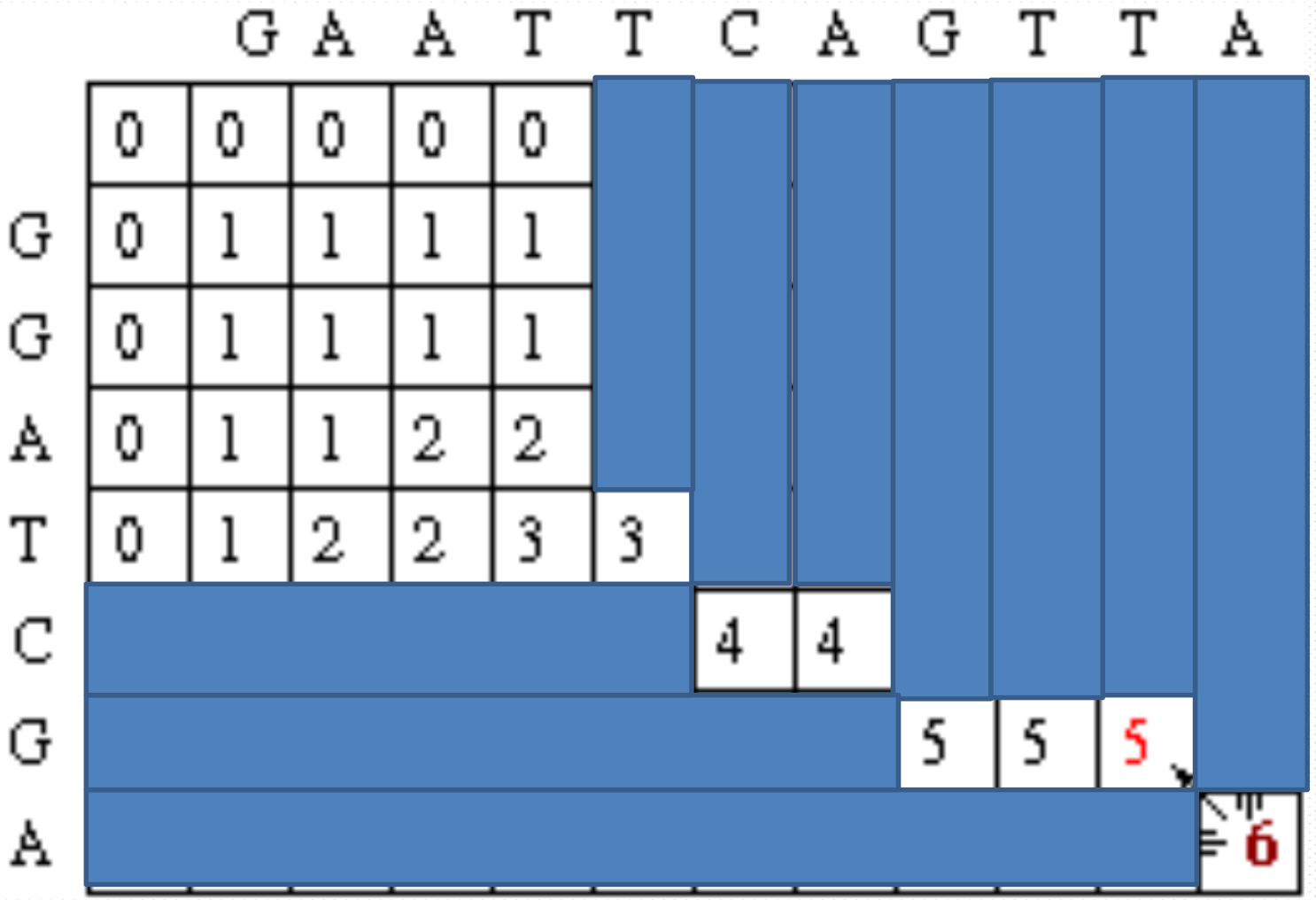
		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0						
G	0	1	1	1	1	1						
G	0	1	1	1	1	1						
A	0	1	1	2	2	2						
T	0	1	2	2	3	3						
C							4	4				
G									5	5	5	
A												6

Another Example

- Traceback step:
  - Position at current cell and look at direct predecessors

```
Seq # 1   T C A G T  T A
              |   |      |
Seq # 2   -  C -  G  -  -  A

Seq # 1   C A G T  T A
              |   |      |
Seq # 2   C -  G  -  -  A
```

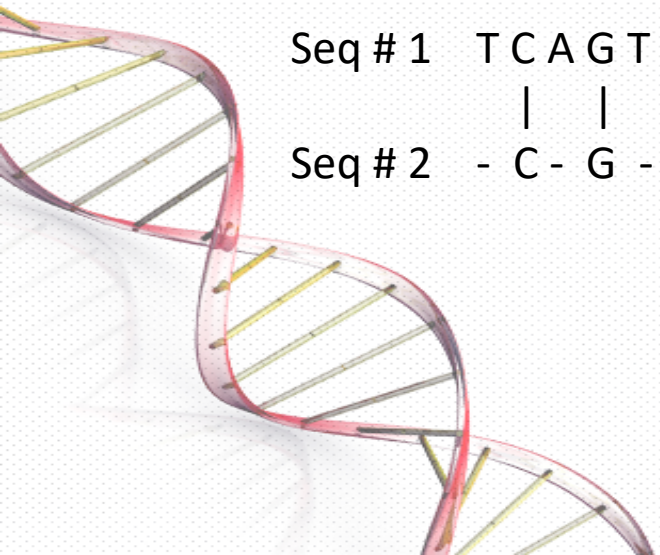


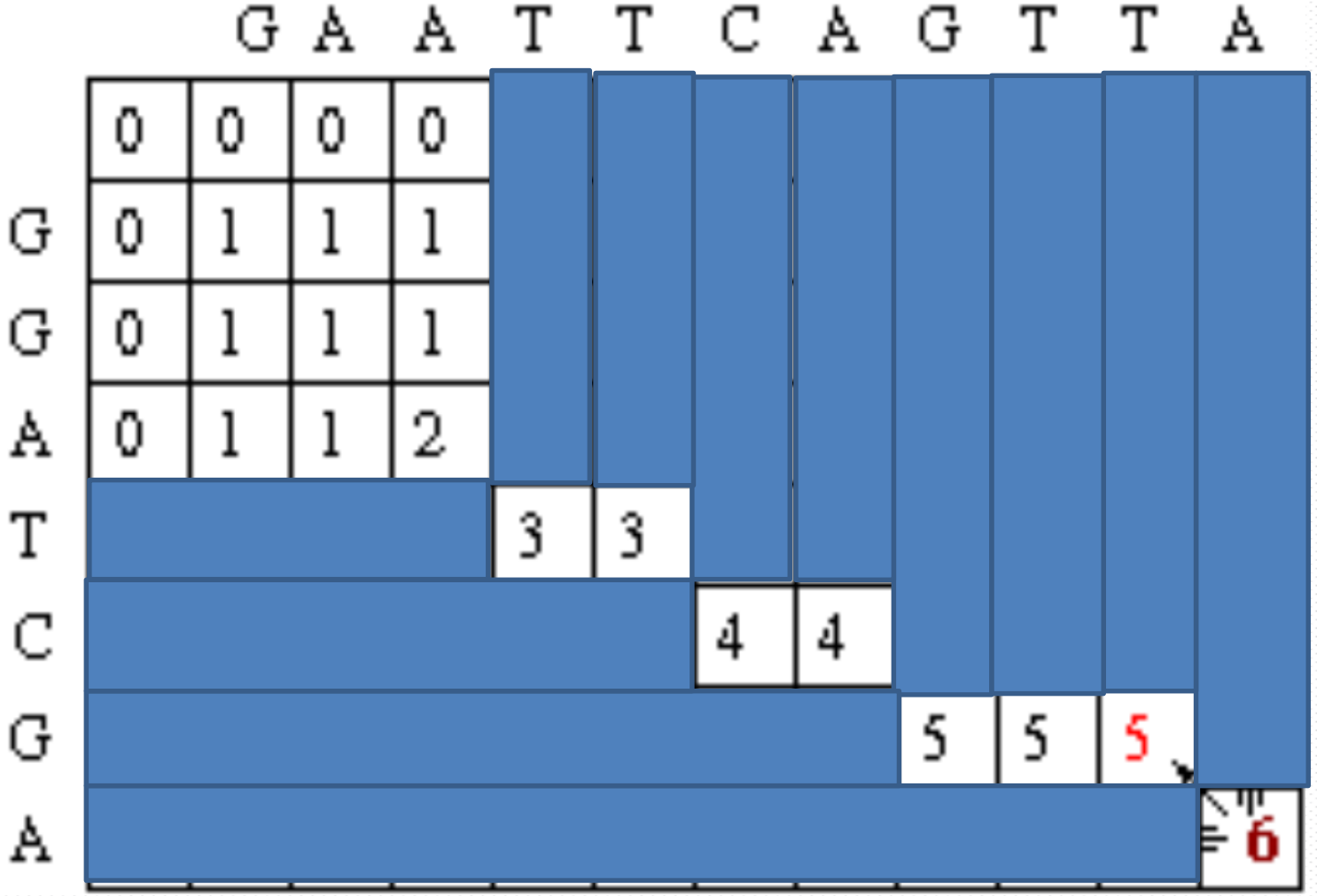


Another Example

- Traceback step:
  - Position at current cell and look at direct predecessors

```
Seq # 1   T T C A G T  T A
          |   |   |      |
Seq # 2   T -  C -  G  -  -  A

Seq # 1   T C A G T  T A
            |   |      |
Seq # 2   -  C -  G  -  -  A
```

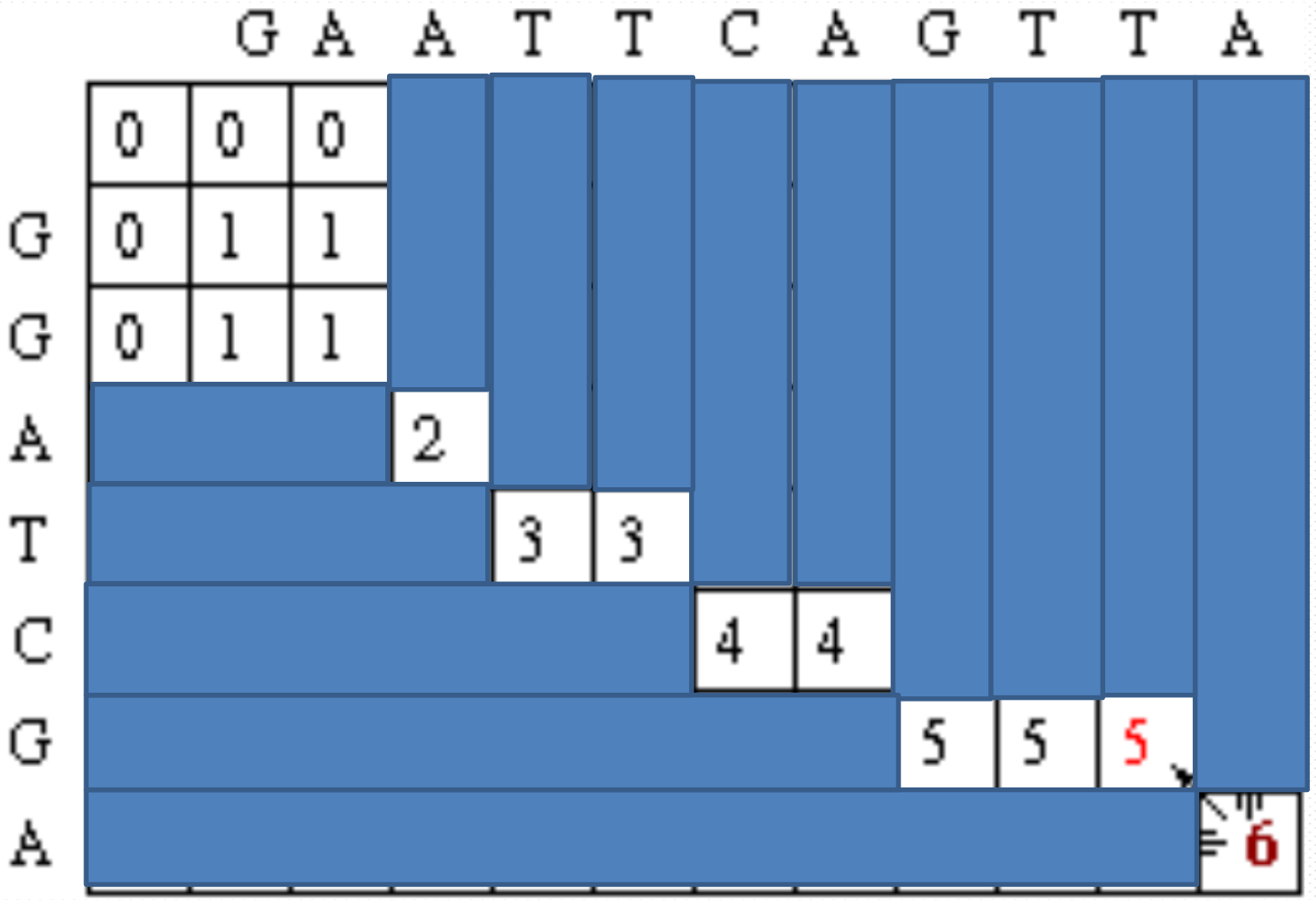




Another Example

- Traceback step:
  - Position at current cell and look at direct predecessors

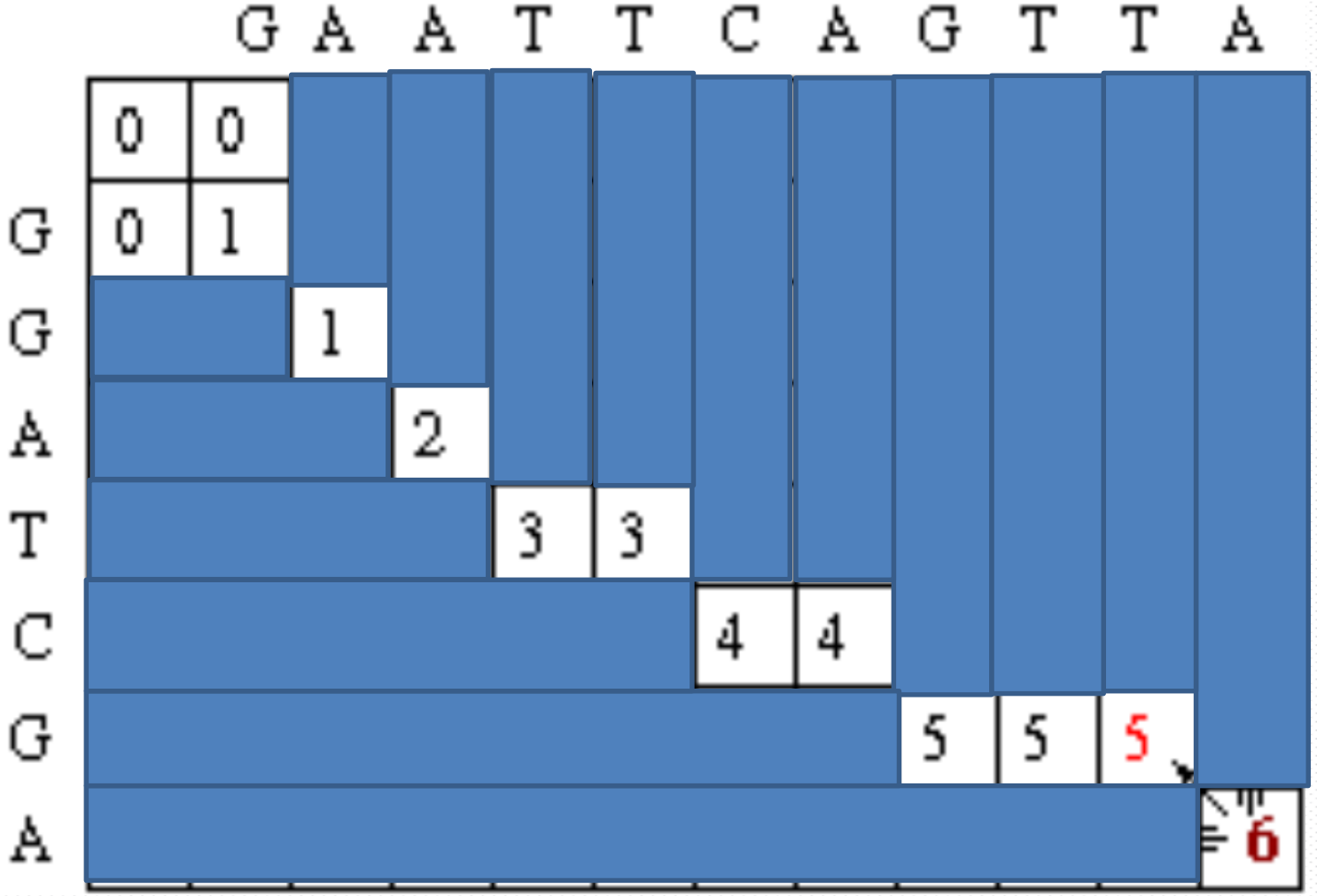
```
Seq # 1   A T T C A G T  T A
          | |   |   |      |
Seq # 2   A T -  C -  G  -  -  A
Seq # 1   T T C A G T  T A
          |   |   |      |
Seq # 2   T -  C -  G  -  -  A
```



Another Example

- Traceback step:
  - Position at current cell and look at direct predecessors

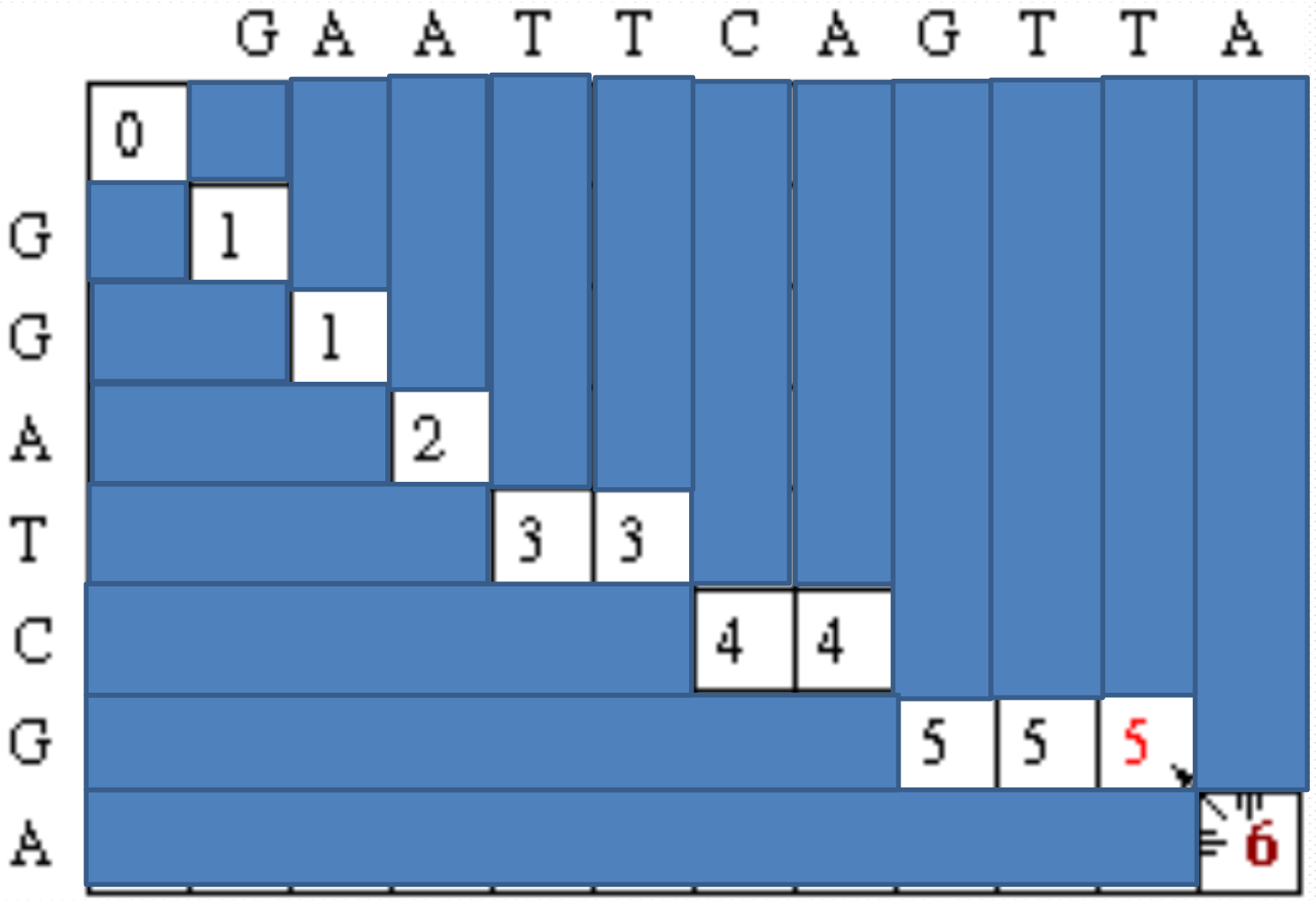
```
Seq # 1   A A T T C A G T  T A
            | |   |   |      |
Seq # 2   G A T -  C -  G  -  - A
Seq # 1   A T T C A G T  T A
            | |   |   |      |
Seq # 2   A T -  C -  G  -  -  A
```



Another Example

- Traceback step:
  - Position at current cell and look at direct predecessors

```
Seq # 1   G A A T T C A G T  T A
          |   | |  |   |      |
Seq # 2   G G A T -  C -  G  - - A

Seq # 1   A A T T C A G T  T A
            | |  |   |      |
Seq # 2   G A T -  C -  G  - - A
```



Pseudocode

```
for i=0 to length(A)
  F(i,0) ← d*i
for j=0 to length(B)
  F(0,j) ← d*j
for i=1 to length(A)
  for j=1 to length(B)
  {
    Match ← F(i-1,j-1) + S(Ai, Bj)
    Delete ← F(i-1, j) + d
    Insert ← F(i, j-1) + d
    F(i,j) ← max(Match, Insert,
Delete)
  }
```

Computational Complexity

- initialization: $O(m)$, $O(n)$ where sequence lengths are m, n
- filling in rest of matrix: $O(mn)$
- traceback: $O(m + n)$
- hence, if sequences have nearly same length, the computational complexity is

$$O(n^2)$$