

# String Matching

CS 4364/5364

# Exact String Matching

- Given string  $P$ , called the pattern, and a longer string  $T$ , called the text, the **exact matching** problem is to find all occurrences, if any, of  $P$  in  $T$ .
- Example:
  - $P = \text{"aba"}$ ,  $T = \text{"bbabaxababay"}$
  - $P$  occurs in  $T$  at positions: 3, 7, & 9
  - Note, that 2 occurrences overlap

# The naive method

bbabaxababay

aba

# The naive method

bbabaxababay  
aba



# The naive method

bbabaxababay

x

aba

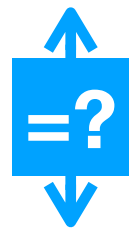
# The naive method

bbabaxababay

aba

# The naive method

bbabaxababay  
aba



# The naive method

bbabaxababay

x

aba




# The naive method

bbabaxababay

aba

# The naive method

bbabaxababay  
aba



# The naive method

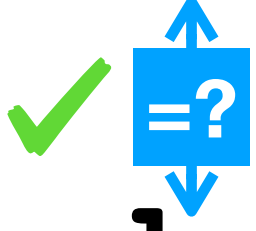
bbabaxababay



aba

# The naive method

bbabaxababay  
aba



# The naive method

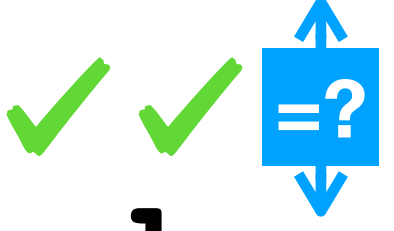
bbabaxababay



aba

# The naive method

bbabaxababay  
aba



# The naive method

bbabaxababay

✓✓✓

aba

# The naive method

bb**aba**xababay  
✓✓✓  
aba



# The naive method

bb**aba**xababay

aba

# The naive method

bb**aba**xababay  
    ↑  
    =?  
    ↓  
    aba

# The naive method

bb**aba**xababay

x

aba

# The naive method

Another example

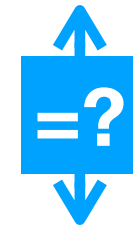
aaaaaaaaaaaaaa

aaa

# The naive method

Another example

aaaaaaaaaaaa



aaa

# The naive method

Another example

aaaaaaaaaaaaaa

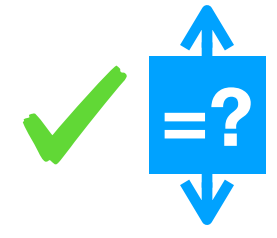


aaa

# The naive method

Another example

aaaaaaaaaaaaa



aaa

# The naive method

Another example

aaaaaaaaaaaaaa

✓✓

aaa



# The naive method

Another example

aaaaaaaaaaaaa  
✓ ✓ =?   
aaa

# The naive method

Another example

aaaaaaaaaaaaaa

✓✓✓

aaa

# The naive method

Another example

aaaaaaaaaaaa

✓✓✓

aaa

# The naive method

Another example

aaaaaa

aaa

# The naive method

Another example

aaaaaaa  
=?  
aaa

# The naive method

Another example

aaaaaaa



aaa

# The naive method

Another example

aaaaaa  
✓ =? aaa

# The naive method

Another example

aaaaaaaaaaaa

✓✓

aaa



# The naive method

Another example

aaa aaaaaaaaaa  
✓ ✓ =?   
aaa

# The naive method

Another example

aaaaaa

✓✓✓

aaa

# The naive method

Another example

aaa<sup>aa</sup>aaaaaaaa

✓✓✓

aaa

How many comparisons would the naive algorithm do on this example?

# The naive method

Another example

aaa<sup>aa</sup>aaaaaaaa

✓✓✓

aaa

How many comparisons would the naive algorithm do on this example?

What did we already know about this comparison before we started?

# Example of faster methods

**xabyabxyz**  
**abxyz**

# Example of faster methods

**xabyabxyz**  
**~~x~~abyabxyz**

# Example of faster methods

**xabyaxyabxz**

~~**xyabxz**~~

**axyabxz**

# Example of faster methods

xabxyabxyz

~~x~~abxyabxyz

abxyabx~~x~~



# Example of faster methods

xabyaxyabxz

~~x~~abyabxz

a**✓****✓****✓**y**✓**a**✓**b**✓**~~x~~

~~x~~abyabxz

# Example of faster methods

xabyaxyabxz

~~x~~abyabxz

a**✓****✓****✓****✓****✓****✓**~~x~~

~~x~~abyabxz

~~x~~abyabxz

# Example of faster methods

xabyaxyabxz

~~x~~abyabxz

a**✓****✓****✓****✓****✓****✓**~~x~~

~~x~~abyabxz

~~x~~abyabxz

~~x~~abyabxz

# Example of faster methods

xabyaxyabxz  
~~x~~abyabxz  
a**✓****✓****✓****✓****✓****✓**~~x~~  
~~x~~abyabxz  
~~x~~abyabxz  
~~x~~abyabxz  
a**✓****✓****✓****✓****✓****✓****✓**

# Example of faster methods

xabxyabxyz

~~x~~abxyabxyz

abxyabxyz

~~x~~abxyabxyz

~~x~~abxyabxyz

~~x~~abxyabxyz

abxyabxyz

xabxyabxyz

# Example of faster methods

xabxyabxyz  
xabxyabxyz  
  abxyabxyz  
    abxyabxyz  
      abxyabxyz  
        abxyabxyz  
          abxyabxyz  
            abxyabxyz

xabxyabxyz  
xabxyabxyz  
  abxyabxyz

# Example of faster methods

xabxyabxyz  
~~x~~abxyabxz  
a**✓****✓****✓****✓****✓****✓**~~x~~  
~~x~~abxyabxz  
~~x~~abxyabxz  
~~x~~abxyabxz  
a**✓****✓****✓****✓****✓****✓****✓**

xabxyabxyz  
~~x~~abxyabxz  
a**✓****✓****✓****✓****✓**~~x~~  
abxyabxz

# Example of faster methods

xabxyabxyz  
xabxyabxyz  
abxyabxyz  
xabxyabxyz  
xabxyabxyz  
xabxyabxyz  
abxyabxyz

xabxyabxyz  
xabxyabxyz  
abxyabxyz  
abxyabxyz



# Example of faster methods

xabxyabxyabxz  
x bxyabxz  
a bxyabxz  
x bxyabxz  
x bxyabxz  
x bxyabxz  
a bxyabxz

xabxyabxyabxz  
x bxyabxz  
a bxyabxz  
a bxyabxz

xabxyabxyabxz  
x bxyabxz  
a bxyabxz  
abxyabxz

# Example of faster methods

xabxyabxyabxz  
x bxyabxz  
a bxyabxz  
x bxyabxz  
x bxyabxz  
x bxyabxz  
a bxyabxz

xabxyabxyabxz  
x bxyabxz  
a bxyabxz  
a bxyabxz

xabxyabxyabxz  
x bxyabxz  
a bxyabxz  
a bxyabxz

# Example of faster methods

xabxyabxyz  
xabxyabxyz  
  abxyabxyz  
    abxyabxyz  
      abxyabxyz  
        abxyabxyz  
          abxyabxyz  
            abxyabxyz

xabxyabxyz  
xabxyabxyz  
  abxyabxyz  
    abxyabxyz  
      abxyabxyz

xabxyabxyz  
xabxyabxyz  
  abxyabxyz  
    abxyabxyz  
      abxyabxyz

If we knew ahead of time the comparisons we could skip (based on where in P the mismatches are), we could decrease the running time!

# Preprocessing

**Definition** Given a string  $S$  and a position  $i > 1$ , let  $Z_i(S)$  be the length of the longest substring of  $S$  that matches a prefix of  $S$ .

that is,  $Z_i(S) = j$  if  $S[1...j] = S[i...(i+j-1)]$  (and  $S[1...(j+1)] \neq S[i...(i+j)]$  )

for  $S = \text{"aabcaabxaaz"}$

$$Z_5(S) = 3$$

$$Z_6(S) = 1$$

$$Z_7(S) = Z_8(S) = 0$$

$$Z_9(S) = 2$$

# Preprocessing

**Definition** For any position  $i > 1$  where  $Z_i > 0$ , the **Z-box** at  $i$  is defined by the interval starting at  $i$  and ending at position  $i + Z_i - 1$



# Preprocessing

**Definition** For any position  $i > 1$  where  $Z_i > 0$ , the **Z-box** at  $i$  is defined by the interval starting at  $i$  and ending at position  $i + Z_i - 1$

**Definition** For every  $j > 1$ ,  $r_j$  is the right most endpoint of the Z-boxes that begin at or before position  $j$ .

Or:  $r_j$  is the largest value  $k + Z_k - 1$  over all  $0 < k \leq j$



# Preprocessing

**Definition** For any position  $i > 1$  where  $Z_i > 0$ , the **Z-box** at  $i$  is defined by the interval starting at  $i$  and ending at position  $i + Z_i - 1$

**Definition** For every  $j > 1$ ,  $r_j$  is the right most endpoint of the Z-boxes that begin at or before position  $j$ .

Or:  $r_j$  is the largest value  $k + Z_k - 1$  over all  $0 < k \leq j$

**Definition** For every  $j > 1$ ,  $l_j$  is the left most position  $k$  such that  $r_k = j$ .



# Preprocessing

- Calculation of all  $Z_i$  can be done in  $O(n)$  time (where  $|S| = n$ )



# Linear-time exact sequence matching

# Linear-time exact sequence matching

- Construct a new sequence  $S = P\$T$  where  $\$$  is a letter that's not in either of the two strings

# Linear-time exact sequence matching

- Construct a new sequence  $S = P\$T$  where  $\$$  is a letter that's not in either of the two strings
- What happens when we compute all of the  $Z$  values for  $S$ ?

# Linear-time exact sequence matching

- Construct a new sequence  $S = P\$T$  where  $\$$  is a letter that is not in either of the two strings
- What happens when we compute all of the  $Z$  values for  $S$ ?
- Do we need to save  $Z_{(n+5)}$ ? (where  $|P| = n$ )

# Boyer-Moore (at a high level)

- Worst-case running time is still  $O(m+n)$ , but in practice it is sub-linear
- Re-think the problem and look at suffixes of the pattern, then use some preprocessing information to skip the pattern ahead by more than 1

# Boyer-Moore (at a high level)

- The bad character rule
  - when matching (from the right) and a mismatch is found (say  $x$  in  $T \neq y$  in  $P$ )
  - shift the pattern so that the next instance of  $x$  in  $P$  that is to the left of the current position is at the current position in  $T$

**x**abxyabx**y**abxz  
abxyabx**x**

# Boyer-Moore (at a high level)

- The bad character rule
  - when matching (from the right) and a mismatch is found (say  $x$  in  $T \neq y$  in  $P$ )
  - shift the pattern so that the next instance of  $x$  in  $P$  that is to the left of the current position is at the current position in  $T$

**xabxyabx**y**abxz**  
**abx**y**abxz**

# Boyer-Moore (at a high level)

- The bad character rule
  - when matching (from the right) and a mismatch is found (say  $x$  in  $T \neq y$  in  $P$ )
  - shift the pattern so that the next instance of  $x$  in  $P$  that is to the left of the current position is at the current position in  $T$

**xabxyabx****y****abxz**  
**abx****y****abxz**



# Boyer-Moore (at a high level)

- The bad character rule
  - when matching (from the right) and a mismatch is found (say  $x$  in  $T \neq y$  in  $P$ )
  - shift the pattern so that the next instance of  $x$  in  $P$  that is to the left of the current position is at the current position in  $T$

**xabxyabxyabxz**  
**abxyabxz**

**Does this skip any instances of  $P$  in  $T$ ?**

# Boyer-Moore (at a high level)

- The good suffix rule
  - suppose a substring of  $T$ ,  $t$  matches a suffix of  $P$  but a mismatch occurs at the next character
  - find the right most copy of  $t$  in  $P$  such that the preceding character is different
  - shift  $P$  so that this substring is matched with  $t$

**prstabst**u**abvqxr**st****  
**qcabdab**d**ab**

# Boyer-Moore (at a high level)

- The good suffix rule
  - suppose a substring of  $T$ ,  $t$  matches a suffix of  $P$  but a mismatch occurs at the next character
  - find the right most copy of  $t$  in  $P$  such that the preceding character is different
  - shift  $P$  so that this substring is matched with  $t$

**prstabst**u**abvqxr**s**t**  
**qc**a**bdab**d**ab**

# Boyer-Moore (at a high level)

- The good suffix rule
  - suppose a substring of  $T$ ,  $t$  matches a suffix of  $P$  but a mismatch occurs at the next character
  - find the right most copy of  $t$  in  $P$  such that the preceding character is different
  - shift  $P$  so that this substring is matched with  $t$

**prstabst**u**abvqxr**s**t**

**qc**a**bdab**d**ab**

# Boyer-Moore (at a high level)

- The good suffix rule
  - suppose a substring of  $T$ ,  $t$  matches a suffix of  $P$  but a mismatch occurs at the next character
  - find the right most copy of  $t$  in  $P$  such that the preceding character is different
  - shift  $P$  so that this substring is matched with  $t$

**prstabst**u**ab**v**qxr**s**t**  
**qc**a**bdab**d**ab**

**Does this skip any instances of  $P$  in  $T$ ?**

# Other methods

- Knuth-Morris-Pratt (KMP)
  - able to work with  $T$  arriving **online**
  - running time is still  $O(m+n)$
  - uses pattern suffixes (closer to suffix tree which we will see next)
- Aho-Corasick
  - efficiently finds all occurrences of a **set** of patterns
  - puts patterns into a tree to search all at once
- Karp-Rabin
  - uses bit operations in place of character comparisons
  - structure similar to on-hot-encodings to represent strings

# Edit Distance

- Up to now, need the exact string in a set
- What if I wanted to know how similar two strings are?

# Edit Distance

- Up to now, need the exact string in a set
- What if I wanted to know how similar two strings are?
- Problem: Given strings  $S_1$  and  $S_2$  what is the minimum number of edits (insertions, deletions, replacements) needed to convert  $S_1$  into  $S_2$ ?



# Edit Distance

- Up to now, need the exact string in a set
- What if I wanted to know how similar two strings are?
- Problem: Given strings  $S_1$  and  $S_2$  what is the minimum number of edits (insertions, deletions, replacements) needed to convert  $S_1$  into  $S_2$ ?
- Example:  $S_1 = \mathbf{baseball}$  &  $S_2 = \mathbf{ballcap}$ .
  - RRR DR  
**baseball**  
**ballca p**
  - 5 operations: change s→l, e→l, b→c, delete l, l→p

# Global Alignment Problem

- An **alignment** of two sequences is formed by inserting gap characters, '-', in arbitrary locations along the sequences so that they end up with the same length and there are no two spaces at the same position of the two augmented strings.

```
baseball---  
----ballcap
```

# Global Alignment Problem

- An **alignment** of two sequences is formed by inserting gap characters, '-', in arbitrary locations along the sequences so that they end up with the same length and there are no two spaces at the same position of the two augmented strings.

```
baseball---  
----ballcap
```

```
baseball  
ballca-p
```

# Global Alignment Problem

- An **alignment** of two sequences is formed by inserting gap characters, '-', in arbitrary locations along the sequences so that they end up with the same length and there are no two spaces at the same position of the two augmented strings.

baseball  
-ballcap

baseball---  
----ballcap

baseball  
ballca-p

# Global Alignment Problem

- An **alignment** of two sequences is formed by inserting gap characters, '-', in arbitrary locations along the sequences so that they end up with the same length and there are no two spaces at the same position of the two augmented strings.

baseball  
-ballcap

baseball---  
----ballcap

baseball  
ballca-p

**How do we know which one of these is best?**

# Alignment

- In general, associate a similarity score with each pair of aligned characters:

- for characters  $x, y \in \Sigma \cup \{-\}$ , let  $\delta(x, y)$  be the similarity of  $x$  and  $y$

- Let the score,  $\Delta$ , of an alignment,  $A = (S'_1, S'_2)$ , be defined as  $\Delta(A) =: \sum_{1 \leq i \leq |S'_1|} \delta(S'_1[i], S'_2[i])$

- Goal of alignment is to maximize that sum

# Alignment

- In general, associate a similarity score with each pair of aligned characters:
  - for characters  $x, y \in \Sigma \cup \{-\}$ , let  $\delta(x, y)$  be the similarity of  $x$  and  $y$

• Let the score,  $\Delta$ , of an alignment,  $A = (S'_1, S'_2)$ , be defined as  $\Delta(A) =: \sum_{1 \leq i \leq |S'_1|} \delta(S'_1[i], S'_2[i])$

- Goal of alignment is to maximize that sum

```
baseball  
ballca-p
```

# Alignment

- In general, associate a similarity score with each pair of aligned characters:
  - for characters  $x, y \in \Sigma \cup \{-\}$ , let  $\delta(x, y)$  be the similarity of  $x$  and  $y$

Let the score,  $\Delta$ , of an alignment,  $A = (S'_1, S'_2)$ , be defined as  $\Delta(A) =: \sum_{1 \leq i \leq |S'_1|} \delta(S'_1[i], S'_2[i])$

- Goal of alignment is to maximize that sum

	-	a	b	c	e	l	p	s
-		-1	-1	-1	-1	-1	-1	-1
a	-1	0	-1	-1	-1	-1	-1	-1
b	-1	-1	0	-1	-1	-1	-1	-1
c	-1	-1	-1	0	-1	-1	-1	-1
e	-1	-1	-1	-1	0	-1	-1	-1
l	-1	-1	-1	-1	-1	0	-1	-1
p	-1	-1	-1	-1	-1	-1	0	-1
s	-1	-1	-1	-1	-1	-1	-1	0

baseball  
ballca-p



# Alignment

- In general, associate a similarity score with each pair of aligned characters:
  - for characters  $x, y \in \Sigma \cup \{-\}$ , let  $\delta(x, y)$  be the similarity of  $x$  and  $y$

• Let the score,  $\Delta$ , of an alignment,  $A = (S'_1, S'_2)$ , be defined as  $\Delta(A) =: \sum_{1 \leq i \leq |S'_1|} \delta(S'_1[i], S'_2[i])$

- Goal of alignment is to maximize that sum

**How can we compute an alignment that optimizes  $\Delta$ ?**

	-	a	b	c	e	l	p	s
-	0	0	0	0	0	0	0	0
a	0	2	-1	-1	-1	-1	-1	-1
b	0	-1	2	-1	-1	-1	-1	-1
c	0	-1	-1	2	-1	-1	-1	-1
e	0	-1	-1	-1	2	-1	-1	-1
l	0	-1	-1	-1	-1	2	-1	-1
p	0	-1	-1	-1	-1	-1	2	-1
s	0	-1	-1	-1	-1	-1	-1	2

# Alignment

- In general, associate a similarity score with each pair of aligned characters:
  - for characters  $x, y \in \Sigma \cup \{-\}$ , let  $\delta(x, y)$  be the similarity of  $x$  and  $y$

Let the score,  $\Delta$ , of an alignment,  $A = (S'_1, S'_2)$ , be defined as  $\Delta(A) =: \sum_{1 \leq i \leq |S'_1|} \delta(S'_1[i], S'_2[i])$

- Goal of alignment is to maximize that sum

How can we compute an alignment that optimizes  $\Delta$ ?

	-	a	b	c	e	l	p	s
-	0	0	0	0	0	0	0	0
a	0	2	-1	-1	-1	-1	-1	-1
b	0	-1	2	-1	-1	-1	-1	-1
c	0	-1	-1	2	-1	-1	-1	-1
e	0	-1	-1	-1	2	-1	-1	-1
l	0	-1	-1	-1	-1	2	-1	-1
p	0	-1	-1	-1	-1	-1	2	-1
s	0	-1	-1	-1	-1	-1	-1	2

baseball---  
 ----ballcap

# Needleman-Wunsch

- brute-force, compute all possible alignments and score them, would take exponential time to compute the optimal alignment
- using dynamic programming Needleman and Wunsch [1970]<sup>1</sup> found that the optimal alignment can be computed in  $O(mn)$ -time.

<sup>1</sup>Notice we have gone back in time, suffix trees were introduced in 1973, Ukkonen's algorithm was 1995, Suffix Arrays were 1993

# Needleman-Wunsch

- Dynamic programming, generally, works by solving sub-problems, storing the results, and combining the solution to (most times) multiple sub-problems to find the answer.
- Given two strings  $S[1\dots n]$  and  $T[1\dots m]$ , find the best alignment.

# Needleman-Wunsch

- Dynamic programming, generally, works by solving sub-problems, storing the results, and combining the solution to (most times) multiple sub-problems to find the answer.
- Given two strings  $S[1\dots n]$  and  $T[1\dots m]$ , find the best alignment given the best alignments of:
  - $S[1\dots(n-1)]$  and  $T[1\dots m]$ ,
  - $S[1\dots n]$  and  $T[1\dots(m-1)]$ , and
  - $S[1\dots(n-1)]$  and  $T[1\dots(m-1)]$

# Needleman-Wunsch

- Define an  $n \times m$  array  $V$ , the cell  $V(i,j)$  will hold the score of the best sub alignments of  $S[1...i]$  and  $T[1...j]$

# Needleman-Wunsch

- Define an  $n \times m$  array  $V$ , the cell  $V(i, j)$  will hold the score of the best sub alignments of  $S[1 \dots i]$  and  $T[1 \dots j]$

- The recurrence relation (the base of any DP)

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$

# Needleman-Wunsch

- Define an  $n \times m$  array  $V$ , the cell  $V(i, j)$  will hold the score of the best sub alignments of  $S[1 \dots i]$  and  $T[1 \dots j]$

- The recurrence relation (the base of any DP)

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$

- The initialization is:

$$V(0, 0) = 0$$

$$V(0, j) = V(0, j-1) + \delta(-, T[j])$$

$$V(i, 0) = V(i-1, 0) + \delta(S[i], -)$$



# Needleman-Wunsch

- Define an  $n \times m$  array  $V$ , the cell  $V(i, j)$  will hold the score of the best sub alignments of  $S[1 \dots i]$  and  $T[1 \dots j]$

- The recurrence relation (the base of any DP)

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$

- The initialization is:

$$V(0, 0) = 0$$

$$V(0, j) = V(0, j-1) + \delta(-, T[j])$$

$$V(i, 0) = V(i-1, 0) + \delta(S[i], -)$$

Optimal alignment score is in  $V(n, m)$

# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

	A	A	C	C	C	G
	0					
A	-1					
A						
G						
G						
C						
C						

The cost of the optimal alignment of "A" and ""

# Needleman-Wunch

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

	A	A	C	C	C	G
	0					
A	-1					
A	-2					
G						
G						
C						
C						

The cost of the optimal alignment of "AA" and ""

# Needleman-Wunch

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

		A	A	C	C	C	G
	0	-1					
A	-1						
A	-2						
G	-3						
G	-4						
c	-5						
c	-6						

The cost of the optimal alignment of "" and "A"

# Needleman-Wunch

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	0	-1	-2	-3	-4	-5	-6
<b>A</b>	-1						
<b>A</b>	-2						
<b>G</b>	-3						
<b>G</b>	-4						
<b>c</b>	-5						
<b>c</b>	-6						

The cost of the optimal alignment of "A" and "A"

# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	<b>0</b>	-1	-2	-3	-4	-5	-6
<b>A</b>	-1						
<b>A</b>	-2						
<b>G</b>	-3						
<b>G</b>	-4						
<b>C</b>	-5						
<b>C</b>	-6						

# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$



		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>	
		0	-1	-2	-3	-4	-5	-6
<b>A</b>		-1						
<b>A</b>		-2						
<b>G</b>		-3						
<b>G</b>		-4						
<b>C</b>		-5						
<b>C</b>		-6						

# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(1, 0) + \delta(-, A)$$

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$



		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>	
		0	-1	-2	-3	-4	-5	-6
<b>A</b>	-1							
<b>A</b>	-2							
<b>G</b>	-3							
<b>G</b>	-4							
<b>C</b>	-5							
<b>C</b>	-6							



# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(1, 0) + \delta(-, A)$$

**A**

-

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$



		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>	
		0	-1	-2	-3	-4	-5	-6
<b>A</b>	-1							
<b>A</b>	-2							
<b>G</b>	-3							
<b>G</b>	-4							
<b>C</b>	-5							
<b>C</b>	-6							

# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(1, 0) + \delta(-, A)$$

$$\begin{array}{c} \mathbf{A} \quad - \\ - \quad + \quad \mathbf{A} \end{array}$$

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$



		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>	
		0	-1	-2	-3	-4	-5	-6
<b>A</b>	-1							
<b>A</b>	-2							
<b>G</b>	-3							
<b>G</b>	-4							
<b>C</b>	-5							
<b>C</b>	-6							

# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(1, 0) + \delta(-, A) = -2$$

$$\begin{array}{c} \mathbf{A} \quad - \\ - \quad + \quad \mathbf{A} \end{array}$$

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$



		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	<b>0</b>	-1	-2	-3	-4	-5	-6
<b>A</b>	-1						
<b>A</b>	-2						
<b>G</b>	-3						
<b>G</b>	-4						
<b>C</b>	-5						
<b>C</b>	-6						

# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$


$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(0, 1) + \delta(A, -)$$

$$\begin{array}{c} - \\ \mathbf{A} \end{array} + \begin{array}{c} \mathbf{A} \\ - \end{array}$$

$$V(1, 0) + \delta(-, A) = -2$$

$$\begin{array}{c} \mathbf{A} \\ - \end{array} + \begin{array}{c} - \\ \mathbf{A} \end{array}$$

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[i]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$


		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	0	-1	-2	-3	-4	-5	-6
<b>A</b>	-1						
<b>A</b>	-2						
<b>G</b>	-3						
<b>G</b>	-4						
<b>C</b>	-5						
<b>C</b>	-6						

# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$


$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(0, 1) + \delta(A, -) = -2$$

$$\begin{array}{c} - \\ \mathbf{A} \end{array} + \begin{array}{c} \mathbf{A} \\ - \end{array}$$

$$V(1, 0) + \delta(-, A) = -2$$

$$\begin{array}{c} \mathbf{A} \\ - \end{array} + \begin{array}{c} - \\ \mathbf{A} \end{array}$$

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[i]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$


		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	0	-1	-2	-3	-4	-5	-6
<b>A</b>	-1						
<b>A</b>	-2						
<b>G</b>	-3						
<b>G</b>	-4						
<b>C</b>	-5						
<b>C</b>	-6						

# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(0,0) + \delta(A,A)$$

$$+ \begin{matrix} \mathbf{A} \\ + \\ \mathbf{A} \end{matrix}$$

$$V(0,1) + \delta(A,-) = -2$$

$$\begin{matrix} - & \mathbf{A} \\ \mathbf{A} & + \\ & - \end{matrix}$$

$$V(1,0) + \delta(-,A) = -2$$

$$\begin{matrix} \mathbf{A} & - \\ - & + \\ & \mathbf{A} \end{matrix}$$

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \leftarrow \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>	
		0	-1	-2	-3	-4	-5	-6
<b>A</b>		-1						
<b>A</b>		-2						
<b>G</b>		-3						
<b>G</b>		-4						
<b>C</b>		-5						
<b>C</b>		-6						

# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(0,0) + \delta(A,A) = 1$$

$$+ \begin{matrix} \mathbf{A} \\ \mathbf{A} \end{matrix}$$

$$V(0,1) + \delta(A,-) = -2$$

$$\begin{matrix} - & \mathbf{A} \\ \mathbf{A} & + \\ & - \end{matrix}$$

$$V(1,0) + \delta(-,A) = -2$$

$$\begin{matrix} \mathbf{A} & - \\ - & + \\ & \mathbf{A} \end{matrix}$$

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[i]) & \text{match/mismatch} \leftarrow \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	<b>0</b>	-1	-2	-3	-4	-5	-6
<b>A</b>	-1						
<b>A</b>	-2						
<b>G</b>	-3						
<b>G</b>	-4						
<b>C</b>	-5						
<b>C</b>	-6						

# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(0,0) + \delta(A,A) = 1$$

$$+ \begin{matrix} \mathbf{A} \\ \mathbf{A} \end{matrix}$$

$$V(0,1) + \delta(A,-) = -2$$

$$- \begin{matrix} \mathbf{A} \\ \mathbf{A} \end{matrix} + \begin{matrix} \mathbf{A} \\ - \end{matrix}$$

$$V(1,0) + \delta(-,A) = -2$$

$$\begin{matrix} \mathbf{A} \\ - \end{matrix} + \begin{matrix} - \\ \mathbf{A} \end{matrix}$$

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	<b>0</b>	-1	-2	-3	-4	-5	-6
<b>A</b>	-1						
<b>A</b>	-2						
<b>G</b>	-3						
<b>G</b>	-4						
<b>C</b>	-5						
<b>C</b>	-6						



# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(0,0) + \delta(A,A) = 1$$

$$+ \begin{matrix} \mathbf{A} \\ \mathbf{A} \end{matrix}$$

$$V(0,1) + \delta(A,-) = -2$$

$$- \begin{matrix} \mathbf{A} \\ \mathbf{A} \end{matrix} + \begin{matrix} \mathbf{A} \\ - \end{matrix}$$

$$V(1,0) + \delta(-,A) = -2$$

$$\begin{matrix} \mathbf{A} \\ - \end{matrix} + \begin{matrix} - \\ \mathbf{A} \end{matrix}$$

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	<b>0</b>	-1	-2	-3	-4	-5	-6
<b>A</b>	-1	1					
<b>A</b>	-2						
<b>G</b>	-3						
<b>G</b>	-4						
<b>C</b>	-5						
<b>C</b>	-6						

# Needleman-Wunch

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	0	-1	-2	-3	-4	-5	-6
<b>A</b>	-1	1					
<b>A</b>	-2						
<b>G</b>	-3						
<b>G</b>	-4						
<b>c</b>	-5						
<b>c</b>	-6						

Diagram illustrating the Needleman-Wunch algorithm. The table shows the sequence of characters: A, A, C, C, C, G. The values in the table represent the Needleman-Wunch score for each character at each position. The score for the first 'A' is 1, and the score for the second 'A' is -1. The score for the first 'C' is -2, and the score for the second 'C' is -3. The score for the third 'C' is -4, and the score for the 'G' is -5. The score for the final 'G' is -6. The diagram highlights the transition from the first 'A' to the second 'A' with a green arrow labeled  $+\delta(A,A)$  and a red arrow labeled  $+\delta(-,A)$ . A blue arrow labeled  $+\delta(A,-)$  points from the first 'A' to the second 'A'.

# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	0	-1	-2	-3	-4	-5	-6
<b>A</b>	-1	1 $+\delta(A,A)$	0 $+\delta(A,-)$				
<b>A</b>	-2						
<b>G</b>	-3						
<b>G</b>	-4						
<b>c</b>	-5						
<b>c</b>	-6						

Annotations in the table:
 

- A green arrow points from the cell (A, A) to the cell (A, A) with label  $+\delta(A,A)$ .
- A blue arrow points from the cell (A, A) to the cell (A, A) with label  $+\delta(A,-)$ .
- A red arrow points from the cell (A, A) to the cell (A, A) with label  $+\delta(-,A)$ .

# Needleman-Wunch

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

	A	A	C	C	C	G	
	0	-1	-2	-3	-4	-5	-6
A	-1	1	0				
A	-2						
G	-3						
G	-4						
c	-5						
c	-6						

Diagram illustrating the Needleman-Wunch algorithm. The grid shows the sequence of characters: A, A, C, C, C, G. The values in the grid represent the Needleman-Wunch score at each position. The value 0 is highlighted at the third position (C). Annotations show the transitions: a green arrow labeled  $+\delta(A,C)$  points from the value -2 to 0, a blue arrow labeled  $+\delta(A,-)$  points from the value -1 to 0, and a red arrow labeled  $+\delta(-,C)$  points from the value -3 to 0.

# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

		A	A	C	C	C	G
	0	-1	-2	-3	-4	-5	-6
A	-1	1	0	-1			
A	-2						
G	-3						
G	-4						
c	-5						
c	-6						

Annotations on the table:
 

- A green arrow points from the cell (A, A) to the cell (A, C) with the label  $+\delta(A, C)$ .
- A red arrow points from the cell (A, C) to the cell (A, -) with the label  $+\delta(-, C)$ .
- A blue arrow points from the cell (A, C) to the cell (A, -) with the label  $+\delta(A, -)$ .

# Needleman-Wunch

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>	
	0	-1	-2	-3	-4	-5	-6
<b>A</b>	-1	1	0	-1	-2	-3	-4
<b>A</b>	-2	0	2	1	0	-1	-2
<b>G</b>	-3	-1	1	1	0	-1	0
<b>G</b>	-4	-2	0	0	0	-1	0
<b>c</b>	-5	-3	-1	1	1	1	0
<b>c</b>	-6	-4	-2	0	2	2	1

# Needleman-Wunch

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	<b>0</b>	-1	-2	-3	-4	-5	-6
<b>A</b>	-1	$+\delta(A,A)$	$+\delta(-,A)$				
<b>A</b>	-2	$+\delta(A,-)$					
<b>G</b>	-3						
<b>G</b>	-4						
<b>c</b>	-5						
<b>c</b>	-6						

# Needleman-Wunch

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	<b>0</b>	-1	-2	-3	-4	-5	-6
<b>A</b>	-1	1					
<b>A</b>	-2						
<b>G</b>	-3						
<b>G</b>	-4						
<b>c</b>	-5						
<b>c</b>	-6						

Annotations on the table:
 

- Green arrow from (0, -1) to (1, 0) labeled  $+\delta(A, A)$
- Red arrow from (1, 0) to (0, 1) labeled  $+\delta(-, A)$
- Blue arrow from (1, 0) to (0, 0) labeled  $+\delta(A, -)$



# Needleman-Wunch

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	<b>0</b>	-1	-2	-3	-4	-5	-6
<b>A</b>	-1	1					
<b>A</b>	-2						
<b>G</b>	-3						
<b>G</b>	-4						
<b>c</b>	-5						
<b>c</b>	-6						

# Needleman-Wunch

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	<b>0</b>	-1	-2	-3	-4	-5	-6
<b>A</b>	-1	1	0	-1	-2	-3	-4
<b>A</b>	-2	0	2	1	0	-1	-2
<b>G</b>	-3	-1	1	1	0	-1	0
<b>G</b>	-4	-2	0	0	0	-1	0
<b>c</b>	-5	-3	-1	1	1	1	0
<b>c</b>	-6	-4	-2	0	2	2	1

# Needleman-Wunch

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	<b>0</b>	-1	-2	-3	-4	-5	-6
<b>A</b>	-1	1	0	-1	-2	-3	-4
<b>A</b>	-2	0	2	1	0	-1	-2
<b>G</b>	-3	-1	1	1	0	-1	0
<b>G</b>	-4	-2	0	0	0	-1	0
<b>c</b>	-5	-3	-1	1	1	1	0
<b>c</b>	-6	-4	-2	0	2	2	1

# Needleman-Wunch

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	<b>0</b>	-1	-2	-3	-4	-5	-6
<b>A</b>	-1	1	0	-1	-2	-3	-4
<b>A</b>	-2	0	2	1	0	-1	-2
<b>G</b>	-3	-1	1	1	0	-1	0
<b>G</b>	-4	-2	0	0	0	-1	0
<b>c</b>	-5	-3	-1	1	1	1	0
<b>c</b>	-6	-4	-2	0	2	<u>2</u>	1

# Needleman-Wunch

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

G  
-

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	<b>0</b>	-1	-2	-3	-4	-5	-6
<b>A</b>	-1	1	0	-1	-2	-3	-4
<b>A</b>	-2	0	2	1	0	-1	-2
<b>G</b>	-3	-1	1	1	0	-1	0
<b>G</b>	-4	-2	0	0	0	-1	0
<b>c</b>	-5	-3	-1	1	1	1	0
<b>c</b>	-6	-4	-2	0	2	<u>2</u>	1

# Needleman-Wunch

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

G  
-

		A	A	C	C	C	G
	0	-1	-2	-3	-4	-5	-6
A	-1	1	0	-1	-2	-3	-4
A	-2	0	2	1	0	-1	-2
G	-3	-1	1	1	0	-1	0
G	-4	-2	0	0	0	-1	0
c	-5	-3	-1	1	1	1	0
c	-6	-4	-2	0	2	2	1

# Needleman-Wunch

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

C G  
C -

		A	A	C	C	C	G
	0	-1	-2	-3	-4	-5	-6
A	-1	1	0	-1	-2	-3	-4
A	-2	0	2	1	0	-1	-2
G	-3	-1	1	1	0	-1	0
G	-4	-2	0	0	0	-1	0
c	-5	-3	-1	1	1	1	0
c	-6	-4	-2	0	2	2	1

# Needleman-Wunch

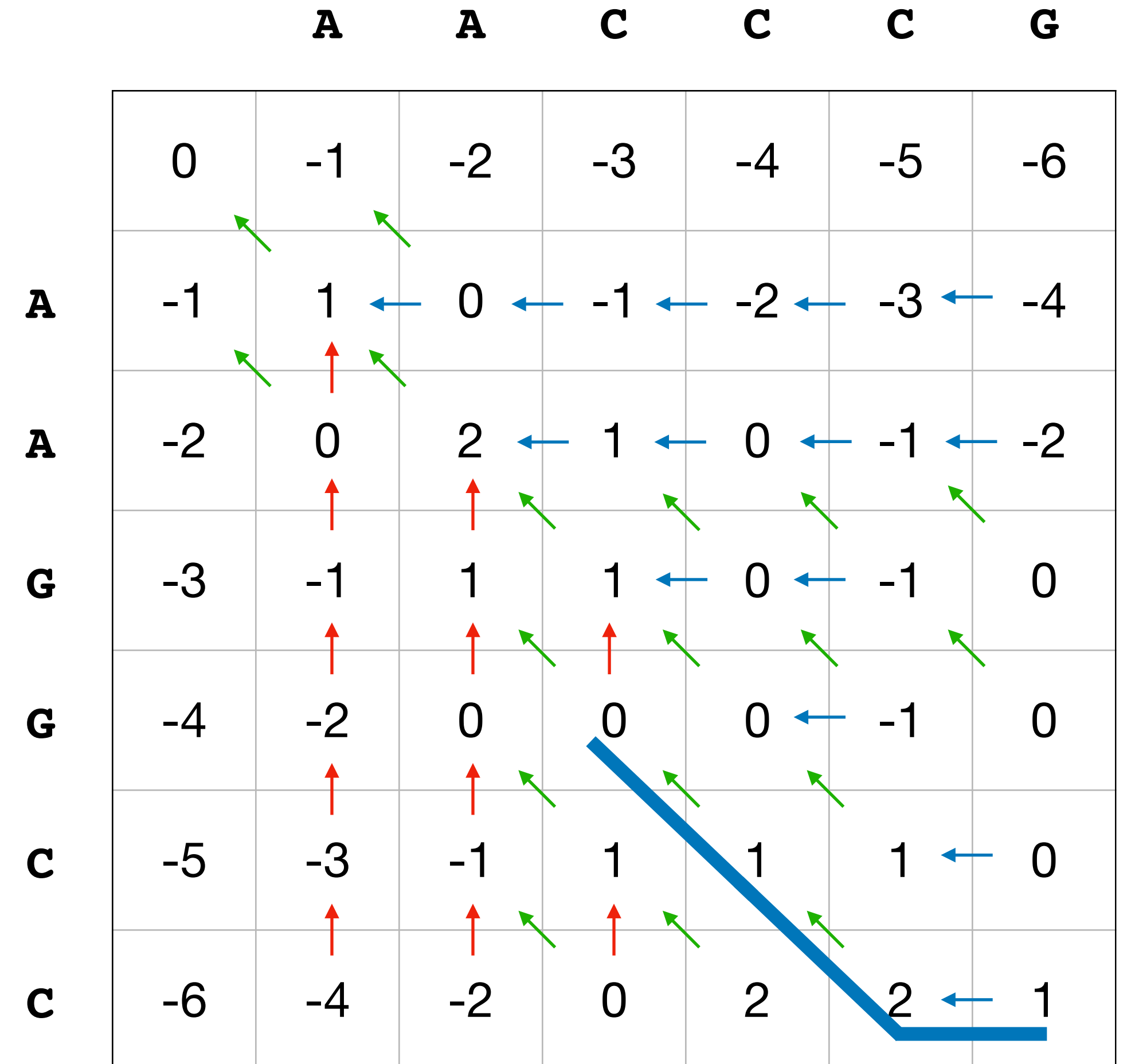
$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

C G  
C -





# Needleman-Wunch

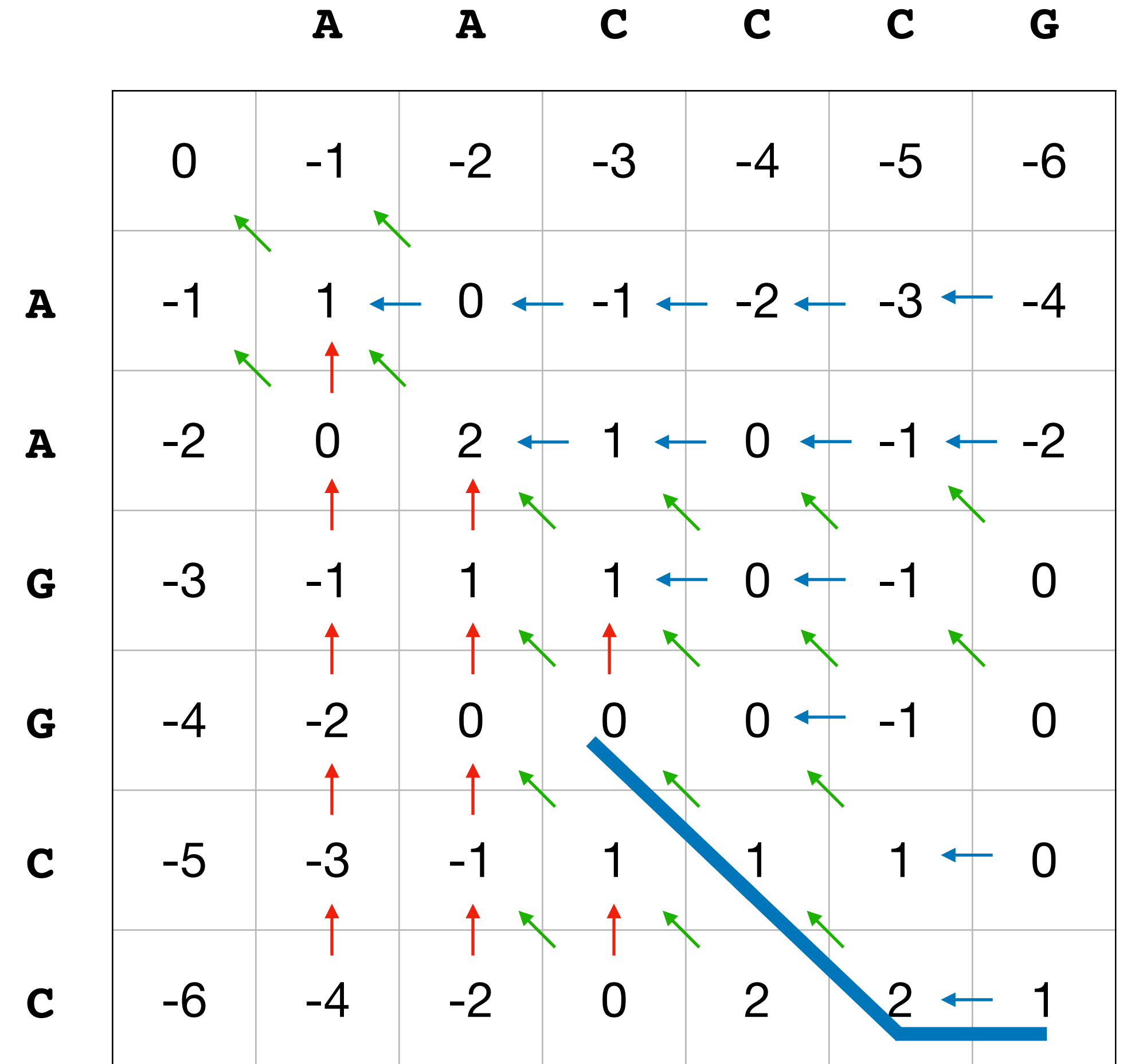
$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

C C G  
C C -



# Needleman-Wunch

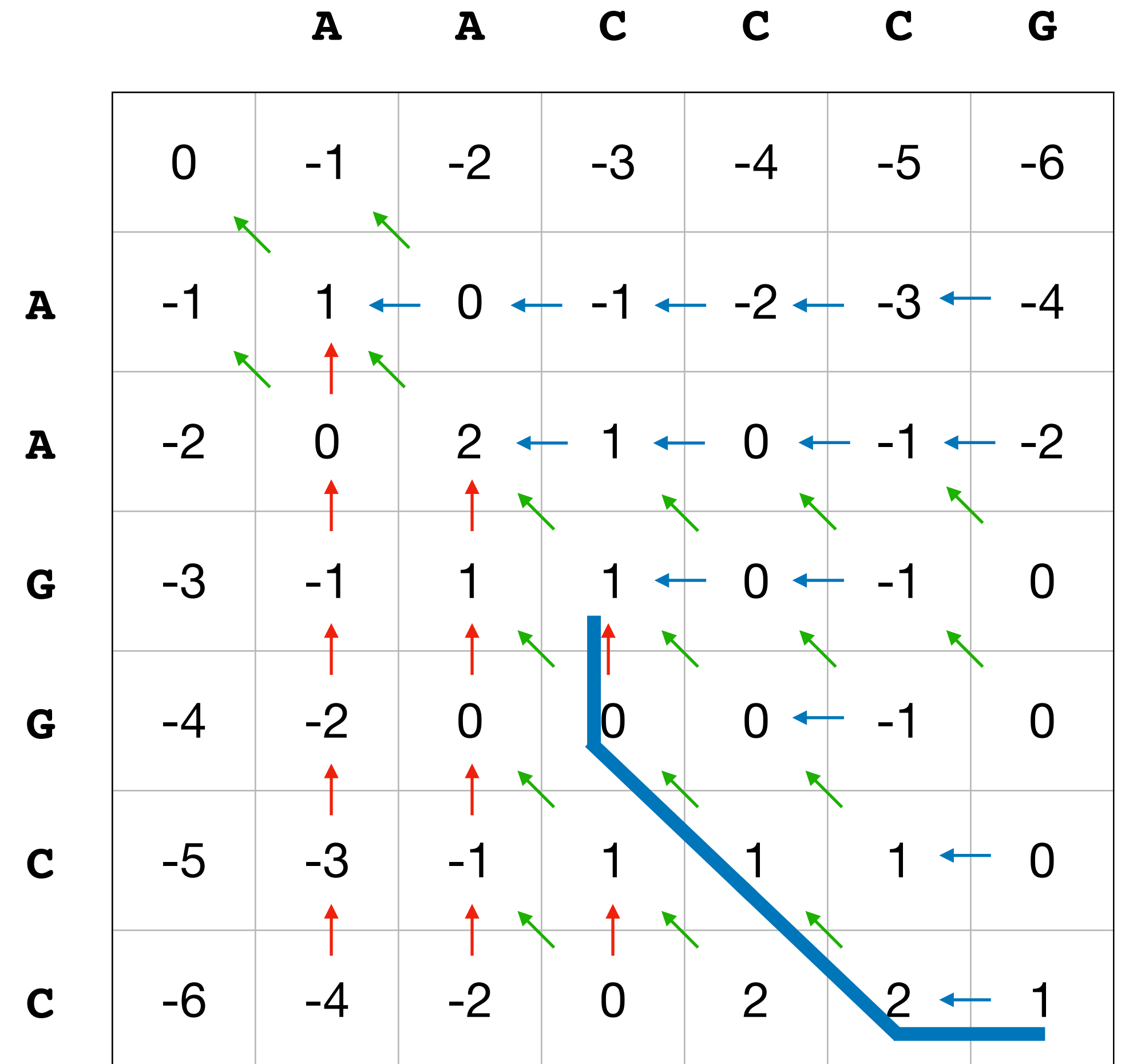
$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

C C G  
C C -



# Needleman-Wunch

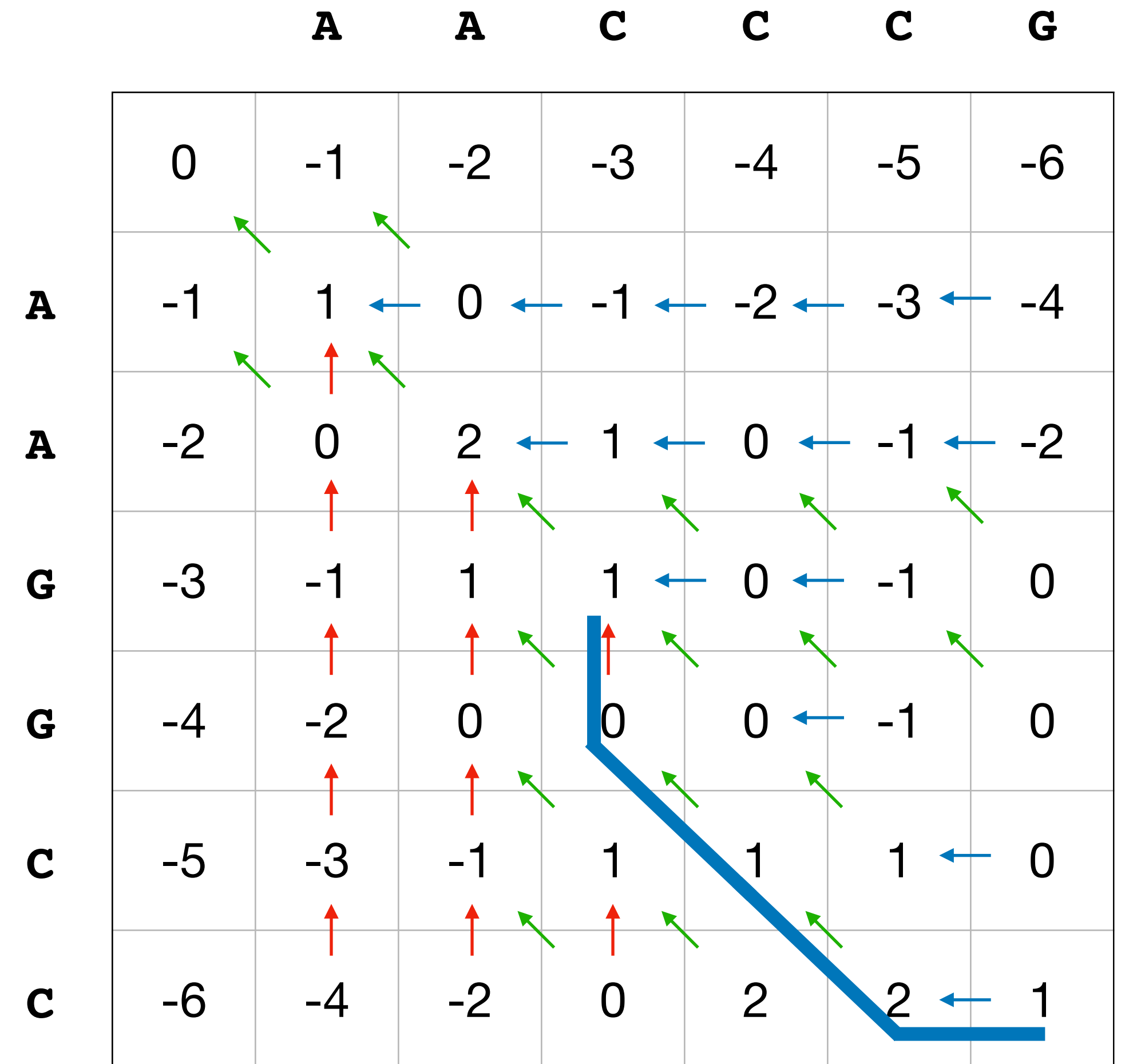
$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

- C C G  
G C C -



# Needleman-Wunch

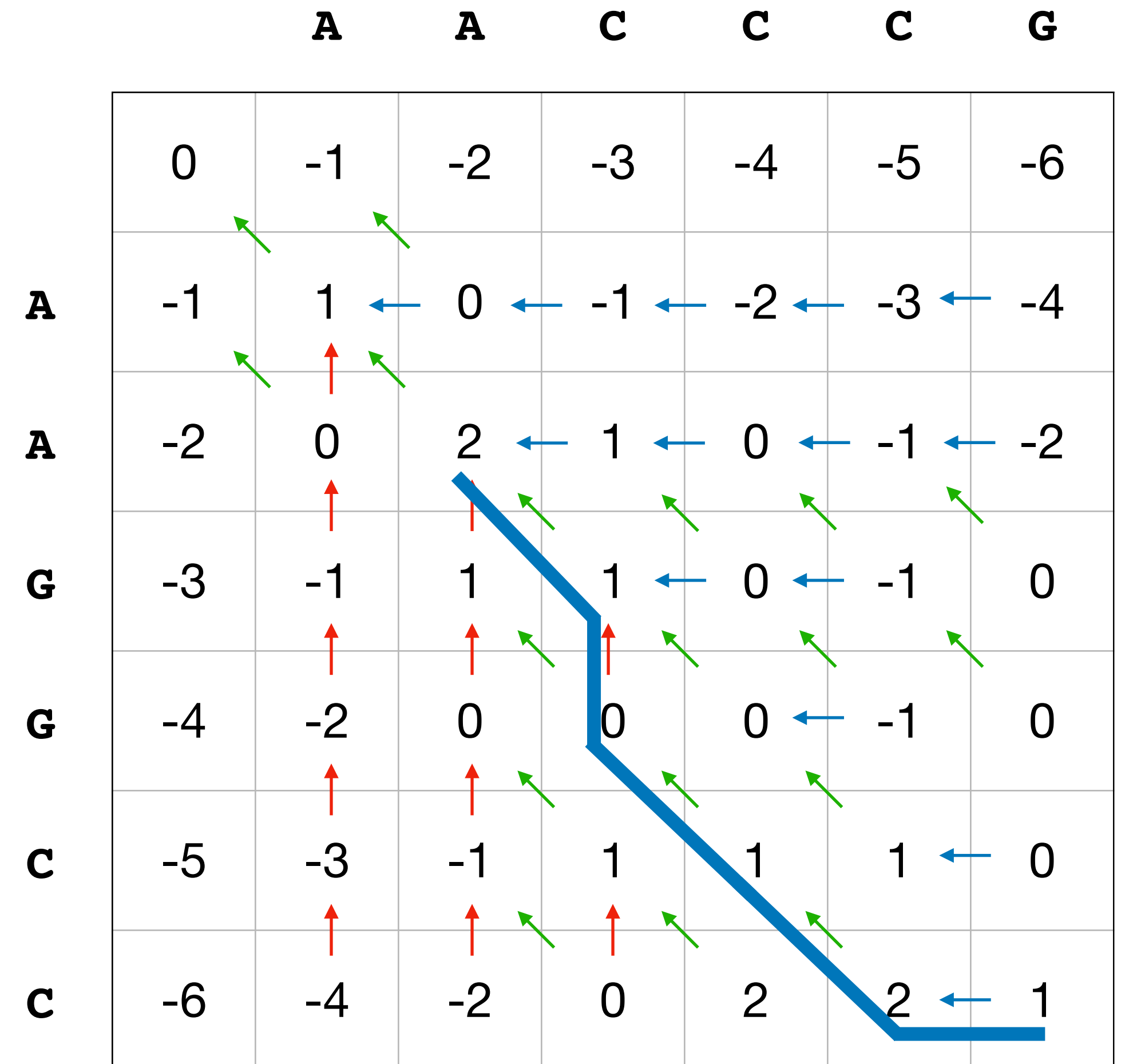
$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

- C C G  
G C C -



# Needleman-Wunch

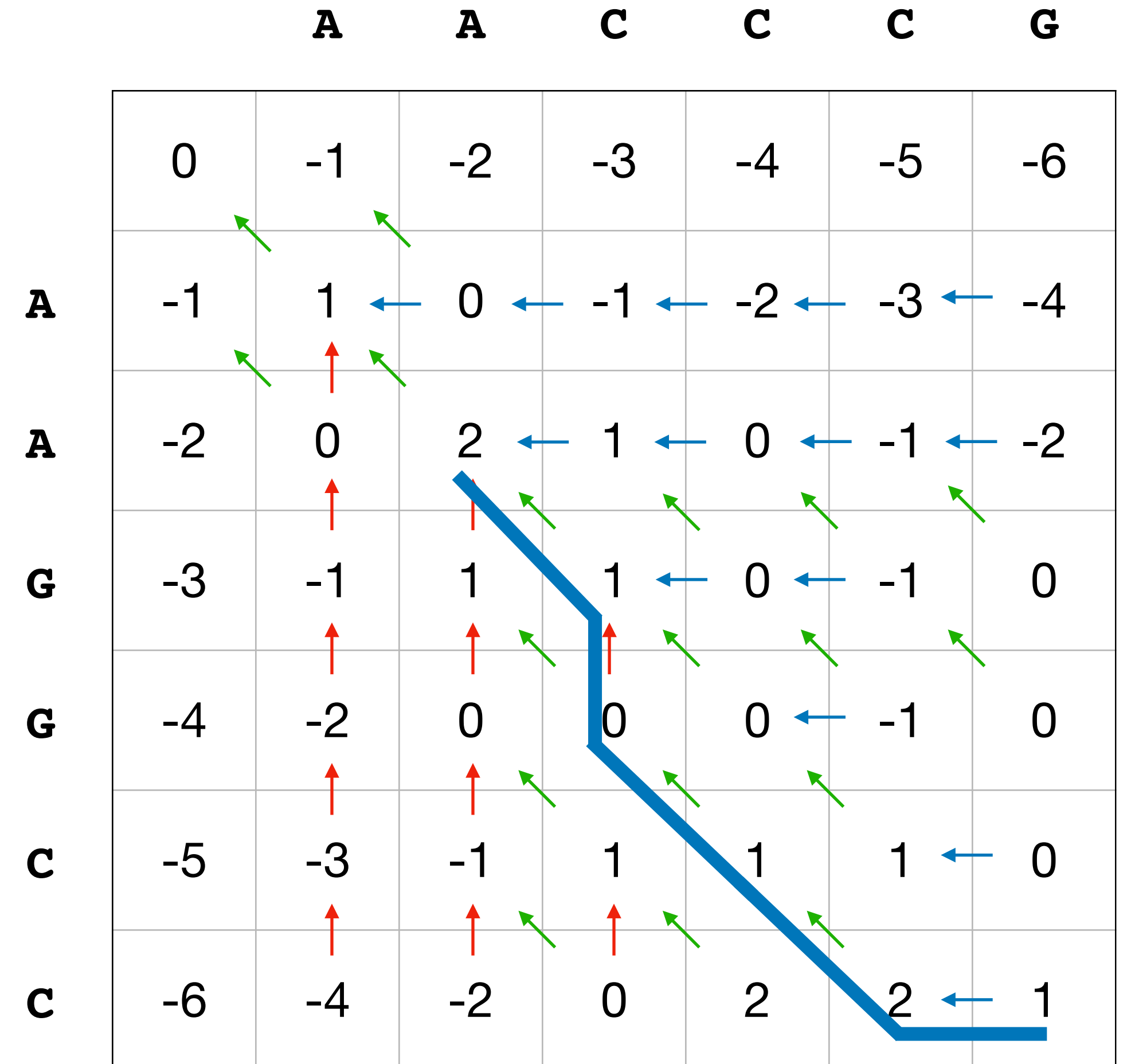
$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

C - C C G  
G G C C -



# Needleman-Wunch

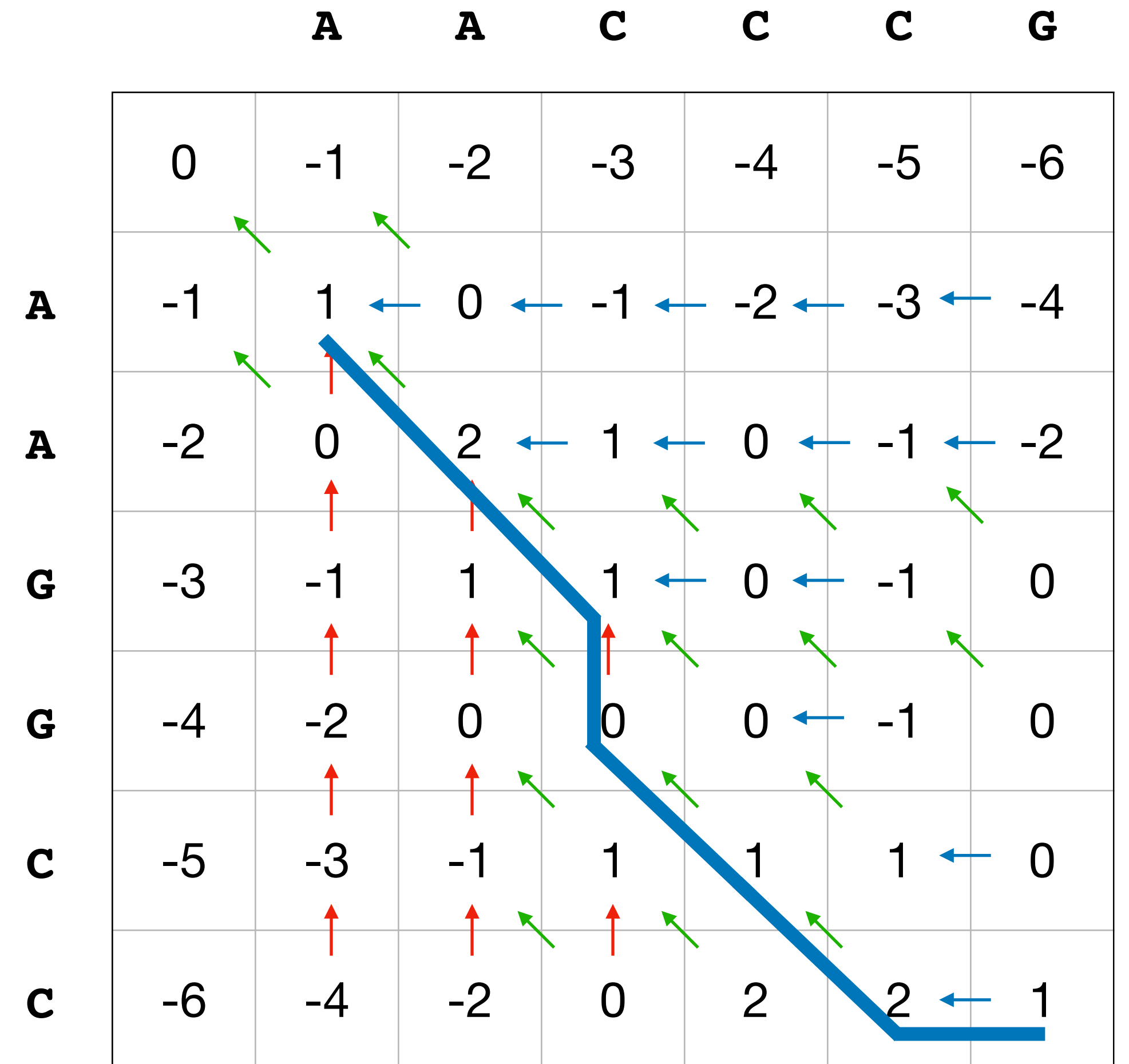
$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

C - C C G  
G G C C -



# Needleman-Wunch

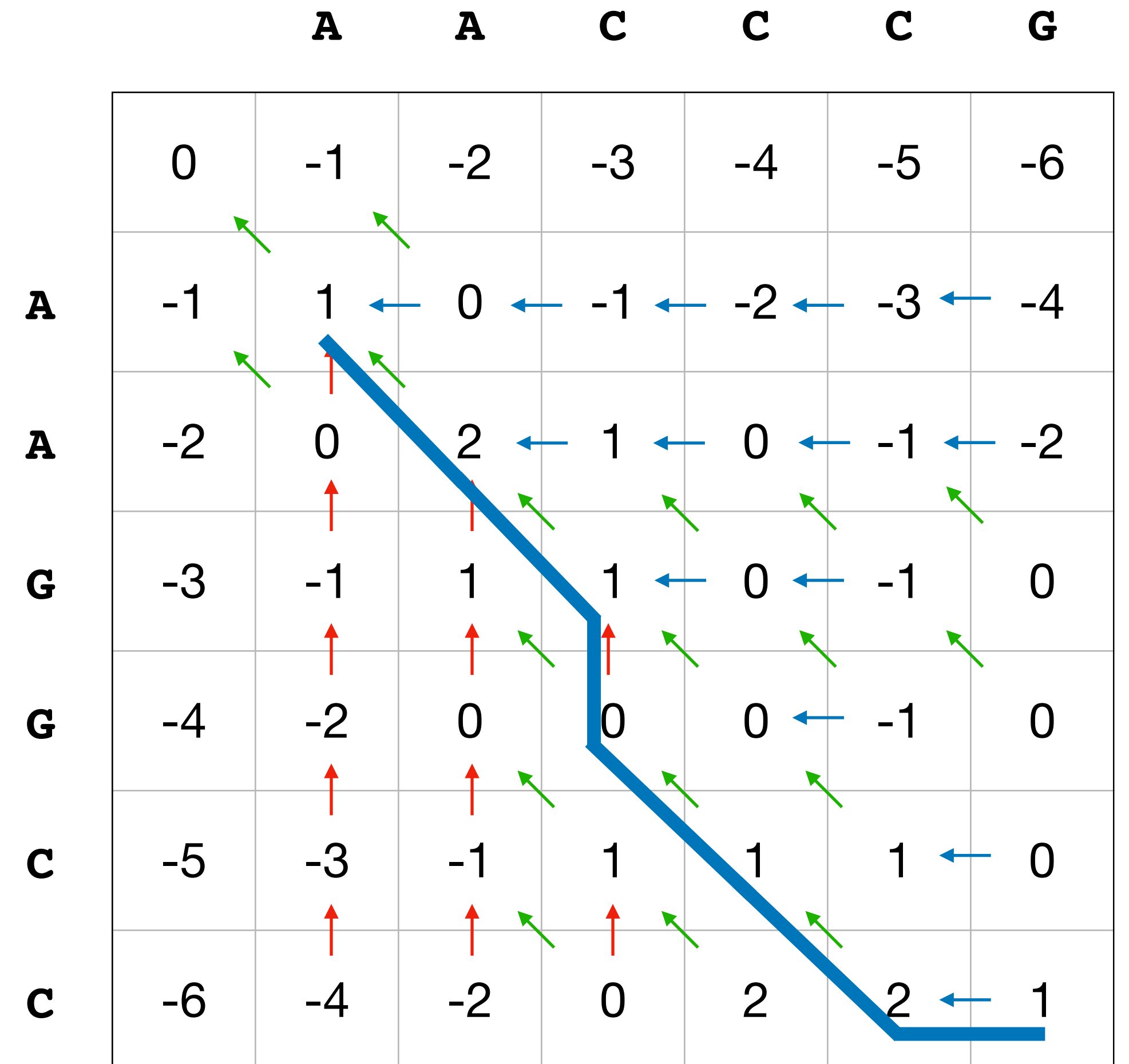
$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

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



# Needleman-Wunch

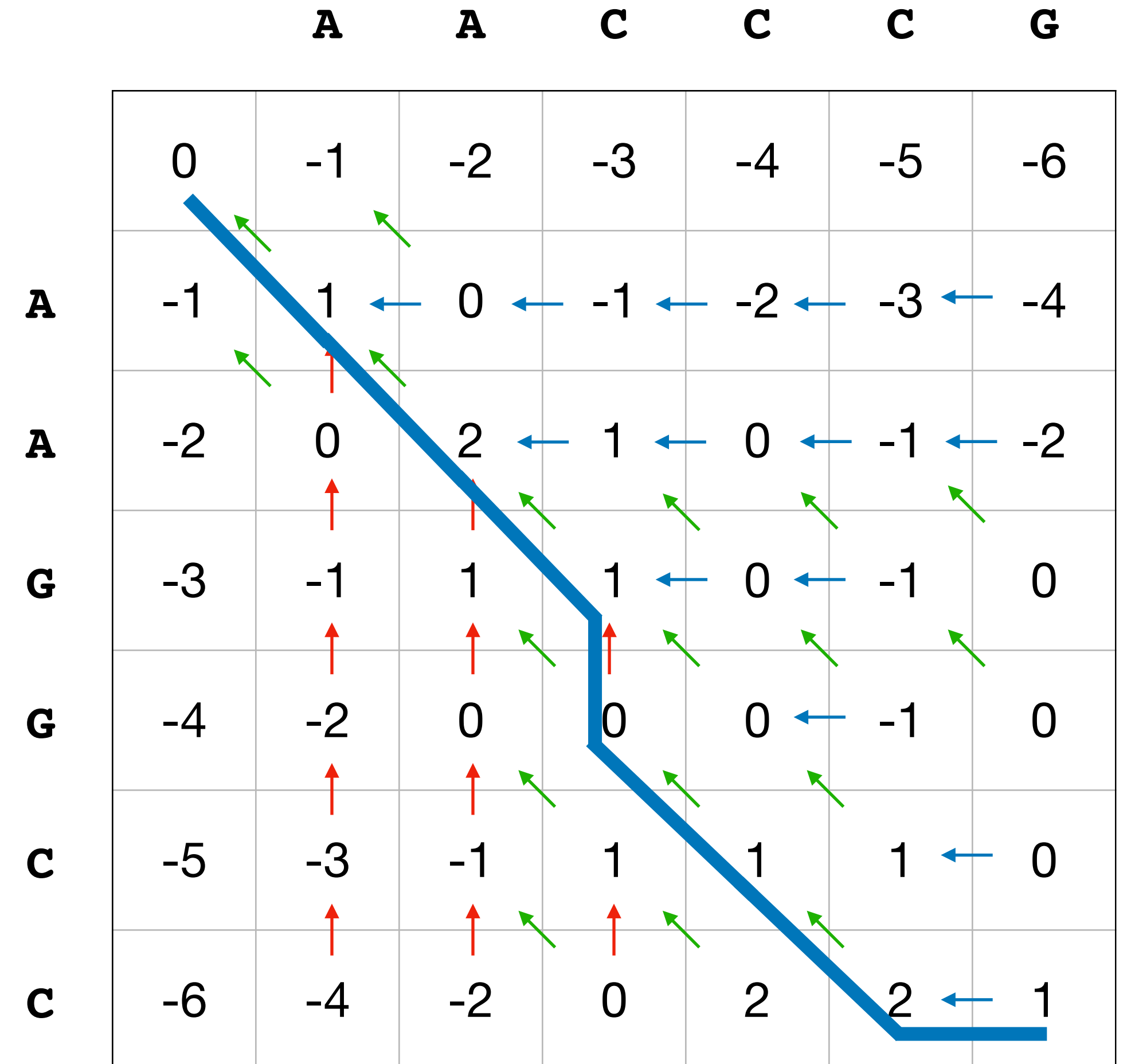
$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

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





# Needleman-Wunch

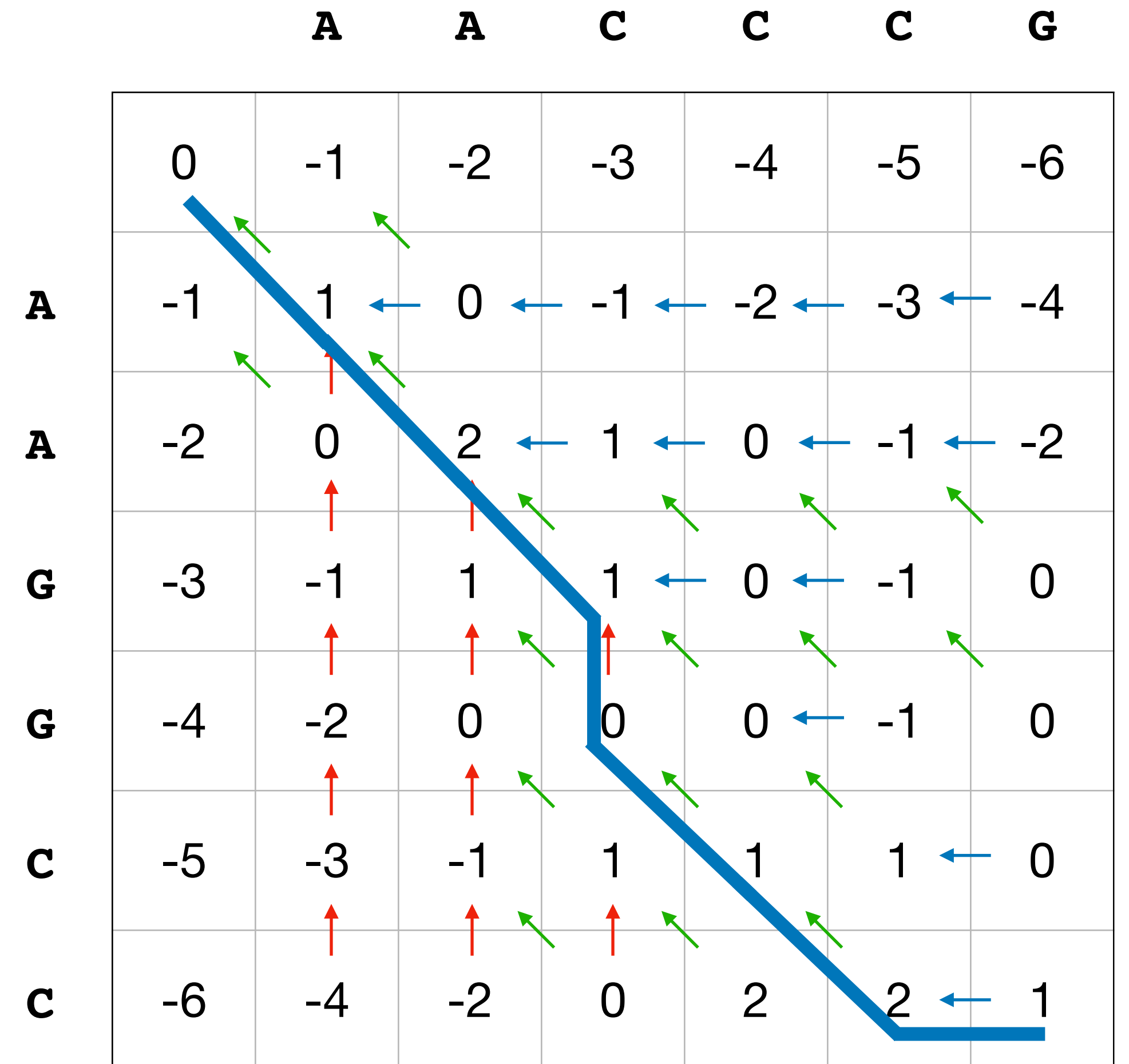
$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

**A A C - C C G**  
**A A G G C C -**



# Needleman-Wunch

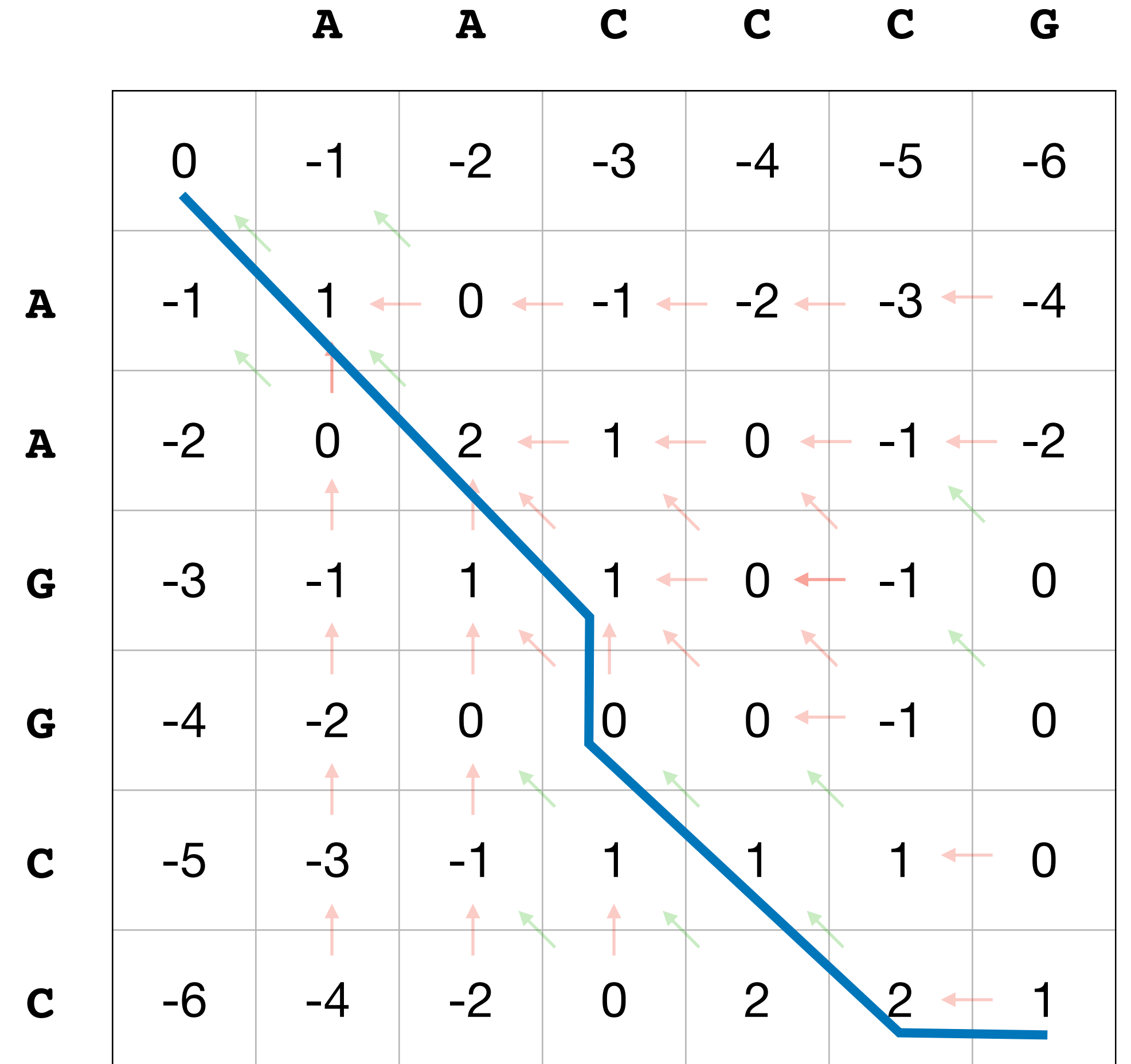
$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

**A A C - C C G**  
**A A G G C C -**



# Needleman-Wunch

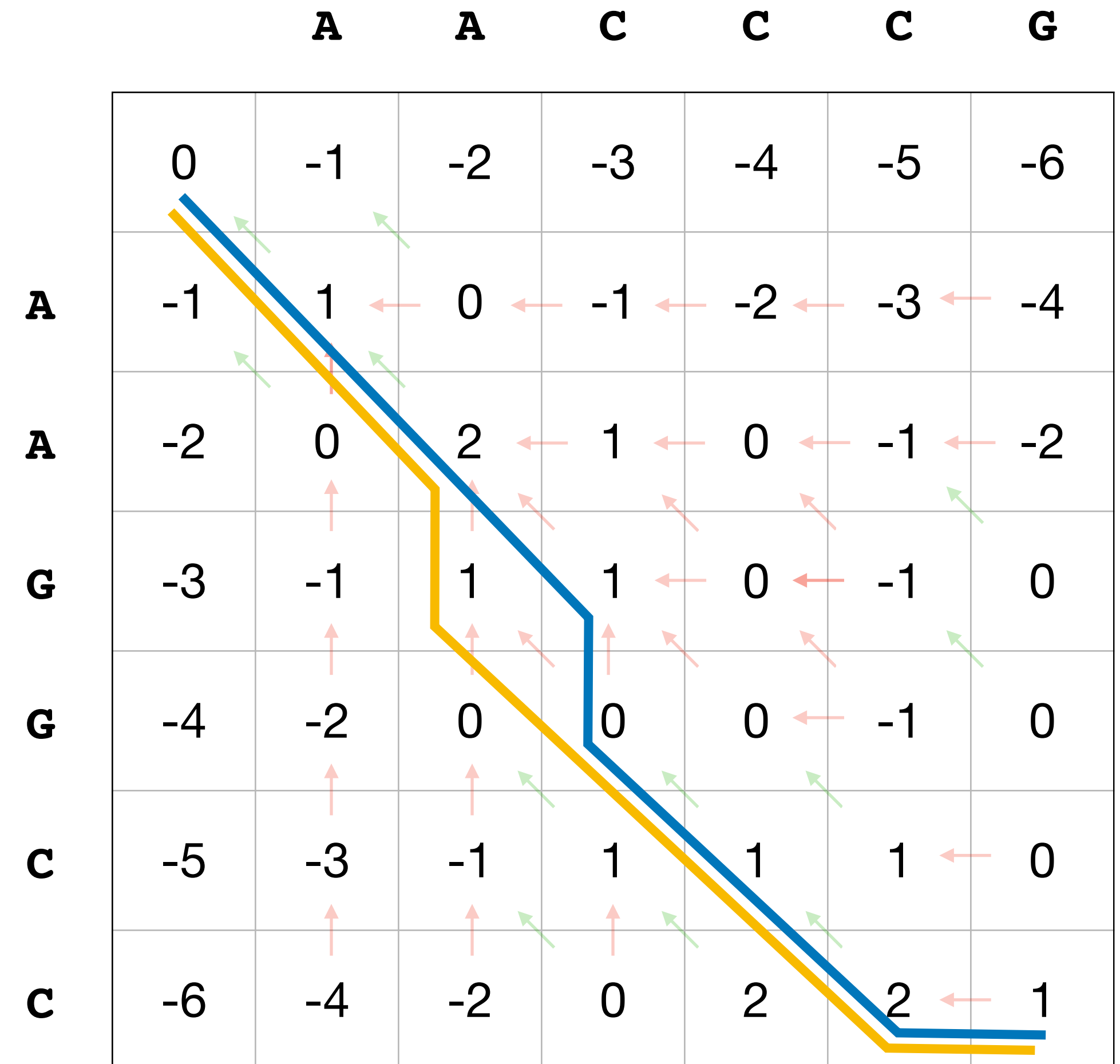
$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

**A A C - C C G**  
**A A G G C C -**



# Needleman-Wunch

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

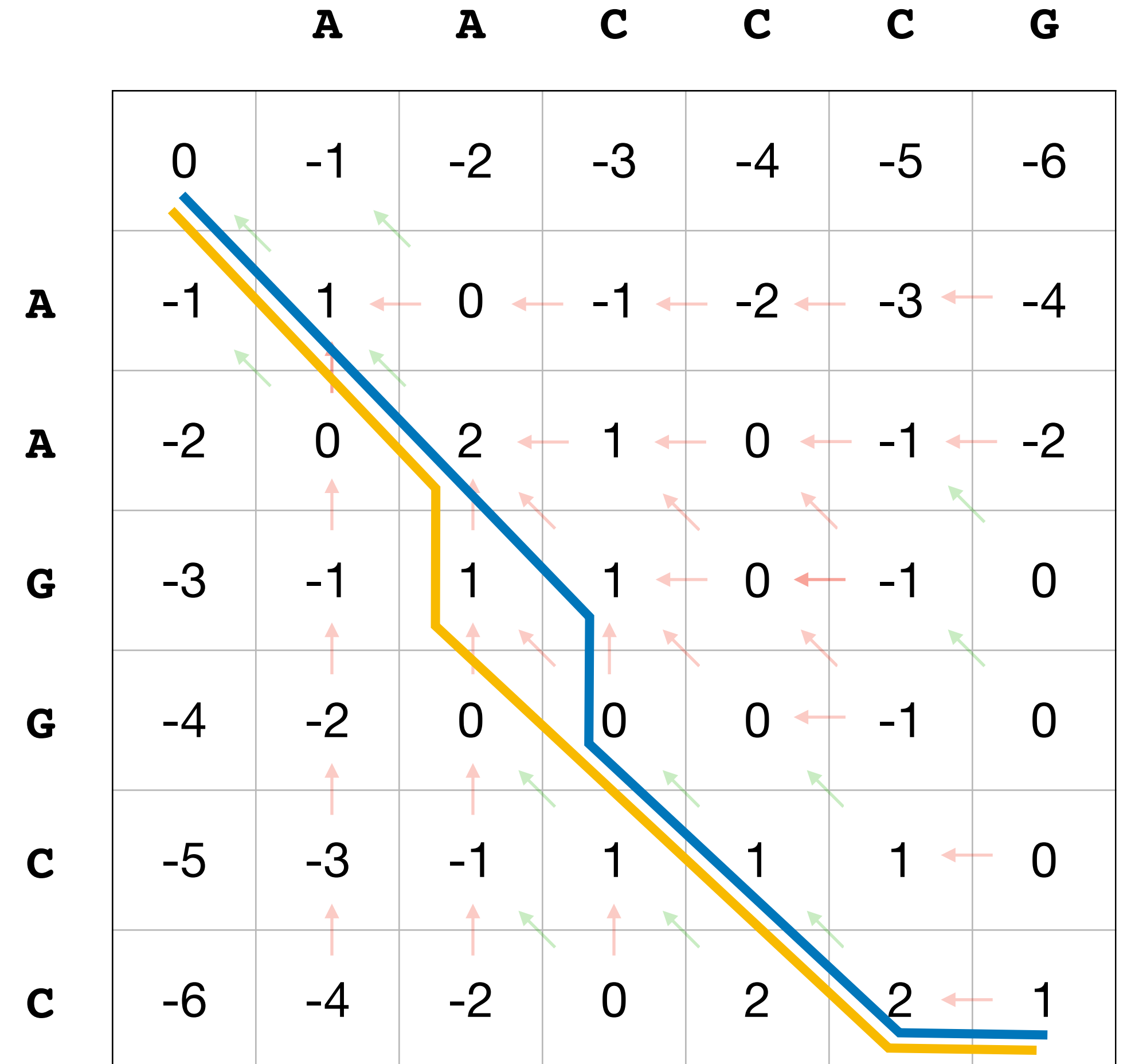
$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

**A A C - C C G**  
**A A G G C C -**

**A A - C C C G**  
**A A G G C C -**



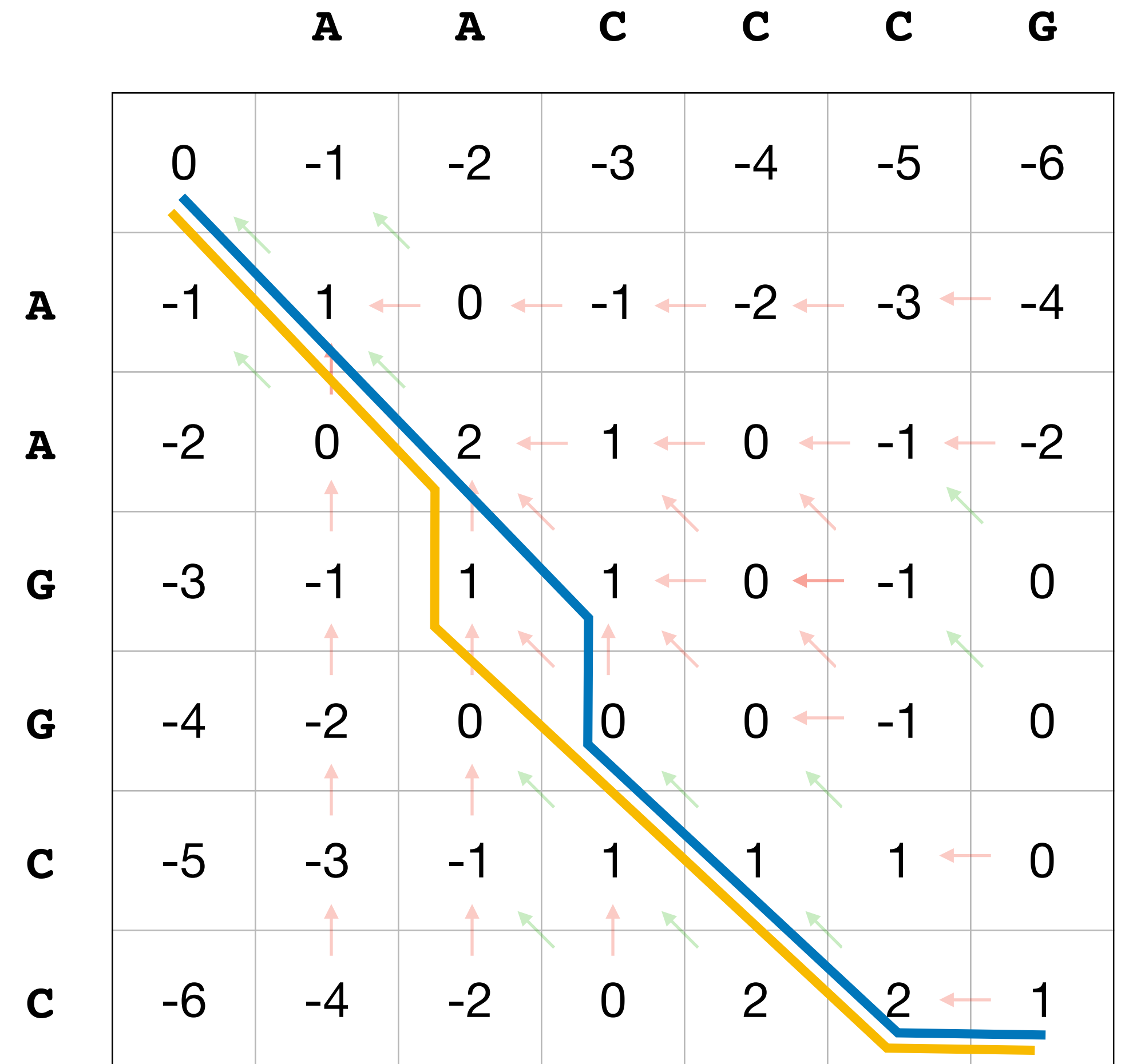
# Needleman-Wunch

What about the running time and memory requirements?

```

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

A A - C C C G
A A G G C C -
    
```



# Needleman-Wunch

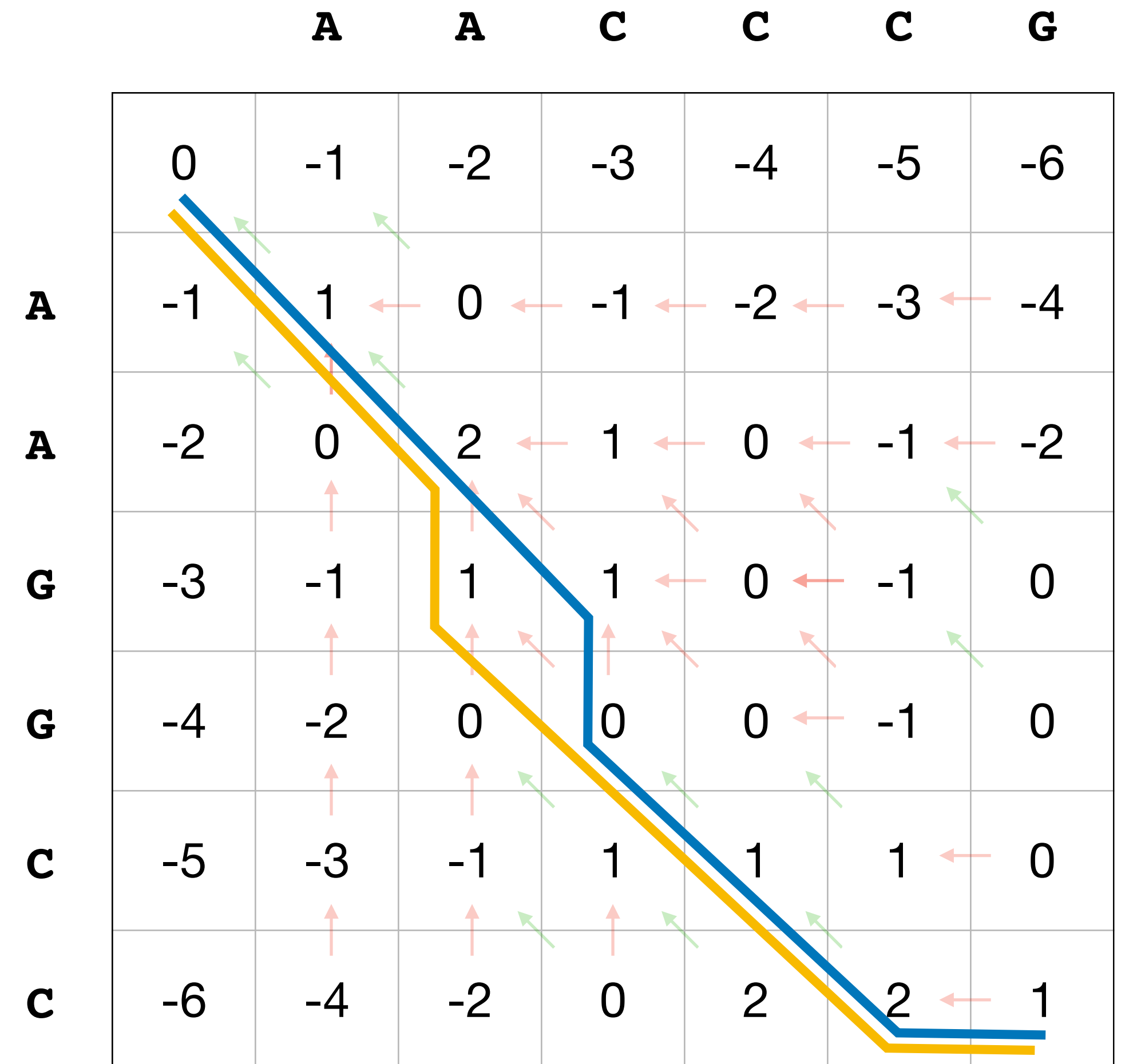
What about the running time and memory requirements?

- Filling in each cell of the table:

```

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

A A - C C C G
A A G G C C -
    
```



# Needleman-Wunch

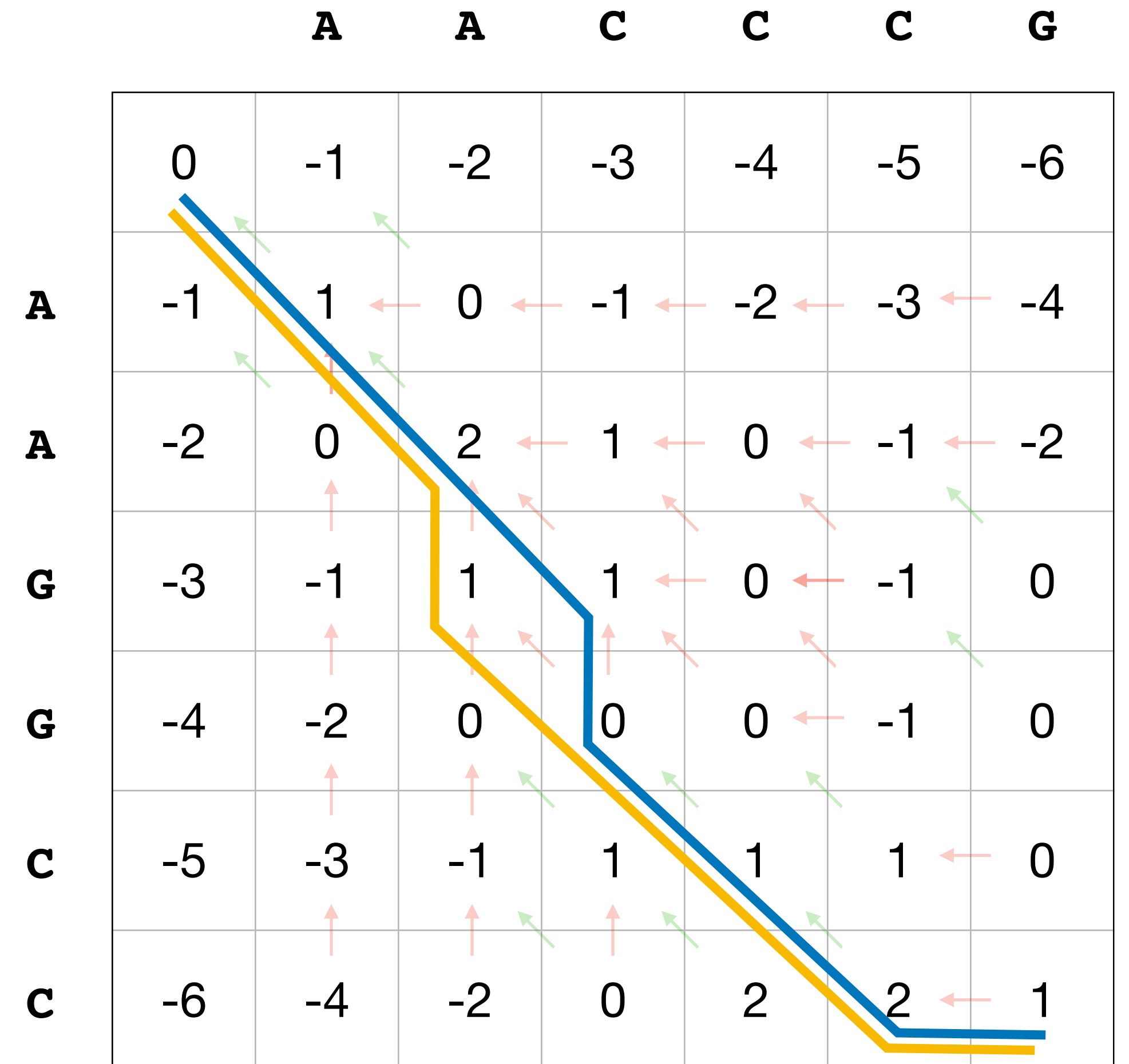
What about the running time and memory requirements?

- Filling in each cell of the table:  
 $O(1)$ -time,  $O(1)$ -space

```

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

A A - C C C G
A A G G C C -
    
```



# Needleman-Wunch

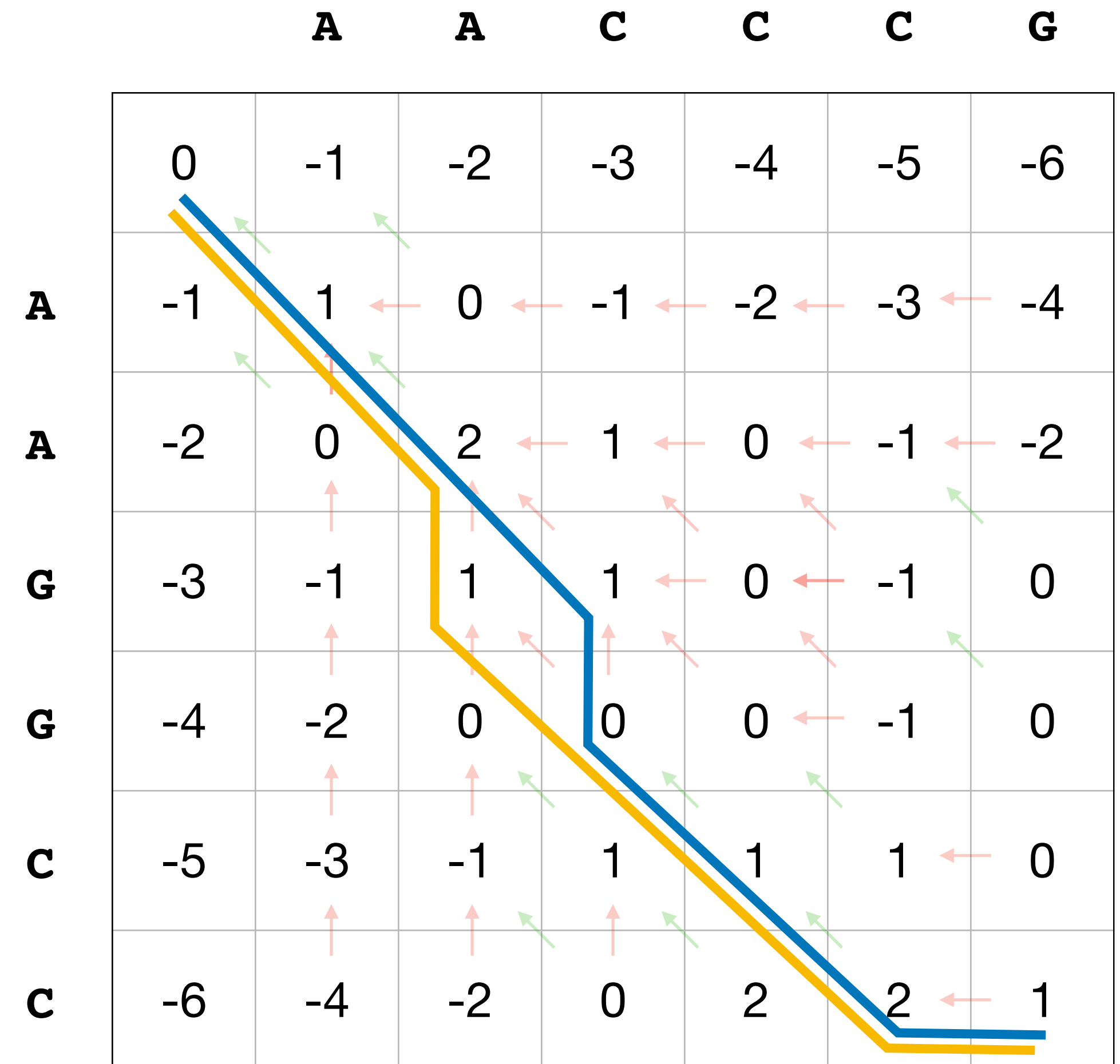
What about the running time and memory requirements?

- Filling in each cell of the table:  
 $O(1)$ -time,  $O(1)$ -space
- Table is  $n \times m$

```

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

A A - C C C G
A A G G C C -
    
```





# Needleman-Wunch

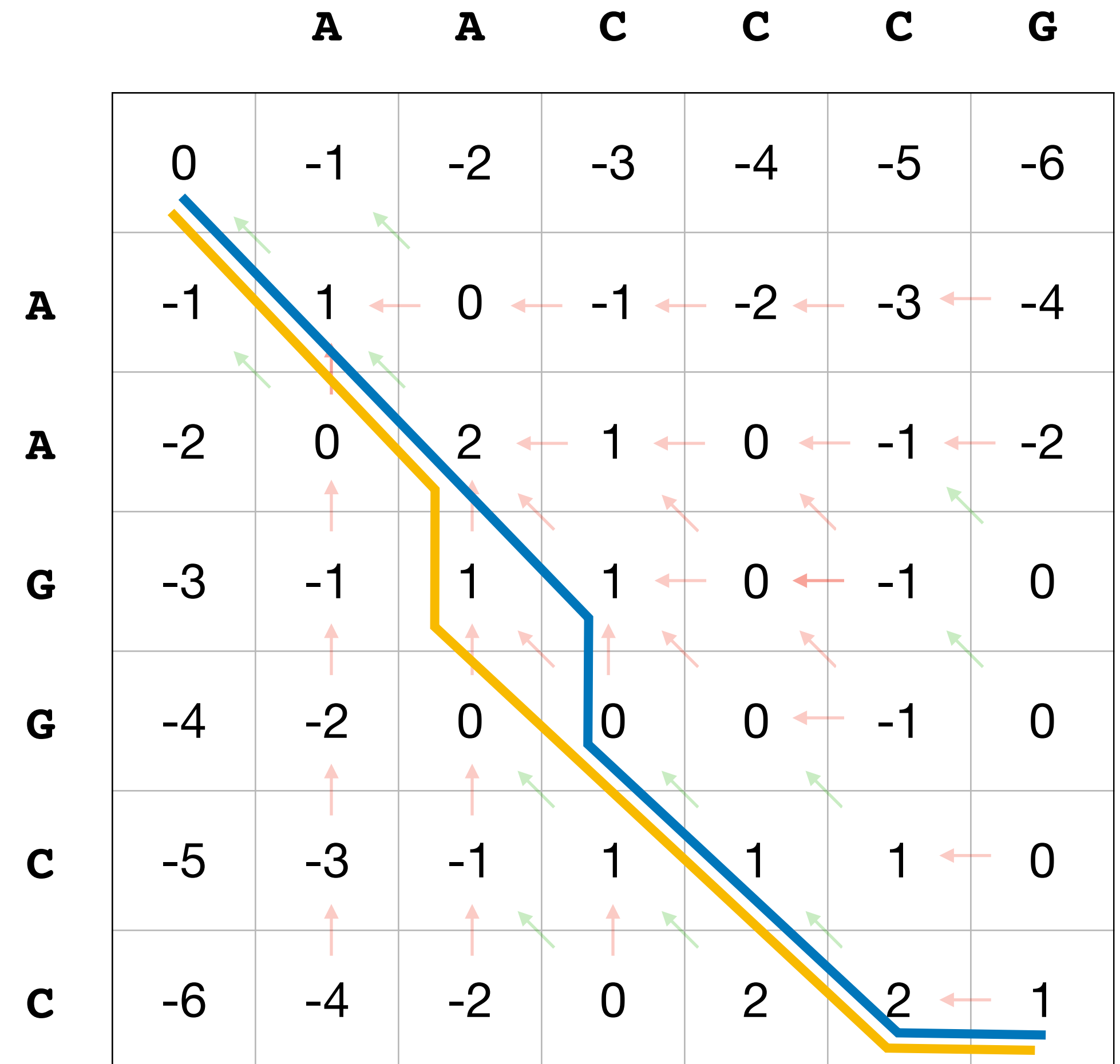
What about the running time and memory requirements?

- Filling in each cell of the table:  
 $O(1)$ -time,  $O(1)$ -space
- Table is  $n \times m$
- Filling in the table:

```

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

A A - C C C G
A A G G C C -
    
```



# Needleman-Wunch

What about the running time and memory requirements?

- Filling in each cell of the table:  
 $O(1)$ -time,  $O(1)$ -space

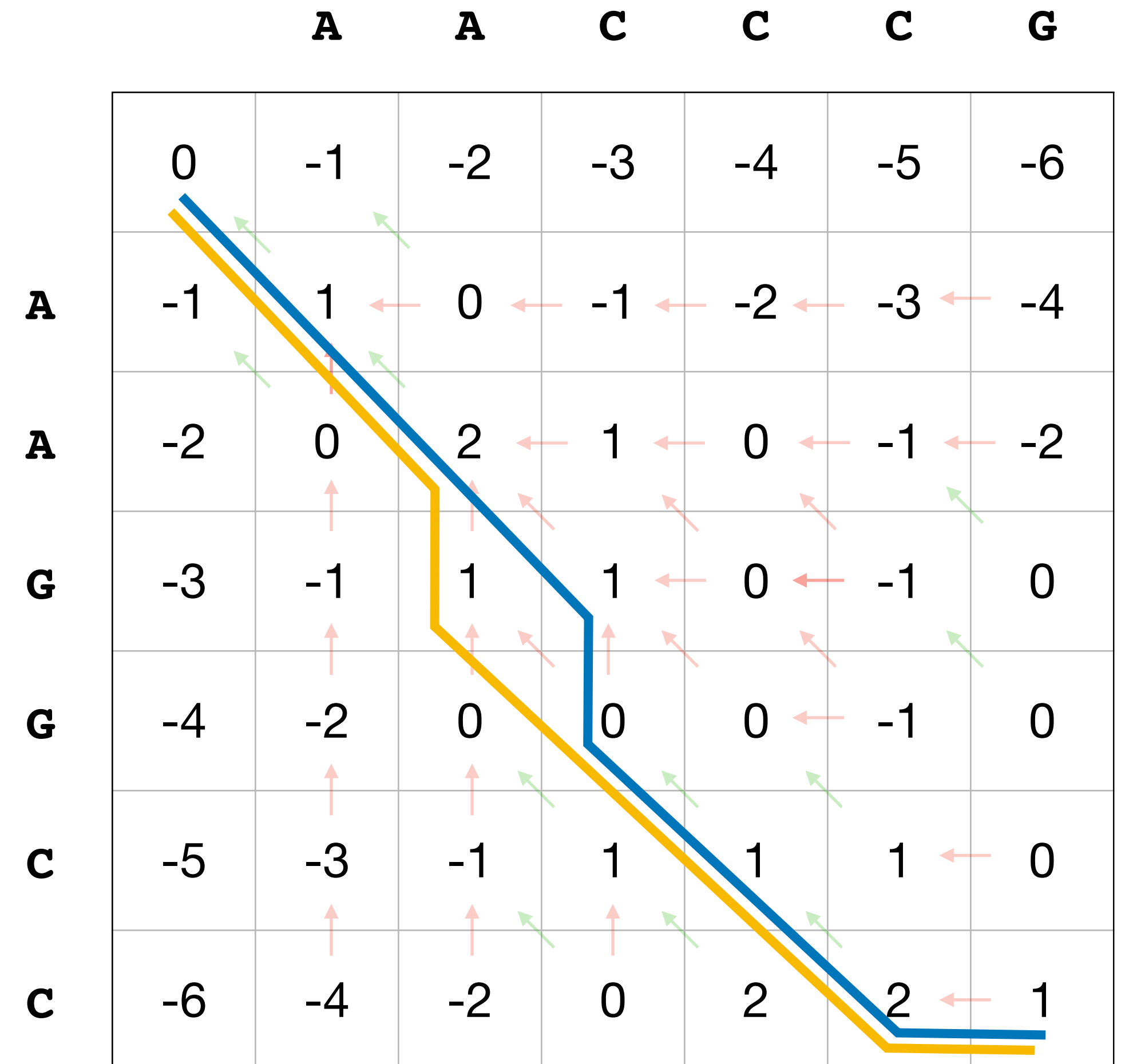
- Table is  $n \times m$

- Filling in the table:  
 $O(mn)$ -time,  $O(mn)$ -space

```

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

A A - C C C G
A A G G C C -
    
```



# Needleman-Wunch

What about the running time and memory requirements?

- Filling in each cell of the table:  
 $O(1)$ -time,  $O(1)$ -space

- Table is  $n \times m$

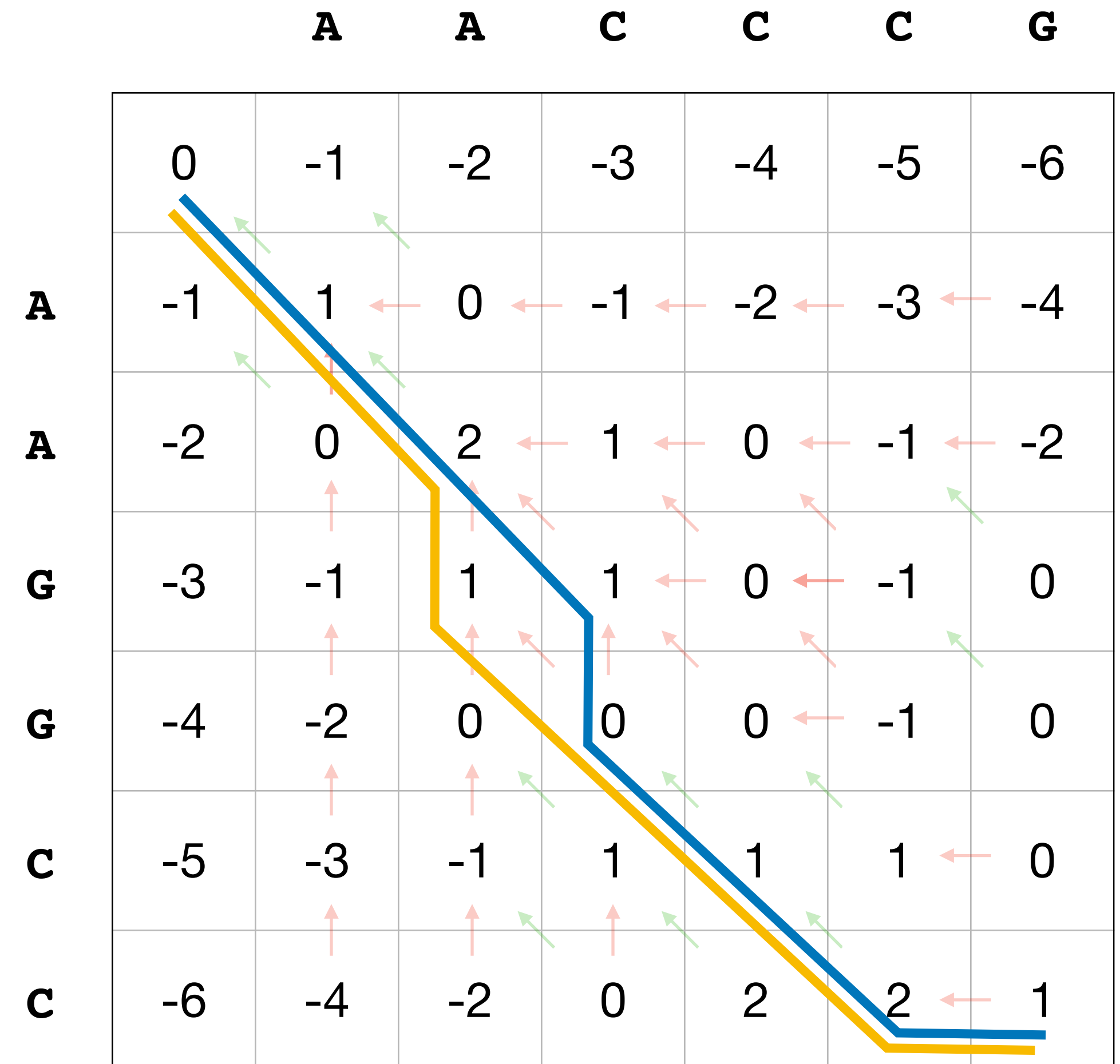
- Filling in the table:  
 $O(mn)$ -time,  $O(mn)$ -space

- Traceback?

```

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

A A - C C C G
A A G G C C -
    
```



# Needleman-Wunch

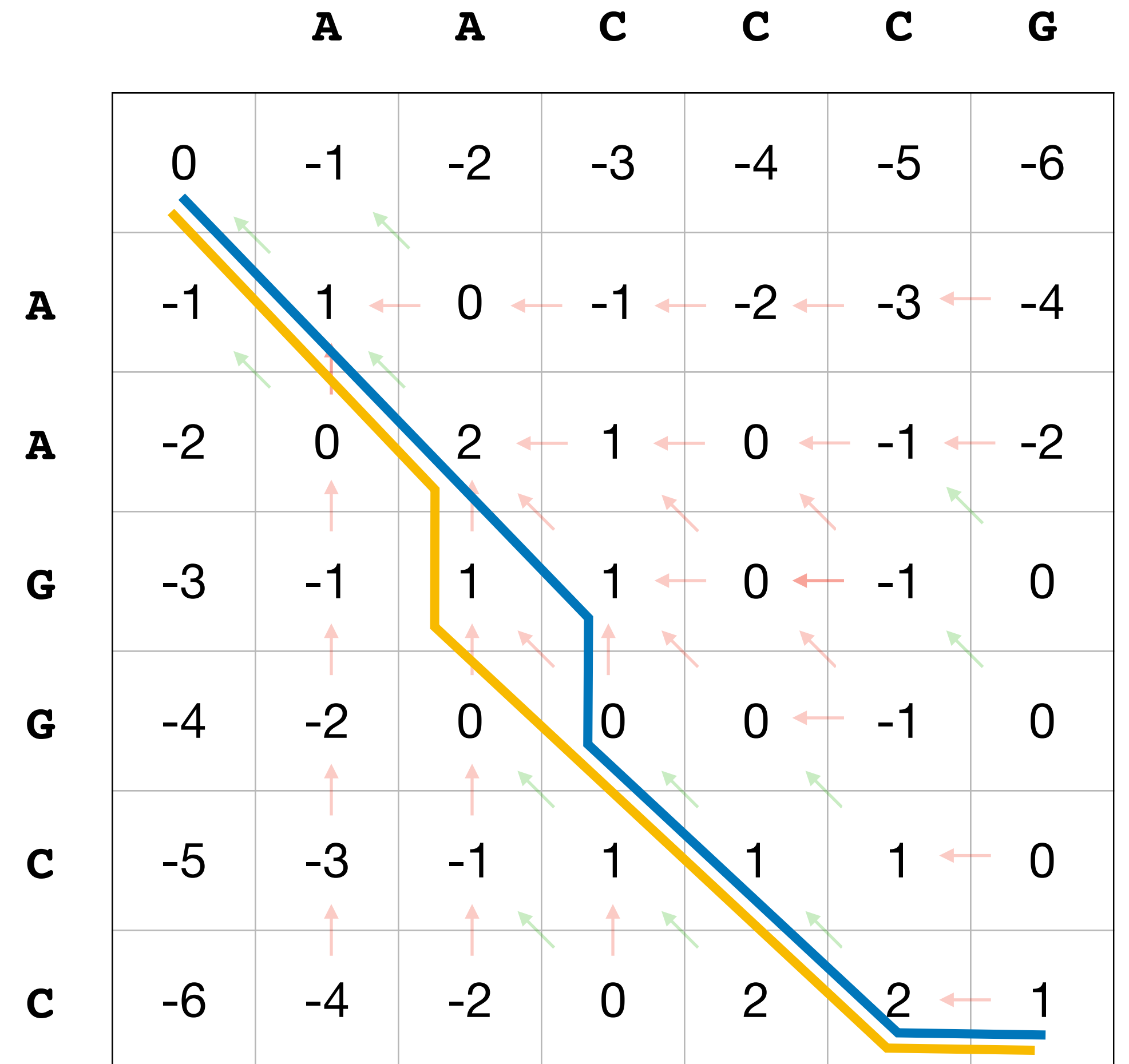
What about the running time and memory requirements?

- Filling in each cell of the table:  
 $O(1)$ -time,  $O(1)$ -space
- Table is  $n \times m$
- Filling in the table:  
 $O(mn)$ -time,  $O(mn)$ -space
- Traceback?
  - Each column of the alignment:

```

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

A A - C C C G
A A G G C C -
    
```



# Needleman-Wunch

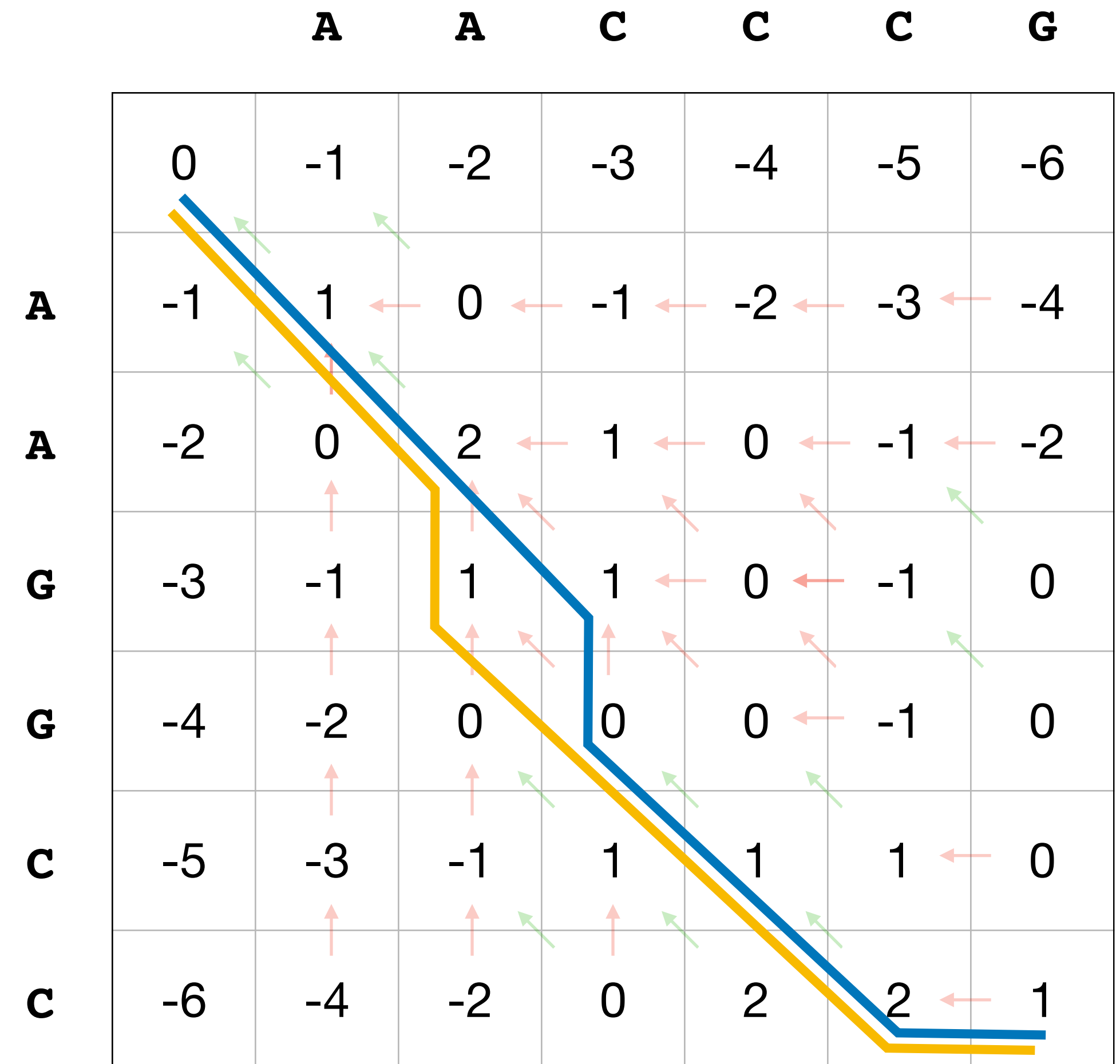
What about the running time and memory requirements?

- Filling in each cell of the table:  
 $O(1)$ -time,  $O(1)$ -space
- Table is  $n \times m$
- Filling in the table:  
 $O(mn)$ -time,  $O(mn)$ -space
- Traceback?  
• Each column of the alignment:  
 $O(1)$ -time

```

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

A A - C C C G
A A G G C C -
    
```



# Needleman-Wunch

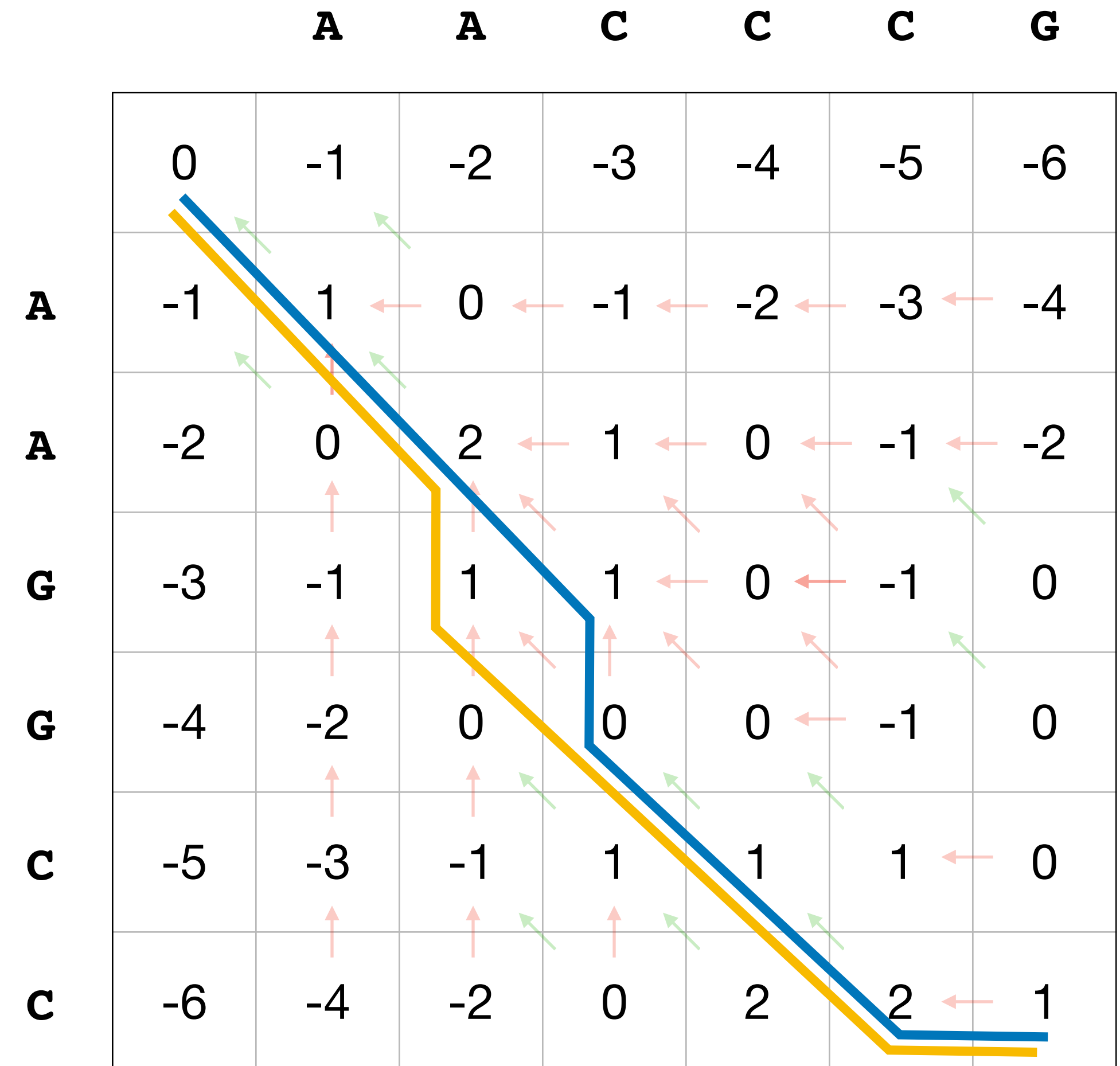
What about the running time and memory requirements?

- Filling in each cell of the table:  
 $O(1)$ -time,  $O(1)$ -space
- Table is  $n \times m$
- Filling in the table:  
 $O(mn)$ -time,  $O(mn)$ -space
- Traceback?
  - Each column of the alignment:  
 $O(1)$ -time
  - Maximum Alignment Length:

```

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

A A - C C C G
A A G G C C -
    
```



# Needleman-Wunch

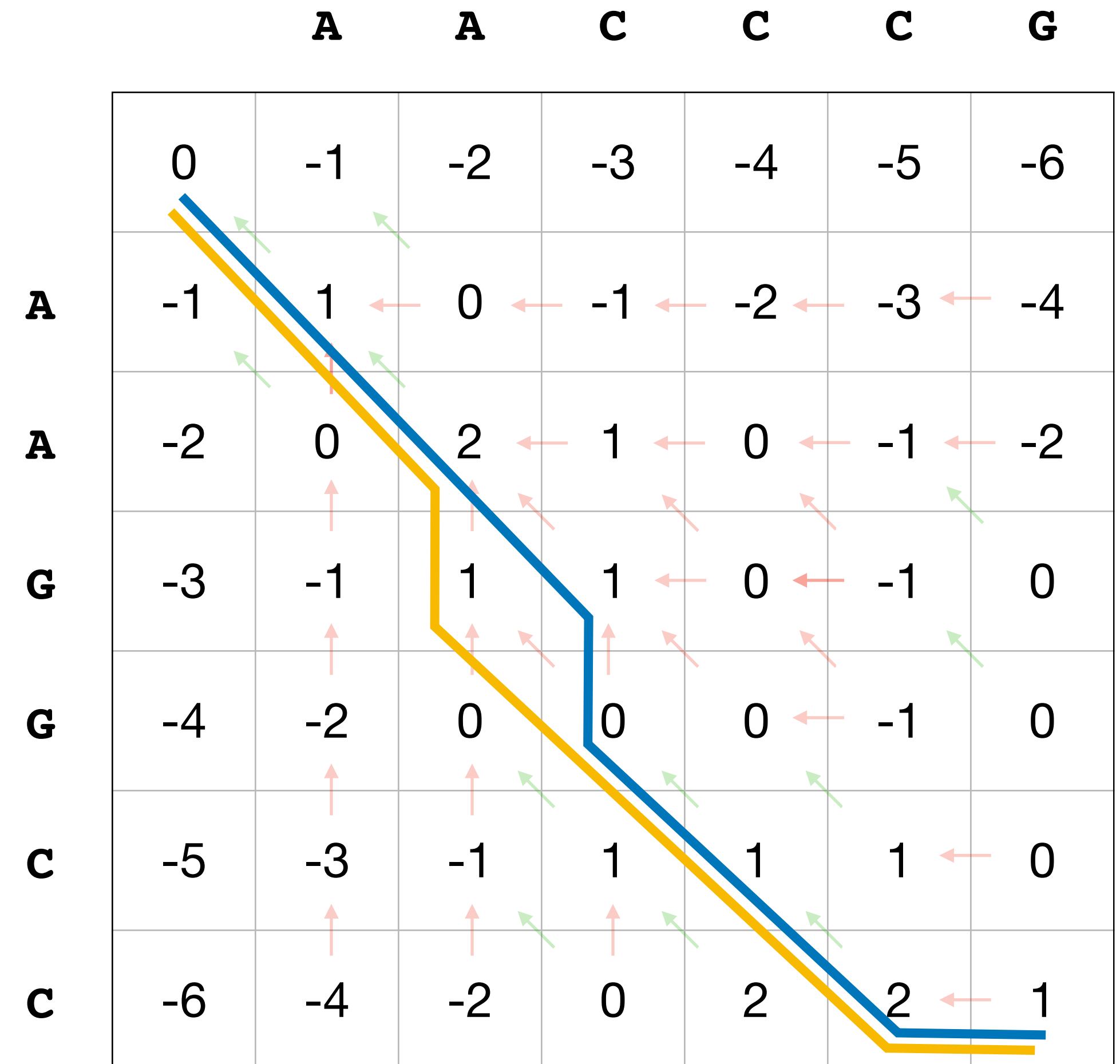
What about the running time and memory requirements?

- Filling in each cell of the table:  
 $O(1)$ -time,  $O(1)$ -space
- Table is  $n \times m$
- Filling in the table:  
 $O(mn)$ -time,  $O(mn)$ -space
- Traceback?
  - Each column of the alignment:  
 $O(1)$ -time
  - Maximum Alignment Length:  
 $O(m+n)$

```

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

A A - C C C G
A A G G C C -
    
```



# Needleman-Wunch

What about the running time and memory requirements?

- Filling in each cell of the table:  
 $O(1)$ -time,  $O(1)$ -space

- Table is  $n \times m$

- Filling in the table:  
 $O(mn)$ -time,  $O(mn)$ -space

- Traceback?

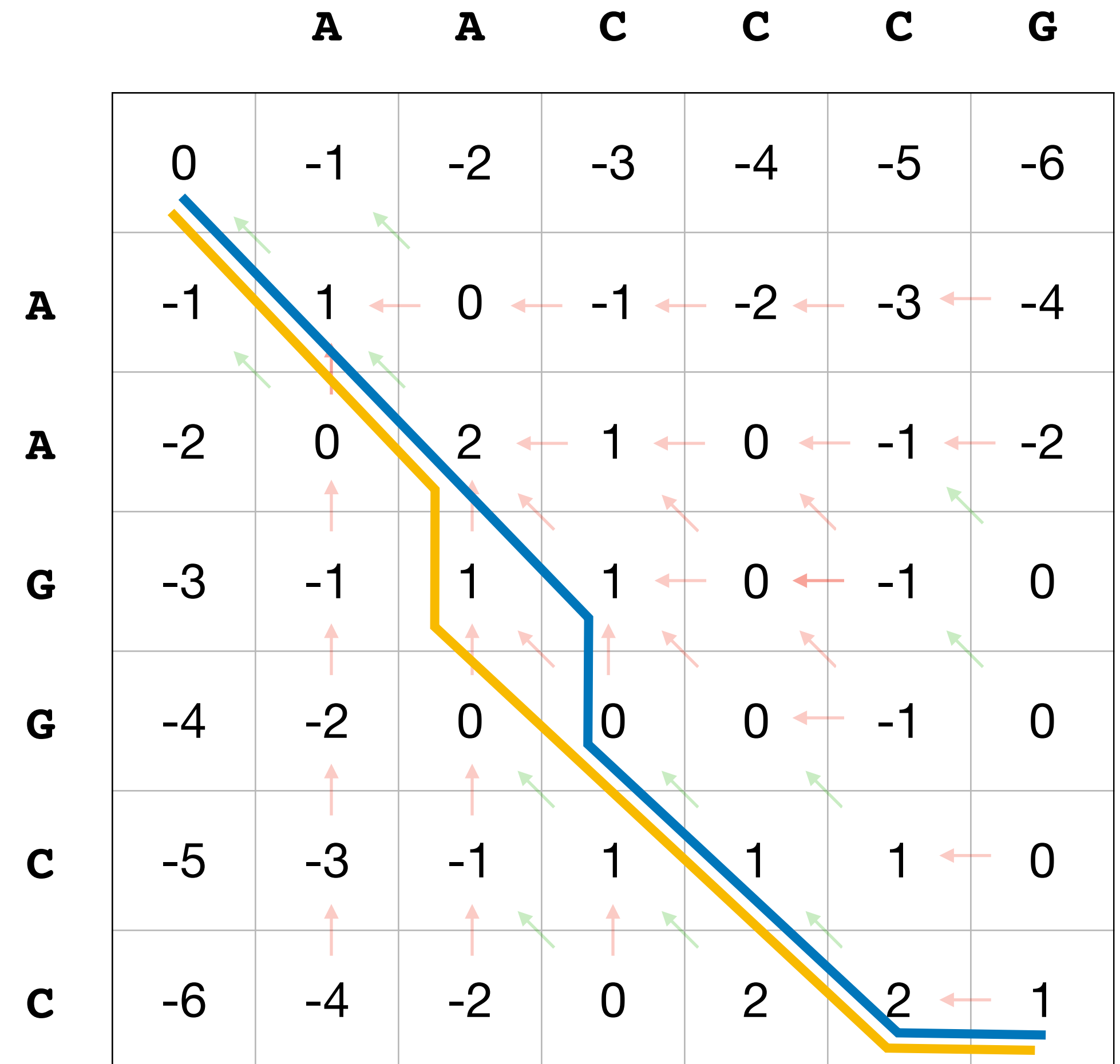
- Each column of the alignment:  
 $O(1)$ -time

- Maximum Alignment Length:  
 $O(m+n)$   
(times the number of optimal alignments)

```

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

A A - C C C G
A A G G C C -
    
```





- The same problem from the end of Tuesday's class:  
Is pattern  $P$  present with at most 1 error in text  $T$ ?

# *Local* Alignment

- Given two strings  $S$  and  $T$ , find the two substrings,  $A$  of  $S$  and  $B$  of  $T$ , with the highest alignment score.
- Brute-force: Align all substrings of  $S$  with all substrings of  $T$ . There are  $\binom{n}{2}$  substrings of  $S$ , and  $\binom{m}{2}$  substrings of  $T$ . The total running time would be  $O(n^3m^3)$ !
- Smith and Waterman [1981] developed an algorithm, similar to Needleman-Wunch, that is able to find the optimal local alignment in  $O(mn)$ -time.

# Smith-Waterman

- Still going to use an  $n \times m$  sized matrix  $V$ , but:
  - each index  $(i, j)$  will hold the maximum global alignment score for all substrings  $S[k \dots i]$  and  $T[h \dots j]$  where  $1 \leq k \leq i$  and  $1 \leq h \leq j$  (the substrings could be empty).
- Then the score of the best local alignment is not necessarily in  $V(i, j)$ , but is  $\max_{i, j} V(i, j)$

# Smith-Waterman

- The recurrence relation

$$V(i, j) = \max \begin{cases} 0 & \text{align empty strings} \\ V(i-1, j-1) + \delta(S[i], T[i]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$

- The initialization is:

$$V(0, j) = V(i, 0) = 0$$

# Smith-Waterman

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(i, j) = \max \begin{cases} 0 & \text{align empty strings} \\ V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$

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

Annotations on the table:
 

- A yellow dot is at the intersection of row 'A' and column 'A'.
- A green arrow points from the dot to the cell above and to the left (row '-', column '-').
- A red arrow points from the dot to the cell above (row '-', column 'A').
- A blue arrow points from the dot to the cell to the left (row 'A', column '-').
- Labels:  $+\delta(A, A)$  (green),  $+\delta(-, A)$  (red),  $+\delta(A, -)$  (blue), and  $0$  (yellow).

# Smith-Waterman

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(i, j) = \max \begin{cases} 0 & \text{align empty strings} \\ V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
		0	0	0	0	0	0
<b>A</b>		0	1				
<b>A</b>		0					
<b>G</b>		0					
<b>G</b>		0					
<b>C</b>		0					
<b>C</b>		0					

# Smith-Waterman

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(i, j) = \max \begin{cases} 0 & \text{align empty strings} \\ V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$

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

Annotations in the table:
 

- Green arrows:  $+\delta(A, A)$  (diagonal from (1,2) to (2,3)) and  $+\delta(A, A)$  (diagonal from (2,2) to (1,3)).
- Blue arrow:  $+\delta(A, -)$  (horizontal from (2,2) to (2,3)).
- Red arrow:  $+\delta(-, A)$  (vertical from (2,3) to (1,3)).
- Yellow dot:  $0$  at cell (2,4).

# Smith-Waterman

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(i, j) = \max \begin{cases} 0 & \text{align empty strings} \\ V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$

		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
		0	0	0	0	0	0
<b>A</b>		0	1	1			
<b>A</b>		0					
<b>G</b>		0					
<b>G</b>		0					
<b>C</b>		0					
<b>C</b>		0					



# Smith-Waterman

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(i, j) = \max \begin{cases} 0 & \text{align empty strings} \\ V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$

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

Annotations on the table:
 

- Green arrows pointing from (1,2) to (0,1), (1,3) to (0,2), and (1,4) to (0,3).
- Blue arrow pointing from (1,3) to (1,2) labeled  $+\delta(A, -)$ .
- Red arrow pointing from (1,4) to (1,3) labeled  $+\delta(A, C)$ .
- Red arrow pointing from (1,4) to (0,4) labeled  $+\delta(-, C)$ .
- Yellow dot at (1,4) with a yellow '0' below it.

# Smith-Waterman

$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(i, j) = \max \begin{cases} 0 & \text{align empty strings} \\ V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$

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

# Smith-Waterman

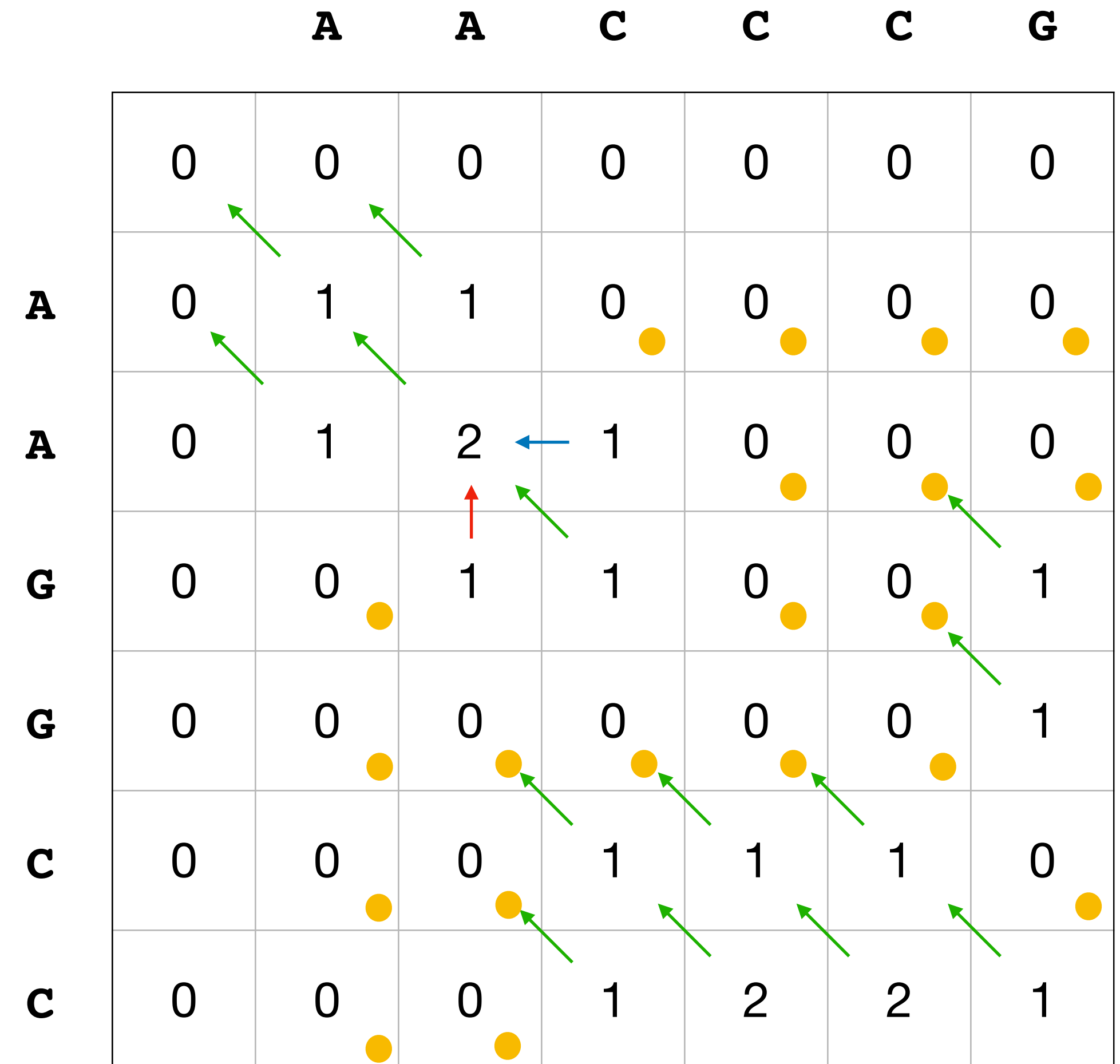
$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

$$V(i, j) = \max \begin{cases} 0 & \text{align empty strings} \\ V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ V(i-1, j) + \delta(S[i], -) & \text{delete} \\ V(i, j-1) + \delta(-, T[j]) & \text{insert} \end{cases}$$



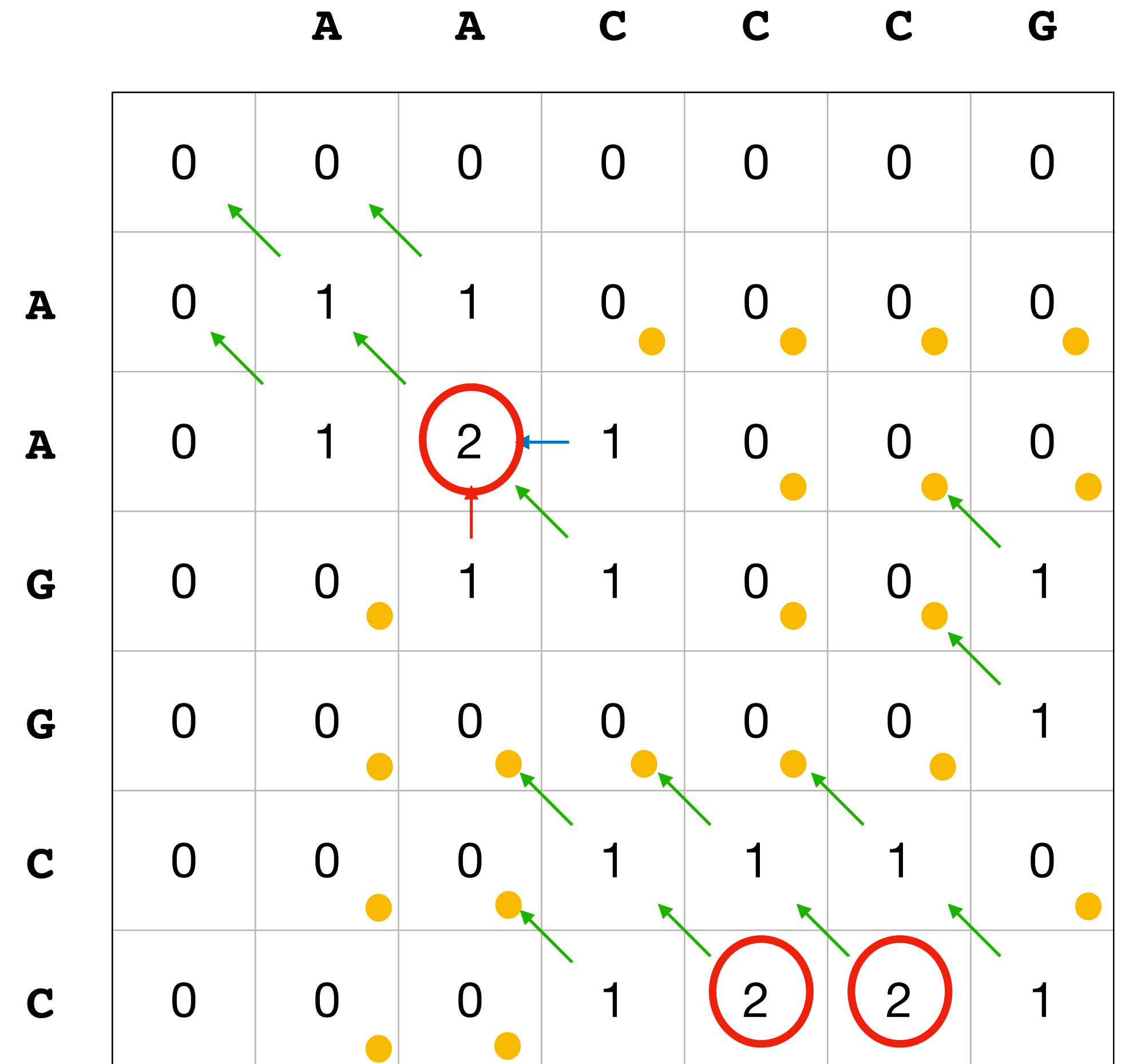
# Smith-Waterman

$\delta(-,x) = -1$  for  $x \in \Sigma$

$\delta(x,-) = -1$  for  $x \in \Sigma$

$\delta(x,y) = 1$  for  $y = x$

$\delta(x,y) = -1$  for  $y \neq x$



# Smith-Waterman

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

		A	A	C	C	C	G
	0	0	0	0	0	0	0
A	0	1	1	0	0	0	0
A	0	1	2	1	0	0	0
G	0	0	1	1	0	0	1
G	0	0	0	0	0	0	1
c	0	0	0	1	1	1	0
c	0	0	0	1	2	2	1

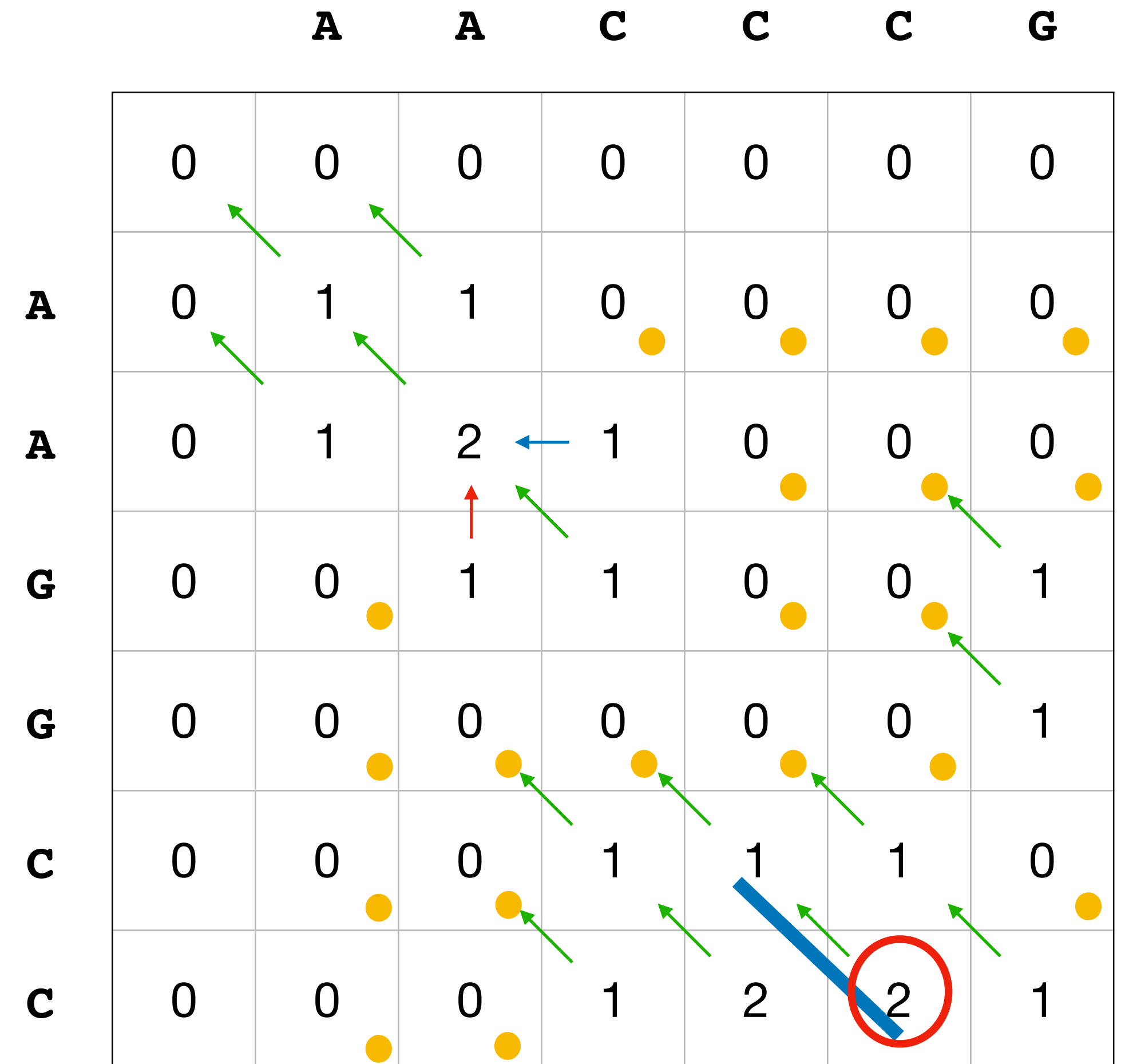
# Smith-Waterman

$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$



# Smith-Waterman

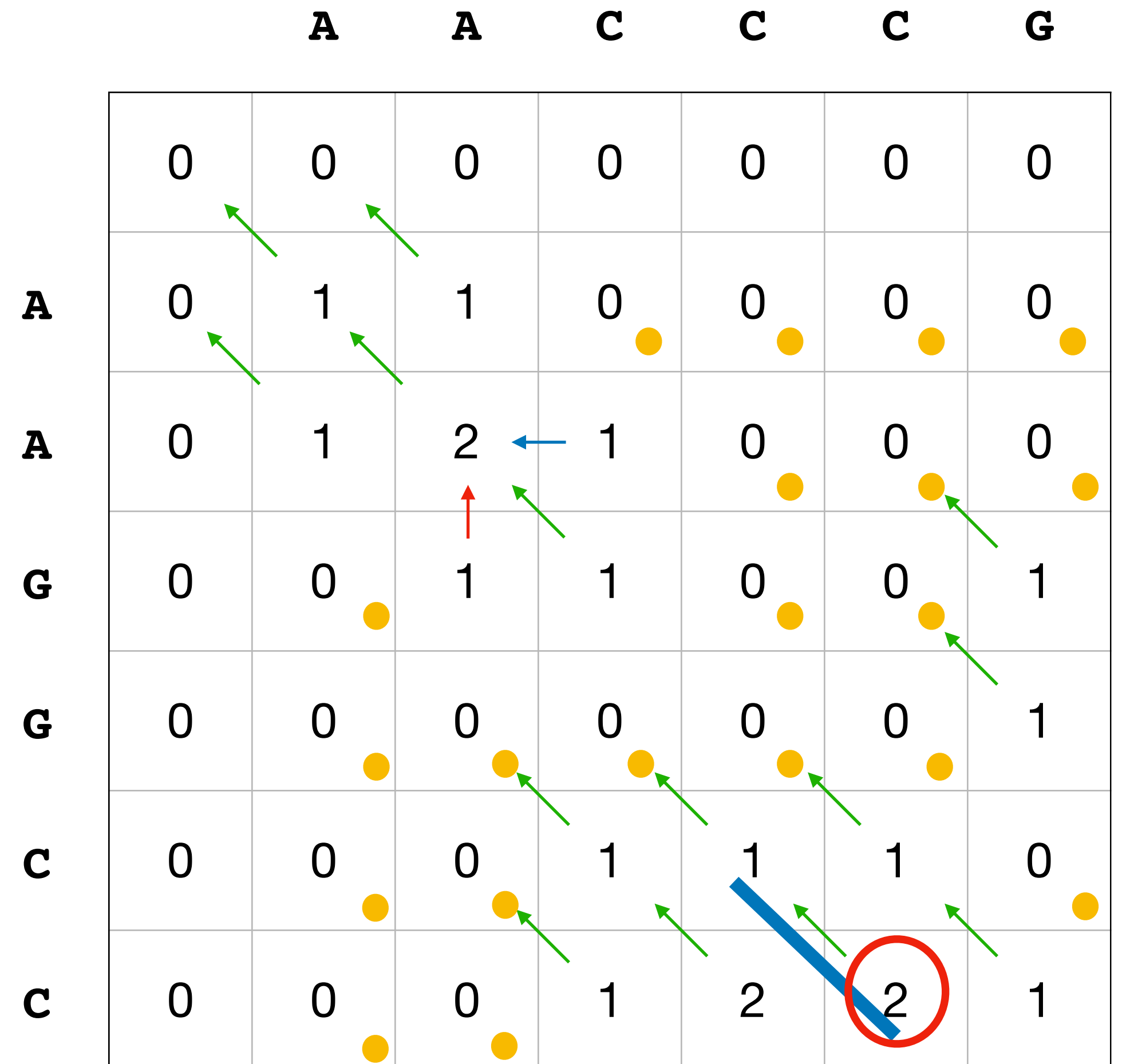
$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

C  
C



# Smith-Waterman

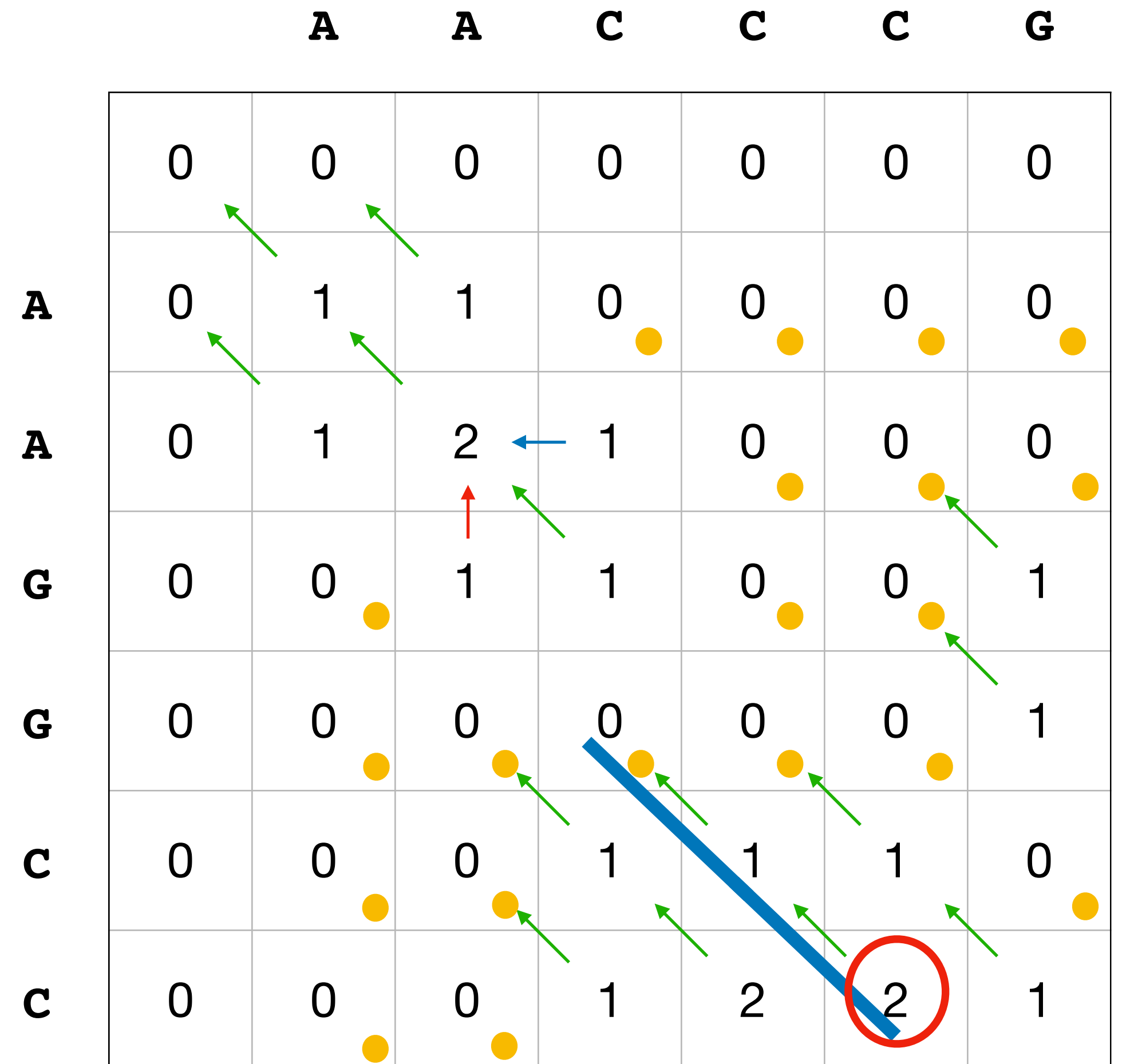
$$\delta(-,x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,-) = -1 \text{ for } x \in \Sigma$$

$$\delta(x,y) = 1 \text{ for } y = x$$

$$\delta(x,y) = -1 \text{ for } y \neq x$$

C  
C





# Smith-Waterman

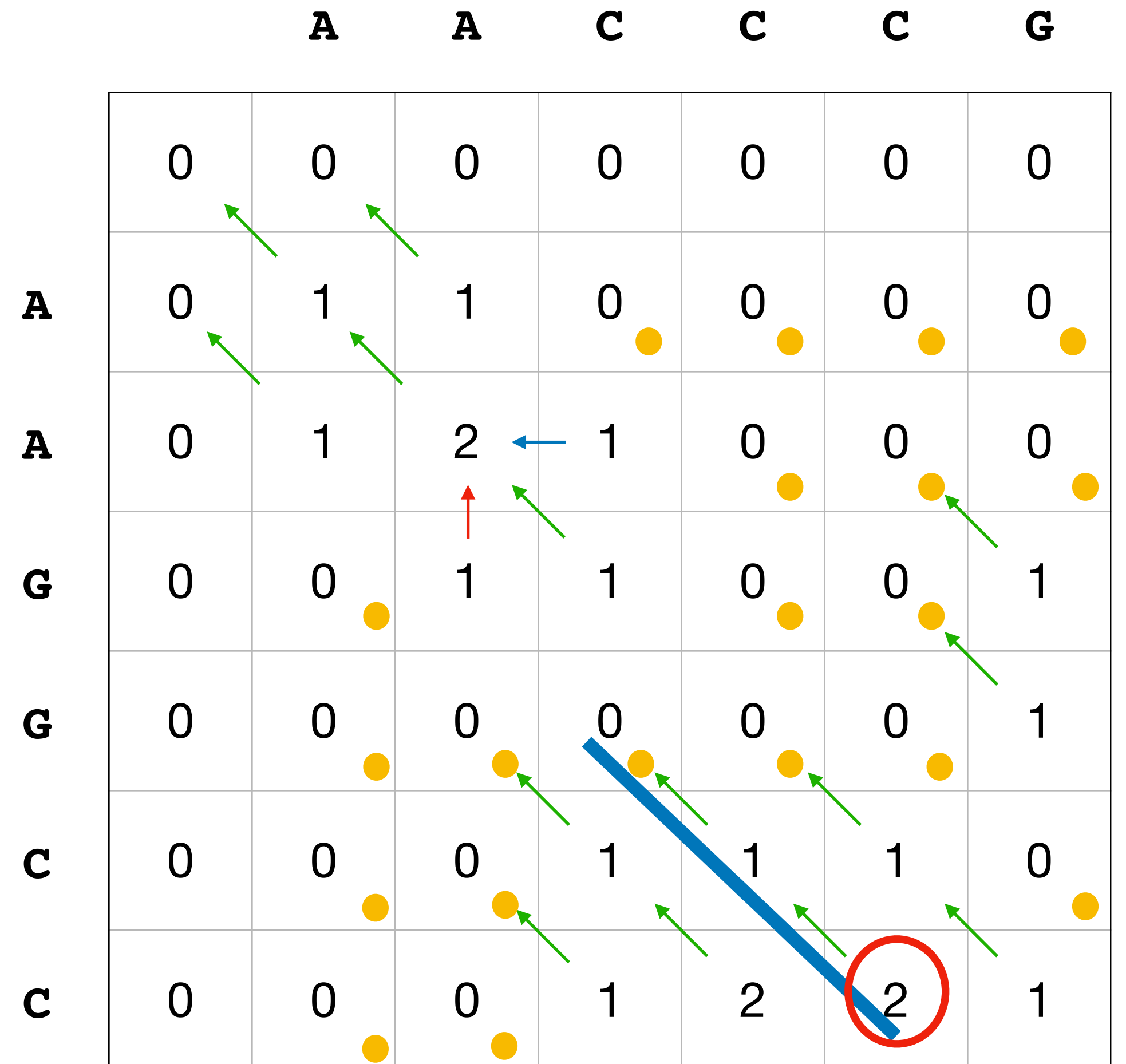
$$\delta(-, x) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, -) = -1 \text{ for } x \in \Sigma$$

$$\delta(x, y) = 1 \text{ for } y = x$$

$$\delta(x, y) = -1 \text{ for } y \neq x$$

C C  
C C



# Semi-global Alignment

- In some cases one of the sequence may be smaller than the other
  - in biology, this may be due to sequencing ending early or some process in the cell
- In this case, we may want to not penalize gaps at the beginning or end of one of the sequences
- How can this be done using a slight adjustment to the Myers-Miller algorithm?

# Semi-global Alignment

Ignored spaces

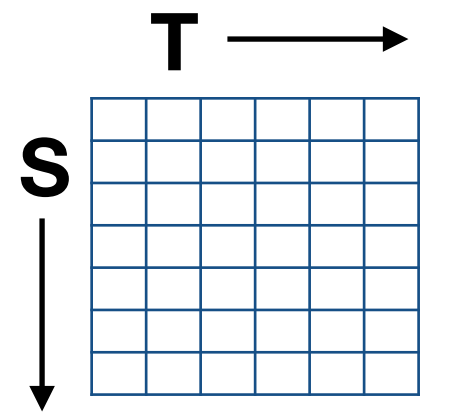
Modification

The beginning of S

The end of S

The beginning of T

The end of T



# Semi-global Alignment

Ignored spaces

Modification

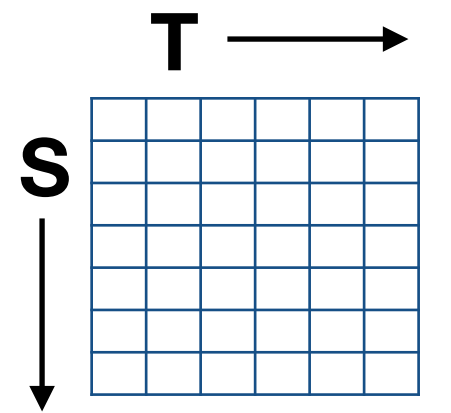
The beginning of S

Initialize column 0 to 0s

The end of S

The beginning of T

The end of T



# Semi-global Alignment

Ignored spaces

Modification

The beginning of S

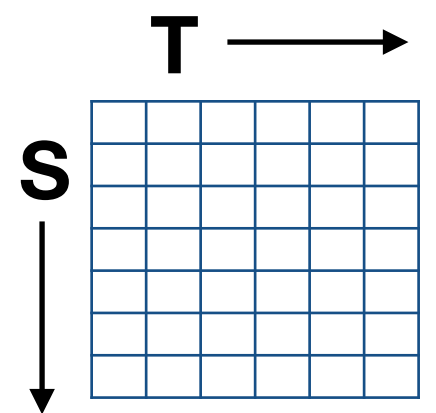
Initialize column 0 to 0s

The end of S

Search for the maximum value in the last column

The beginning of T

The end of T



# Semi-global Alignment

Ignored spaces

Modification

The beginning of S

Initialize column 0 to 0s

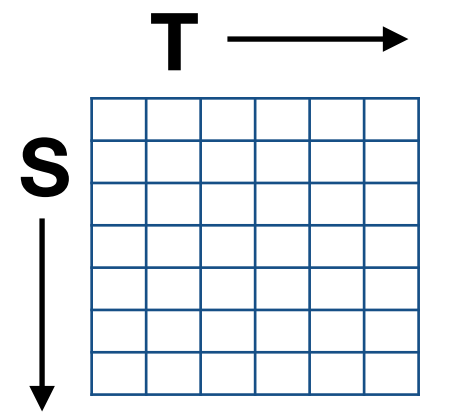
The end of S

Search for the maximum value in the last column

The beginning of T

Initialize row 0 to 0s

The end of T



# Semi-global Alignment

Ignored spaces

Modification

The beginning of S

Initialize column 0 to 0s

The end of S

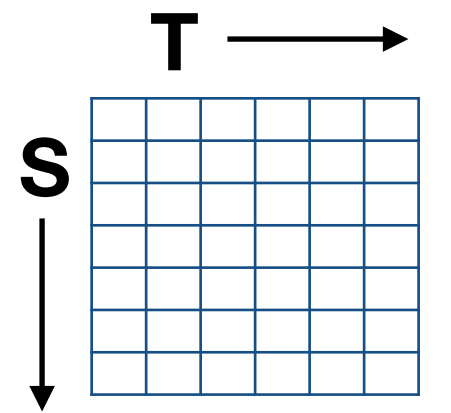
Search for the maximum value in the last column

The beginning of T

Initialize row 0 to 0s

The end of T

Search for the maximum value in the last row



# Lets talk about gaps!

- Up to now inserting a gap character (-) was always a constant cost operation
- In real life (i.e. in biology) starting an inserting 10 bases and inserting 12 bases causes close the same amount of disruption, whereas inserting 0 bases and inserting 2 is a big difference
- So we want to score alignments with this in mind
- Generally, for a gap (set of contiguous insertions or deletions) of length  $k$ , we want the score to be some function  $f(k)$  of the length



# General Gap Costs

- Given that a gap of length  $k$  receives a score  $f(k)$ , in global alignment we change the recursion to the following:

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \delta(S[i], T[j]) & \text{match/mismatch} \\ \max_{0 \leq h \leq j-1} \{ V(i, h) - f(j-h) \} & \text{insert } T[h+1 \dots j] \\ \max_{0 \leq h \leq i-1} \{ V(h, j) - f(i-h) \} & \text{delete } S[h+1 \dots i] \end{cases}$$

- And initialization changes slightly:

$$V(0, 0) = 0$$

$$V(0, j) = -f(j)$$

$$V(i, 0) = -f(i)$$

- This change also increases the running time. Each entry now takes  $O(m+n)$ -time, therefore the total running time is  $O(mn(m+n))$

# Affine Gap Costs

- The one everyone uses!
- Attributed to Gotoh [1982]
- Define the function  $f_{a,b}(k) =: a + b * k$  where  $a$  and  $b$  are tunable parameters (if  $a=0$ , this is the same as before)
- Can still be solved in  $O(mn)$ -time and  $O(mn)$ -space, but we need a bit more sophistication

# Gotoh's Algorithm

- Define 3  $n \times m$  matrices:
  - $G$  -- the alignment score for alignments that end in a match (the last column)
  - $F$  -- the alignment score for alignments that end in an insertion
  - $E$  -- the alignment score for alignments that end in a deletion

# Gotoh's Algorithm

## Recursion

$$F(i, j) = \max \begin{cases} F(i-1, j) - b \\ G(i-1, j) - f_{a,b}(1) \end{cases}$$

$$E(i, j) = \max \begin{cases} E(i, j-1) - b \\ G(i, j-1) - f_{a,b}(1) \end{cases}$$

$$G(i, j) = \max \begin{cases} G(i-1, j-1) + \delta(S[i], T[j]) \\ E(i, j) \\ F(i, j) \end{cases}$$

## Initialization

$$G(0, j) = E(0, j) = -1f_{a,b}(j)$$

$$G(i, 0) = F(i, 0) = -1f_{a,b}(i)$$

$$E(i, 0) = -\infty$$

$$F(0, j) = -\infty$$

# Gotoh's Algorithm

$$G(0,j) = E(0,j) = -1f_{a,b}(j)$$

$$G(i,0) = F(i,0) = -1f_{a,b}(i)$$

$$E(i,0) = -\infty$$

$$F(0,j) = -\infty$$

<b>E</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
<b>A</b>	-4					
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>G</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4					
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>F</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	-4	-5	-6	-7	-8	-9
<b>A</b>	$-\infty$					
<b>A</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>C</b>	$-\infty$					
<b>C</b>	$-\infty$					

$$\delta(x,y) = 10 \text{ for } y = x$$

$$\delta(x,y) = -2 \text{ for } y \neq x$$

$$E(i,j) = \max \begin{cases} E(i,j-1) - b \\ G(i,j-1) - f_{a,b}(1) \end{cases}$$

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + \delta(S[i], T[j]) \\ E(i,j) \\ F(i,j) \end{cases}$$

$$F(i,j) = \max \begin{cases} F(i-1,j) - b \\ G(i-1,j) - f_{a,b}(1) \end{cases}$$

$$a = 3$$

$$b = 1$$

# Gotoh's Algorithm

$$G(0,j) = E(0,j) = -1f_{a,b}(j)$$

$$G(i,0) = F(i,0) = -1f_{a,b}(i)$$

$$E(i,0) = -\infty$$

$$F(0,j) = -\infty$$

<b>E</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
<b>A</b>	-4					
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>G</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4					
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>F</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	-4	-5	-6	-7	-8	-9
<b>A</b>	$-\infty$					
<b>A</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>C</b>	$-\infty$					
<b>C</b>	$-\infty$					

$$\delta(x,y) = 10 \text{ for } y = x$$

$$\delta(x,y) = -2 \text{ for } y \neq x$$

$$E(i,j) = \max \begin{cases} E(i,j-1) - b \\ G(i,j-1) - f_{a,b}(1) \end{cases}$$

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + \delta(S[i], T[j]) \\ E(i,j) \\ F(i,j) \end{cases}$$

$$F(i,j) = \max \begin{cases} F(i-1,j) - b \\ G(i-1,j) - f_{a,b}(1) \end{cases}$$

$$a = 3$$

$$b = 1$$

# Gotoh's Algorithm

$$G(0,j) = E(0,j) = -1f_{a,b}(j)$$

$$G(i,0) = F(i,0) = -1f_{a,b}(i)$$

$$E(i,0) = -\infty$$

$$F(0,j) = -\infty$$

<b>E</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
<b>A</b>	-4					
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

-b -a

<b>G</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4					
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>F</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	-4	-5	-6	-7	-8	-9
<b>A</b>	$-\infty$					
<b>A</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>C</b>	$-\infty$					
<b>C</b>	$-\infty$					

$$\delta(x,y) = 10 \text{ for } y = x$$

$$\delta(x,y) = -2 \text{ for } y \neq x$$

$$E(i,j) = \max \begin{cases} E(i,j-1) - b \\ G(i,j-1) - f_{a,b}(1) \end{cases}$$

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + \delta(S[i], T[j]) \\ E(i,j) \\ F(i,j) \end{cases}$$

$$F(i,j) = \max \begin{cases} F(i-1,j) - b \\ G(i-1,j) - f_{a,b}(1) \end{cases}$$

$$a = 3$$

$$b = 1$$

# Gotoh's Algorithm

$$G(0,j) = E(0,j) = -1f_{a,b}(j)$$

$$G(i,0) = F(i,0) = -1f_{a,b}(i)$$

$$E(i,0) = -\infty$$

$$F(0,j) = -\infty$$

<b>E</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
<b>A</b>	-4	-8				
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

-b -a

<b>G</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4					
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>F</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	-4	-5	-6	-7	-8	-9
<b>A</b>	$-\infty$					
<b>A</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>C</b>	$-\infty$					
<b>C</b>	$-\infty$					

$$\delta(x,y) = 10 \text{ for } y = x$$

$$\delta(x,y) = -2 \text{ for } y \neq x$$

$$E(i,j) = \max \begin{cases} E(i,j-1) - b \\ G(i,j-1) - f_{a,b}(1) \end{cases}$$

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + \delta(S[i], T[j]) \\ E(i,j) \\ F(i,j) \end{cases}$$

$$F(i,j) = \max \begin{cases} F(i-1,j) - b \\ G(i-1,j) - f_{a,b}(1) \end{cases}$$

$$a = 3$$

$$b = 1$$



# Gotoh's Algorithm

$$G(0,j) = E(0,j) = -1f_{a,b}(j)$$

$$G(i,0) = F(i,0) = -1f_{a,b}(i)$$

$$E(i,0) = -\infty$$

$$F(0,j) = -\infty$$

<b>E</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
<b>A</b>	-4	-8				
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>G</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4					
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>F</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	-4	-5	-6	-7	-8	-9
<b>A</b>	$-\infty$					
<b>A</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>C</b>	$-\infty$					
<b>C</b>	$-\infty$					

$$\delta(x,y) = 10 \text{ for } y = x$$

$$\delta(x,y) = -2 \text{ for } y \neq x$$

$$E(i,j) = \max \begin{cases} E(i,j-1) - b \\ G(i,j-1) - f_{a,b}(1) \end{cases}$$

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + \delta(S[i], T[j]) \\ E(i,j) \\ F(i,j) \end{cases}$$

$$F(i,j) = \max \begin{cases} F(i-1,j) - b \\ G(i-1,j) - f_{a,b}(1) \end{cases}$$

$$a = 3$$

$$b = 1$$

# Gotoh's Algorithm

$$G(0,j) = E(0,j) = -1f_{a,b}(j)$$

$$G(i,0) = F(i,0) = -1f_{a,b}(i)$$

$$E(i,0) = -\infty$$

$$F(0,j) = -\infty$$

<b>E</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
<b>A</b>	-4	-8				
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>G</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4					
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>F</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	-4	-5	-6	-7	-8	-9
<b>A</b>	$-\infty$					
<b>A</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>C</b>	$-\infty$					
<b>C</b>	$-\infty$					

$$\delta(x,y) = 10 \text{ for } y = x$$

$$\delta(x,y) = -2 \text{ for } y \neq x$$

$$E(i,j) = \max \begin{cases} E(i,j-1) - b \\ G(i,j-1) - f_{a,b}(1) \end{cases}$$

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + \delta(S[i], T[j]) \\ E(i,j) \\ F(i,j) \end{cases}$$

$$F(i,j) = \max \begin{cases} F(i-1,j) - b \\ G(i-1,j) - f_{a,b}(1) \end{cases}$$

$$a = 3$$

$$b = 1$$

# Gotoh's Algorithm

$$G(0,j) = E(0,j) = -1f_{a,b}(j)$$

$$G(i,0) = F(i,0) = -1f_{a,b}(i)$$

$$E(i,0) = -\infty$$

$$F(0,j) = -\infty$$

<b>E</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
<b>A</b>	-4	-8				
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>G</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4					
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>F</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	-4	-5	-6	-7	-8	-9
<b>A</b>	$-\infty$					
<b>A</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>C</b>	$-\infty$					
<b>C</b>	$-\infty$					

$$\delta(x,y) = 10 \text{ for } y = x$$

$$\delta(x,y) = -2 \text{ for } y \neq x$$

$$E(i,j) = \max \begin{cases} E(i,j-1) - b \\ G(i,j-1) - f_{a,b}(1) \end{cases}$$

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + \delta(S[i], T[j]) \\ E(i,j) \\ F(i,j) \end{cases}$$

$$F(i,j) = \max \begin{cases} F(i-1,j) - b \\ G(i-1,j) - f_{a,b}(1) \end{cases}$$

$$a = 3$$

$$b = 1$$

# Gotoh's Algorithm

$$G(0,j) = E(0,j) = -1f_{a,b}(j)$$

$$G(i,0) = F(i,0) = -1f_{a,b}(i)$$

$$E(i,0) = -\infty$$

$$F(0,j) = -\infty$$

<b>E</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
<b>A</b>	-4	-8				
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>G</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4					
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>F</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	-4	-5	-6	-7	-8	-9
<b>A</b>	$-\infty$	-8				
<b>A</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>C</b>	$-\infty$					
<b>C</b>	$-\infty$					

$$\delta(x,y) = 10 \text{ for } y = x$$

$$\delta(x,y) = -2 \text{ for } y \neq x$$

$$E(i,j) = \max \begin{cases} E(i,j-1) - b \\ G(i,j-1) - f_{a,b}(1) \end{cases}$$

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + \delta(S[i], T[j]) \\ E(i,j) \\ F(i,j) \end{cases}$$

$$F(i,j) = \max \begin{cases} F(i-1,j) - b \\ G(i-1,j) - f_{a,b}(1) \end{cases}$$

$$a = 3$$

$$b = 1$$

# Gotoh's Algorithm

$$G(0,j) = E(0,j) = -1f_{a,b}(j)$$

$$G(i,0) = F(i,0) = -1f_{a,b}(i)$$

$$E(i,0) = -\infty$$

$$F(0,j) = -\infty$$

<b>E</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
<b>A</b>	-4	-8				
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>G</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4					
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>F</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	-4	-5	-6	-7	-8	-9
<b>A</b>	$-\infty$	-8				
<b>A</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>C</b>	$-\infty$					
<b>C</b>	$-\infty$					

$$\delta(x,y) = 10 \text{ for } y = x$$

$$\delta(x,y) = -2 \text{ for } y \neq x$$

$$E(i,j) = \max \begin{cases} E(i,j-1) - b \\ G(i,j-1) - f_{a,b}(1) \end{cases}$$

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + \delta(S[i], T[j]) \\ E(i,j) \\ F(i,j) \end{cases}$$

$$F(i,j) = \max \begin{cases} F(i-1,j) - b \\ G(i-1,j) - f_{a,b}(1) \end{cases}$$

$$a = 3$$

$$b = 1$$

# Gotoh's Algorithm

$$G(0,j) = E(0,j) = -1f_{a,b}(j)$$

$$G(i,0) = F(i,0) = -1f_{a,b}(i)$$

$$E(i,0) = -\infty$$

$$F(0,j) = -\infty$$

<b>E</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
<b>A</b>	-4	-8				
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>G</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4					
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>F</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	-4	-5	-6	-7	-8	-9
<b>A</b>	$-\infty$	-8				
<b>A</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>C</b>	$-\infty$					
<b>C</b>	$-\infty$					

$$\delta(x,y) = 10 \text{ for } y = x$$

$$\delta(x,y) = -2 \text{ for } y \neq x$$

$$E(i,j) = \max \begin{cases} E(i,j-1) - b \\ G(i,j-1) - f_{a,b}(1) \end{cases}$$

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + \delta(S[i], T[j]) \\ E(i,j) \\ F(i,j) \end{cases}$$

$$F(i,j) = \max \begin{cases} F(i-1,j) - b \\ G(i-1,j) - f_{a,b}(1) \end{cases}$$

$$a = 3$$

$$b = 1$$

# Gotoh's Algorithm

$$G(0,j) = E(0,j) = -1f_{a,b}(j)$$

$$G(i,0) = F(i,0) = -1f_{a,b}(i)$$

$$E(i,0) = -\infty$$

$$F(0,j) = -\infty$$

<b>E</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
<b>A</b>	-4	-8				
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>G</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4					
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>F</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	-4	-5	-6	-7	-8	-9
<b>A</b>	$-\infty$	-8				
<b>A</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>C</b>	$-\infty$					
<b>C</b>	$-\infty$					

$$\delta(x,y) = 10 \text{ for } y = x$$

$$\delta(x,y) = -2 \text{ for } y \neq x$$

$$E(i,j) = \max \begin{cases} E(i,j-1) - b \\ G(i,j-1) - f_{a,b}(1) \end{cases}$$

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + \delta(S[i], T[j]) \\ E(i,j) \\ F(i,j) \end{cases}$$

$$F(i,j) = \max \begin{cases} F(i-1,j) - b \\ G(i-1,j) - f_{a,b}(1) \end{cases}$$

$$a = 3$$

$$b = 1$$

# Gotoh's Algorithm

$$G(0,j) = E(0,j) = -1f_{a,b}(j)$$

$$G(i,0) = F(i,0) = -1f_{a,b}(i)$$

$$E(i,0) = -\infty$$

$$F(0,j) = -\infty$$

<b>E</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
<b>A</b>	-4	-8				
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>G</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4					
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>F</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	-4	-5	-6	-7	-8	-9
<b>A</b>	$-\infty$	-8				
<b>A</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>C</b>	$-\infty$					
<b>C</b>	$-\infty$					

$$\delta(x,y) = 10 \text{ for } y = x$$

$$\delta(x,y) = -2 \text{ for } y \neq x$$

$$E(i,j) = \max \begin{cases} E(i,j-1) - b \\ G(i,j-1) - f_{a,b}(1) \end{cases}$$

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + \delta(S[i], T[j]) \\ E(i,j) \\ F(i,j) \end{cases}$$

$$F(i,j) = \max \begin{cases} F(i-1,j) - b \\ G(i-1,j) - f_{a,b}(1) \end{cases}$$

$$a = 3$$

$$b = 1$$



# Gotoh's Algorithm

$$G(0,j) = E(0,j) = -1f_{a,b}(j)$$

$$G(i,0) = F(i,0) = -1f_{a,b}(i)$$

$$E(i,0) = -\infty$$

$$F(0,j) = -\infty$$

<b>E</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
<b>A</b>	-4	-8				
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>G</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4	10				
<b>A</b>	-5					
<b>G</b>	-6					
<b>G</b>	-7					
<b>C</b>	-8					
<b>C</b>	-9					

<b>F</b>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
$-\infty$	-4	-5	-6	-7	-8	-9
<b>A</b>	$-\infty$	-8				
<b>A</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>G</b>	$-\infty$					
<b>C</b>	$-\infty$					
<b>C</b>	$-\infty$					

$$\delta(x,y) = 10 \text{ for } y = x$$

$$\delta(x,y) = -2 \text{ for } y \neq x$$

$$E(i,j) = \max \begin{cases} E(i,j-1) - b \\ G(i,j-1) - f_{a,b}(1) \end{cases}$$

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + \delta(S[i], T[j]) \\ E(i,j) \\ F(i,j) \end{cases}$$

$$F(i,j) = \max \begin{cases} F(i-1,j) - b \\ G(i-1,j) - f_{a,b}(1) \end{cases}$$

$$a = 3$$

$$b = 1$$

# Gotoh's Algorithm

$$G(0,j) = E(0,j) = -1f_{a,b}(j)$$

$$G(i,0) = F(i,0) = -1f_{a,b}(i)$$

$$E(i,0) = -\infty$$

$$F(0,j) = -\infty$$

<i>E</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
<b>A</b>	-4	-8	-9	-10	-11	-12	-13
<b>A</b>	-5	6	2	1	0	-1	-2
<b>G</b>	-6	5	16	12	11	10	9
<b>G</b>	-7	4	15	14	10	9	20
<b>C</b>	-8	3	14	13	12	8	19
<b>C</b>	-9	2	13	21	20	22	18

<i>G</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4	10	6	5	4	3	2
<b>A</b>	-5	6	20	16	15	14	13
<b>G</b>	-6	5	16	18	14	13	24
<b>G</b>	-7	4	15	14	16	12	23
<b>C</b>	-8	3	14	25	24	26	22
<b>C</b>	-9	2	13	24	35	34	30

<i>F</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	$-\infty$	-4	-5	-6	-7	-8	-9
<b>A</b>	$-\infty$	-8	6	5	4	3	2
<b>A</b>	$-\infty$	-9	2	16	15	14	13
<b>G</b>	$-\infty$	-10	1	15	14	13	12
<b>G</b>	$-\infty$	-11	0	11	10	12	11
<b>C</b>	$-\infty$	-12	-1	10	21	20	22
<b>C</b>	$-\infty$	-13	-2	8	20	31	30

$$\delta(x,y) = 10 \text{ for } y = x$$

$$\delta(x,y) = -2 \text{ for } y \neq x$$

$$E(i,j) = \max \begin{cases} E(i,j-1) - b \\ G(i,j-1) - f_{a,b}(1) \end{cases}$$

$$G(i,j) = \max \begin{cases} G(i-1,j-1) + \delta(S[i], T[j]) \\ E(i,j) \\ F(i,j) \end{cases}$$

$$F(i,j) = \max \begin{cases} F(i-1,j) - b \\ G(i-1,j) - f_{a,b}(1) \end{cases}$$

$$a = 3$$

$$b = 1$$

# Gotoh's Algorithm

<i>E</i>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	-∞	-∞	-∞	-∞	-∞	-∞
<b>A</b>	-4	-8	-9	-10	-11	-12
<b>A</b>	-5	6	2	1	0	-1
<b>G</b>	-6	5	16	12	11	10
<b>G</b>	-7	4	15	14	10	9
<b>C</b>	-8	3	14	13	12	8
<b>C</b>	-9	2	13	21	20	22

<i>G</i>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	0	-4	-5	-6	-7	-8
<b>A</b>	-4	10	6	5	4	3
<b>A</b>	-5	6	20	16	15	14
<b>G</b>	-6	5	16	18	14	13
<b>G</b>	-7	4	15	14	16	12
<b>C</b>	-8	3	14	25	24	26
<b>C</b>	-9	2	13	24	35	34

<i>F</i>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	-∞	-4	-5	-6	-7	-8
<b>A</b>	-∞	-8	6	5	4	3
<b>A</b>	-∞	-9	2	16	15	14
<b>G</b>	-∞	-10	1	15	14	13
<b>G</b>	-∞	-11	0	11	10	12
<b>C</b>	-∞	-12	-1	10	21	20
<b>C</b>	-∞	-13	-2	8	20	31

(6,6)

# Gotoh's Algorithm

<i>E</i>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>	
	-∞	-∞	-∞	-∞	-∞	-∞	
<b>A</b>	-4	-8	-9	-10	-11	-12	-13
<b>A</b>	-5	6	2	1	0	-1	-2
<b>G</b>	-6	5	16	12	11	10	9
<b>G</b>	-7	4	15	14	10	9	20
<b>C</b>	-8	3	14	13	12	8	19
<b>C</b>	-9	2	13	21	20	22	18

<i>G</i>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>	
	0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4	10	6	5	4	3	2
<b>A</b>	-5	6	20	16	15	14	13
<b>G</b>	-6	5	16	18	14	13	24
<b>G</b>	-7	4	15	14	16	12	23
<b>C</b>	-8	3	14	25	24	26	22
<b>C</b>	-9	2	13	24	35	34	30

<i>F</i>	<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>	
	-∞	-4	-5	-6	-7	-8	-9
<b>A</b>	-∞	-8	6	5	4	3	2
<b>A</b>	-∞	-9	2	16	15	14	13
<b>G</b>	-∞	-10	1	15	14	13	12
<b>G</b>	-∞	-11	0	11	10	12	11
<b>C</b>	-∞	-12	-1	10	21	20	22
<b>C</b>	-∞	-13	-2	8	20	31	30

(6,6)

# Gotoh's Algorithm

<i>E</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	-∞	-∞	-∞	-∞	-∞	-∞	-∞
<b>A</b>	-4	-8	-9	-10	-11	-12	-13
<b>A</b>	-5	6	2	1	0	-1	-2
<b>G</b>	-6	5	16	12	11	10	9
<b>G</b>	-7	4	15	14	10	9	20
<b>C</b>	-8	3	14	13	12	8	19
<b>C</b>	-9	2	13	21	20	22	18

<i>G</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4	10	6	5	4	3	2
<b>A</b>	-5	6	20	16	15	14	13
<b>G</b>	-6	5	16	18	14	13	24
<b>G</b>	-7	4	15	14	16	12	23
<b>C</b>	-8	3	14	25	24	26	22
<b>C</b>	-9	2	13	24	35	34	30

<i>F</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	-∞	-4	-5	-6	-7	-8	-9
<b>A</b>	-∞	-8	6	5	4	3	2
<b>A</b>	-∞	-9	2	16	15	14	13
<b>G</b>	-∞	-10	1	15	14	13	12
<b>G</b>	-∞	-11	0	11	10	12	11
<b>C</b>	-∞	-12	-1	10	21	20	22
<b>C</b>	-∞	-13	-2	8	20	31	30

**G**  
-

(5,6)

# Gotoh's Algorithm

<i>E</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
<b>A</b>	-4	-8	-9	-10	-11	-12	-13
<b>A</b>	-5	6	2	1	0	-1	-2
<b>G</b>	-6	5	16	12	11	10	9
<b>G</b>	-7	4	15	14	10	9	20
<b>C</b>	-8	3	14	13	12	8	19
<b>C</b>	-9	2	13	21	20	22	18

<i>G</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4	10	6	5	4	3	2
<b>A</b>	-5	6	20	16	15	14	13
<b>G</b>	-6	5	16	18	14	13	24
<b>G</b>	-7	4	15	14	16	12	23
<b>C</b>	-8	3	14	25	24	26	22
<b>C</b>	-9	2	13	24	35	34	30

<i>F</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	$-\infty$	-4	-5	-6	-7	-8	-9
<b>A</b>	$-\infty$	-8	6	5	4	3	2
<b>A</b>	$-\infty$	-9	2	16	15	14	13
<b>G</b>	$-\infty$	-10	1	15	14	13	12
<b>G</b>	$-\infty$	-11	0	11	10	12	11
<b>C</b>	$-\infty$	-12	-1	10	21	20	22
<b>C</b>	$-\infty$	-13	-2	8	20	31	30

**C G**  
**C -**

**(4,5)**

# Gotoh's Algorithm

<i>E</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	-∞	-∞	-∞	-∞	-∞	-∞	-∞
<b>A</b>	-4	-8	-9	-10	-11	-12	-13
<b>A</b>	-5	6	2	1	0	-1	-2
<b>G</b>	-6	5	16	12	11	10	9
<b>G</b>	-7	4	15	14	10	9	20
<b>C</b>	-8	3	14	13	12	8	19
<b>C</b>	-9	2	13	21	20	22	18

<i>G</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4	10	6	5	4	3	2
<b>A</b>	-5	6	20	16	15	14	13
<b>G</b>	-6	5	16	18	14	13	24
<b>G</b>	-7	4	15	14	16	12	23
<b>C</b>	-8	3	14	25	24	26	22
<b>C</b>	-9	2	13	24	35	34	30

<i>F</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	-∞	-4	-5	-6	-7	-8	-9
<b>A</b>	-∞	-8	6	5	4	3	2
<b>A</b>	-∞	-9	2	16	15	14	13
<b>G</b>	-∞	-10	1	15	14	13	12
<b>G</b>	-∞	-11	0	11	10	12	11
<b>C</b>	-∞	-12	-1	10	21	20	22
<b>C</b>	-∞	-13	-2	8	20	31	30

**C C G**  
**C C -**

**(3,4)**

# Gotoh's Algorithm

<i>E</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	-∞	-∞	-∞	-∞	-∞	-∞	-∞
<b>A</b>	-4	-8	-9	-10	-11	-12	-13
<b>A</b>	-5	6	2	1	0	-1	-2
<b>G</b>	-6	5	16	12	11	10	9
<b>G</b>	-7	4	15	14	10	9	20
<b>C</b>	-8	3	14	13	12	8	19
<b>C</b>	-9	2	13	21	20	22	18

<i>G</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4	10	6	5	4	3	2
<b>A</b>	-5	6	20	16	15	14	13
<b>G</b>	-6	5	16	18	14	13	24
<b>G</b>	-7	4	15	14	16	12	23
<b>C</b>	-8	3	14	25	24	26	22
<b>C</b>	-9	2	13	24	35	34	30

<i>F</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	-∞	-4	-5	-6	-7	-8	-9
<b>A</b>	-∞	-8	6	5	4	3	2
<b>A</b>	-∞	-9	2	16	15	14	13
<b>G</b>	-∞	-10	1	15	14	13	12
<b>G</b>	-∞	-11	0	11	10	12	11
<b>C</b>	-∞	-12	-1	10	21	20	22
<b>C</b>	-∞	-13	-2	8	20	31	30

**C C G**  
**C C -**

**(3,4)**



# Gotoh's Algorithm

<i>E</i>	A	A	C	C	C	G
	-∞	-∞	-∞	-∞	-∞	-∞
<b>A</b>	-4	-8	-9	-10	-11	-12
<b>A</b>	-5	6	2	1	0	-1
<b>G</b>	-6	5	16	12	11	10
<b>G</b>	-7	4	15	14	10	9
<b>C</b>	-8	3	14	13	12	8
<b>C</b>	-9	2	13	21	20	22

<i>G</i>	A	A	C	C	C	G
	0	-4	-5	-6	-7	-8
<b>A</b>	-4	10	6	5	4	3
<b>A</b>	-5	6	20	16	15	14
<b>G</b>	-6	5	16	18	14	13
<b>G</b>	-7	4	15	14	16	12
<b>C</b>	-8	3	14	25	24	26
<b>C</b>	-9	2	13	24	35	34

<i>F</i>	A	A	C	C	C	G
	-∞	-4	-5	-6	-7	-8
<b>A</b>	-∞	-8	6	5	4	3
<b>A</b>	-∞	-9	2	16	15	14
<b>G</b>	-∞	-10	1	15	14	13
<b>G</b>	-∞	-11	0	11	10	12
<b>C</b>	-∞	-12	-1	10	21	20
<b>C</b>	-∞	-13	-2	8	20	31

- C C G  
G C C -

(3,3)

# Gotoh's Algorithm

<i>E</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	-∞	-∞	-∞	-∞	-∞	-∞	-∞
<b>A</b>	-4	-8	-9	-10	-11	-12	-13
<b>A</b>	-5	6	2	1	0	-1	-2
<b>G</b>	-6	5	16	12	11	10	9
<b>G</b>	-7	4	15	14	10	9	20
<b>C</b>	-8	3	14	13	12	8	19
<b>C</b>	-9	2	13	21	20	22	18

<i>G</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4	10	6	5	4	3	2
<b>A</b>	-5	6	20	16	15	14	13
<b>G</b>	-6	5	16	18	14	13	24
<b>G</b>	-7	4	15	14	16	12	23
<b>C</b>	-8	3	14	25	24	26	22
<b>C</b>	-9	2	13	24	35	34	30

<i>F</i>		<b>A</b>	<b>A</b>	<b>C</b>	<b>C</b>	<b>C</b>	<b>G</b>
	-∞	-4	-5	-6	-7	-8	-9
<b>A</b>	-∞	-8	6	5	4	3	2
<b>A</b>	-∞	-9	2	16	15	14	13
<b>G</b>	-∞	-10	1	15	14	13	12
<b>G</b>	-∞	-11	0	11	10	12	11
<b>C</b>	-∞	-12	-1	10	21	20	22
<b>C</b>	-∞	-13	-2	8	20	31	30

**C - C C G**  
**G G C C -**

(2,2)

# Gotoh's Algorithm

<i>E</i>	A	A	C	C	C	G	
	-∞	-∞	-∞	-∞	-∞	-∞	
<b>A</b>	-4	-8	-9	-10	-11	-12	-13
<b>A</b>	-5	6	2	1	0	-1	-2
<b>G</b>	-6	5	16	12	11	10	9
<b>G</b>	-7	4	15	14	10	9	20
<b>C</b>	-8	3	14	13	12	8	19
<b>C</b>	-9	2	13	21	20	22	18

<i>G</i>	A	A	C	C	C	G	
	0	-4	-5	-6	-7	-8	-9
<b>A</b>	-4	10	6	5	4	3	2
<b>A</b>	-5	6	20	16	15	14	13
<b>G</b>	-6	5	16	18	14	13	24
<b>G</b>	-7	4	15	14	16	12	23
<b>C</b>	-8	3	14	25	24	26	22
<b>C</b>	-9	2	13	24	35	34	30

<i>F</i>	A	A	C	C	C	G	
	-∞	-4	-5	-6	-7	-8	-9
<b>A</b>	-∞	-8	6	5	4	3	2
<b>A</b>	-∞	-9	2	16	15	14	13
<b>G</b>	-∞	-10	1	15	14	13	12
<b>G</b>	-∞	-11	0	11	10	12	11
<b>C</b>	-∞	-12	-1	10	21	20	22
<b>C</b>	-∞	-13	-2	8	20	31	30

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

(1,1)

# Gotoh's Algorithm

<i>E</i>	A	A	C	C	C	G
	-∞	-∞	-∞	-∞	-∞	-∞
<b>A</b>	-4	-8	-9	-10	-11	-12
<b>A</b>	-5	6	2	1	0	-1
<b>G</b>	-6	5	16	12	11	10
<b>G</b>	-7	4	15	14	10	9
<b>C</b>	-8	3	14	13	12	8
<b>C</b>	-9	2	13	21	20	22

<i>G</i>	A	A	C	C	C	G
	0	-4	-5	-6	-7	-8
<b>A</b>	-4	10	6	5	4	3
<b>A</b>	-5	6	20	16	15	14
<b>G</b>	-6	5	16	18	14	13
<b>G</b>	-7	4	15	14	16	12
<b>C</b>	-8	3	14	25	24	26
<b>C</b>	-9	2	13	24	35	34

<i>F</i>	A	A	C	C	C	G
	-∞	-4	-5	-6	-7	-8
<b>A</b>	-∞	-8	6	5	4	3
<b>A</b>	-∞	-9	2	16	15	14
<b>G</b>	-∞	-10	1	15	14	13
<b>G</b>	-∞	-11	0	11	10	12
<b>C</b>	-∞	-12	-1	10	21	20
<b>C</b>	-∞	-13	-2	8	20	31

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

(0,0)

# Computing alignments in linear space

- Up to now, global, local, and semi-global alignments using fixed, general, and affine gap costs have all taken  $O(mn)$ -time and  $O(mn)$ -space
- For long sequences we may not be able to keep the whole table in main memory, therefore if the space can be reduced it would improve the practicality of the algorithms

# Computing alignments in linear space

- Up to now, global, local, and semi-global alignments using fixed, general, and affine gap costs have all taken  $O(mn)$ -time and  $O(mn)$ -space
- For long sequences we may not be able to keep the whole table in main memory, therefore if the space can be reduced it would improve the practicality of the algorithms
- Hershberg [1975] shows that global alignment can be computed in  $O(n+m)$ -space<sup>1</sup>

<sup>1</sup>Myers and Miller [1988] later show that using the same ideas, affine gap alignments can be computed in  $O(m)$ -space ( $m < n$ )

# Computing alignments in linear space

- Up to now, global, local, and semi-global alignments using fixed, general, and affine gap costs have all taken  $O(mn)$ -time and  $O(mn)$ -space
- For long sequences we may not be able to keep the whole table in main memory, therefore if the space can be reduced it would improve the practicality of the algorithms
- Hershberg [1975] shows that global alignment can be computed in  $O(n+m)$ -space<sup>1</sup>
- The key is that when computing  $V$ , (in row-major, left to right order) the value only depends on the row above, and the current values in the same row to the left

<sup>1</sup>Myers and Miller [1988] later show that using the same ideas, affine gap alignments can be computed in  $O(m)$ -space ( $m < n$ )

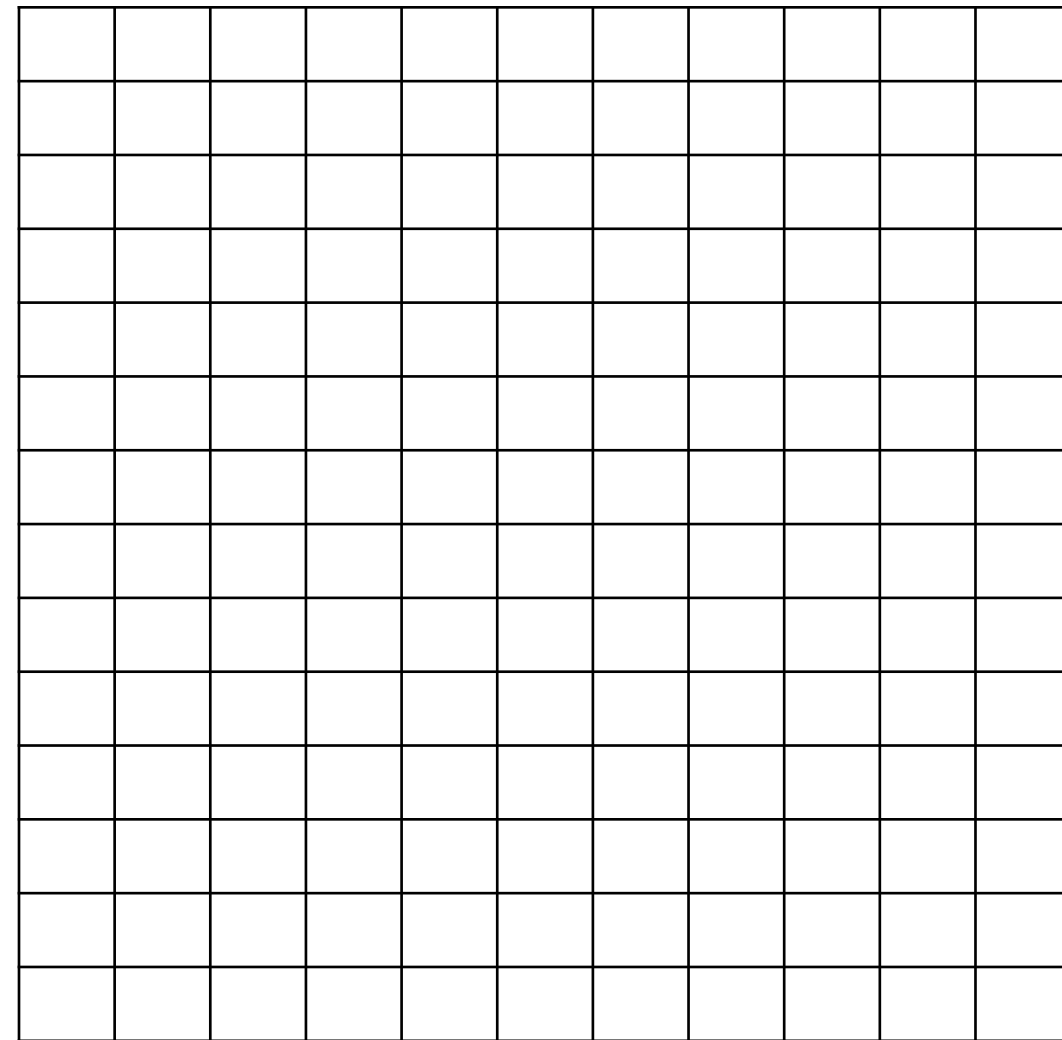
# Computing alignments in linear space

- Up to now, global, local, and semi-global alignments using fixed, general, and affine gap costs have all taken  $O(mn)$ -time and  $O(mn)$ -space
- For long sequences we may not be able to keep the whole table in main memory, therefore if the space can be reduced it would improve the practicality of the algorithms
- Hershberg [1975] shows that global alignment can be computed in  $O(n+m)$ -space<sup>1</sup>
- The key is that when computing  $V$ , (in row-major, left to right order) the value only depends on the row above, and the current values in the same row to the left
- But with only that observation you can find the best alignment **score** in linear space, but not the alignment itself

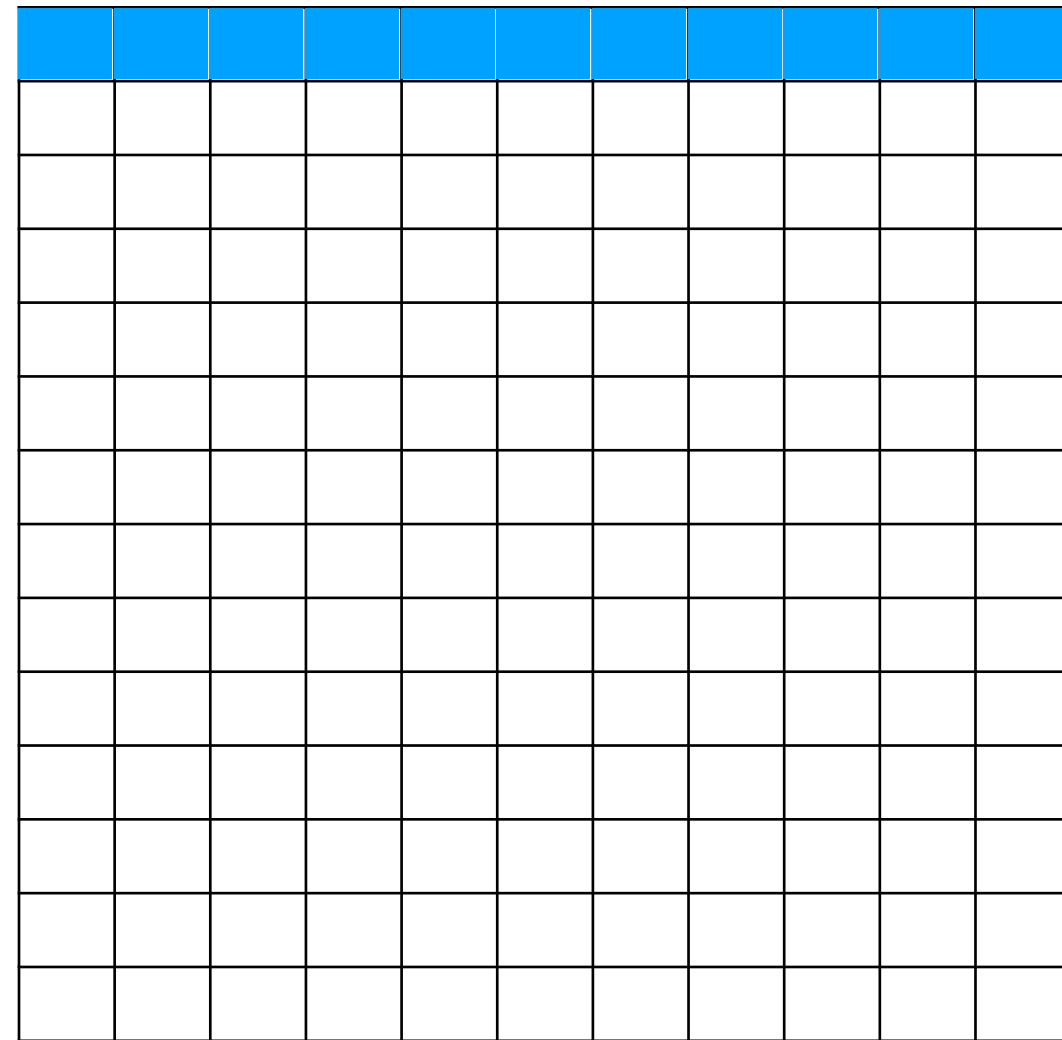
<sup>1</sup>Myers and Miller [1988] later show that using the same ideas, affine gap alignments can be computed in  $O(m)$ -space ( $m < n$ )



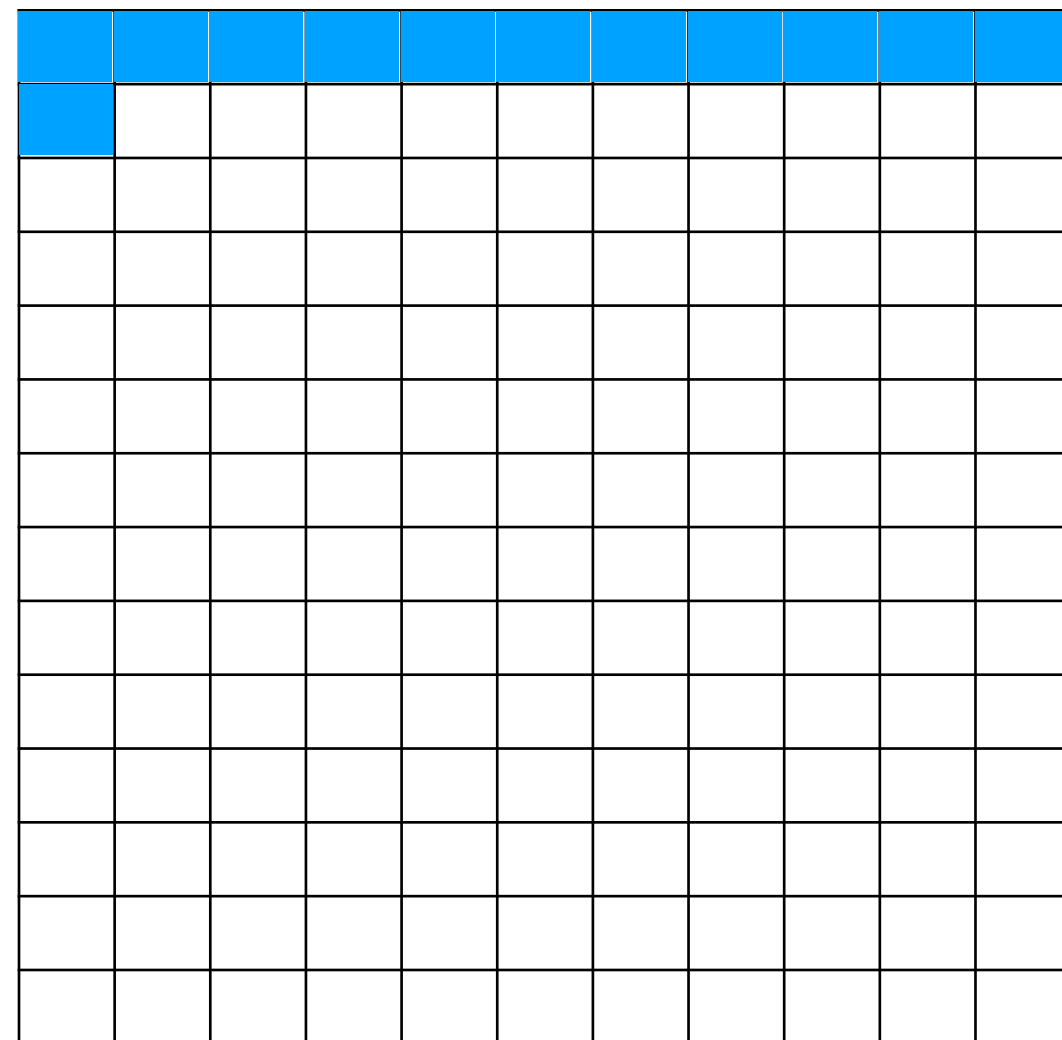
# Hirschberg's Algorithm



# Hirschberg's Algorithm



# Hirschberg's Algorithm

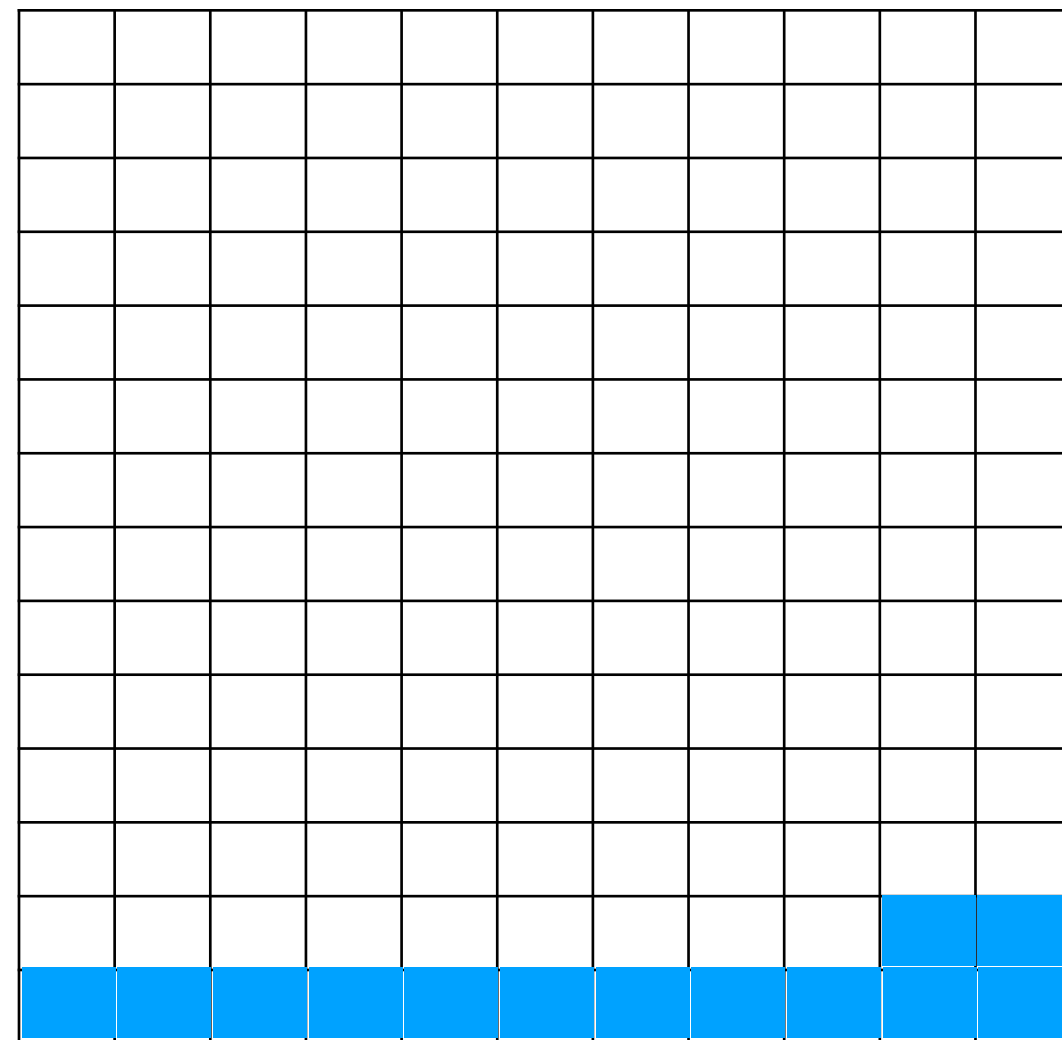




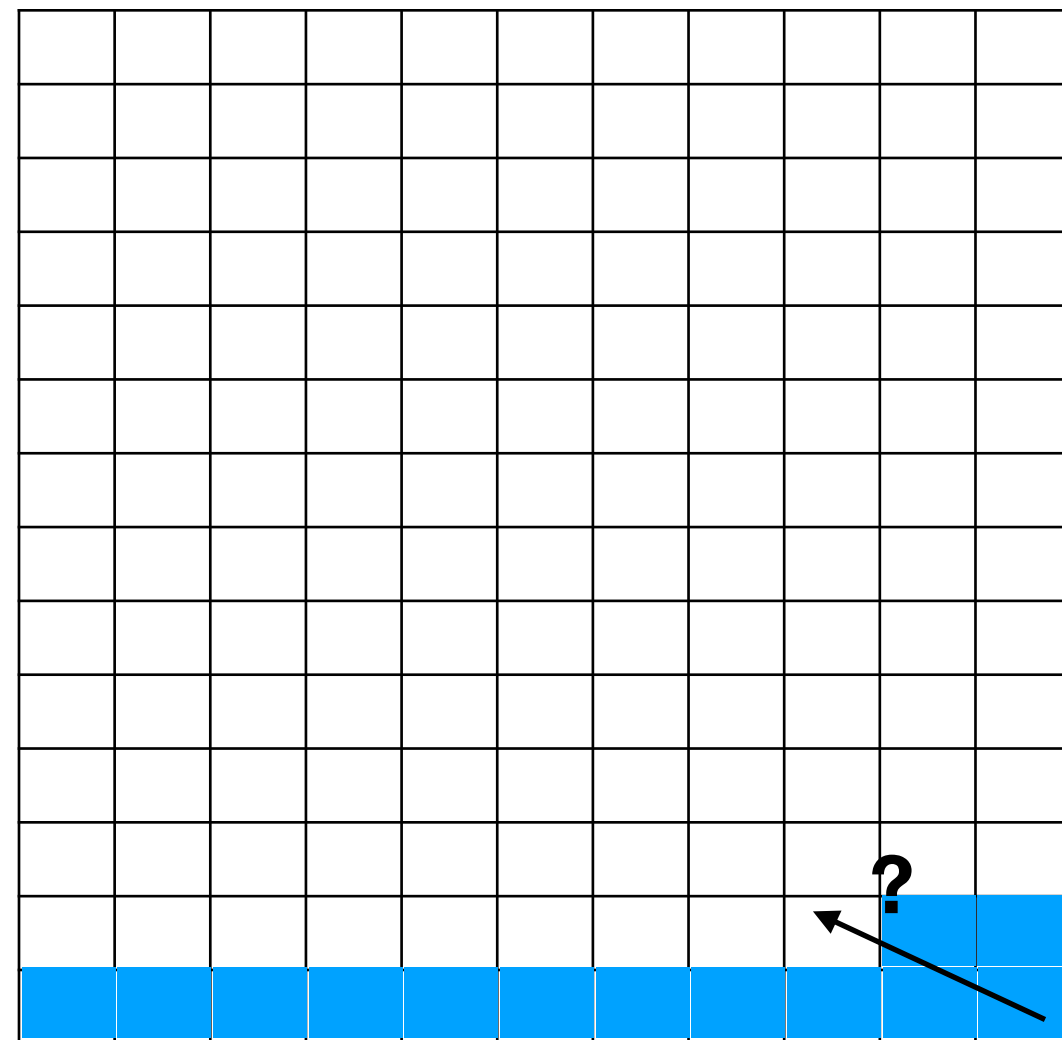




# Hirschberg's Algorithm

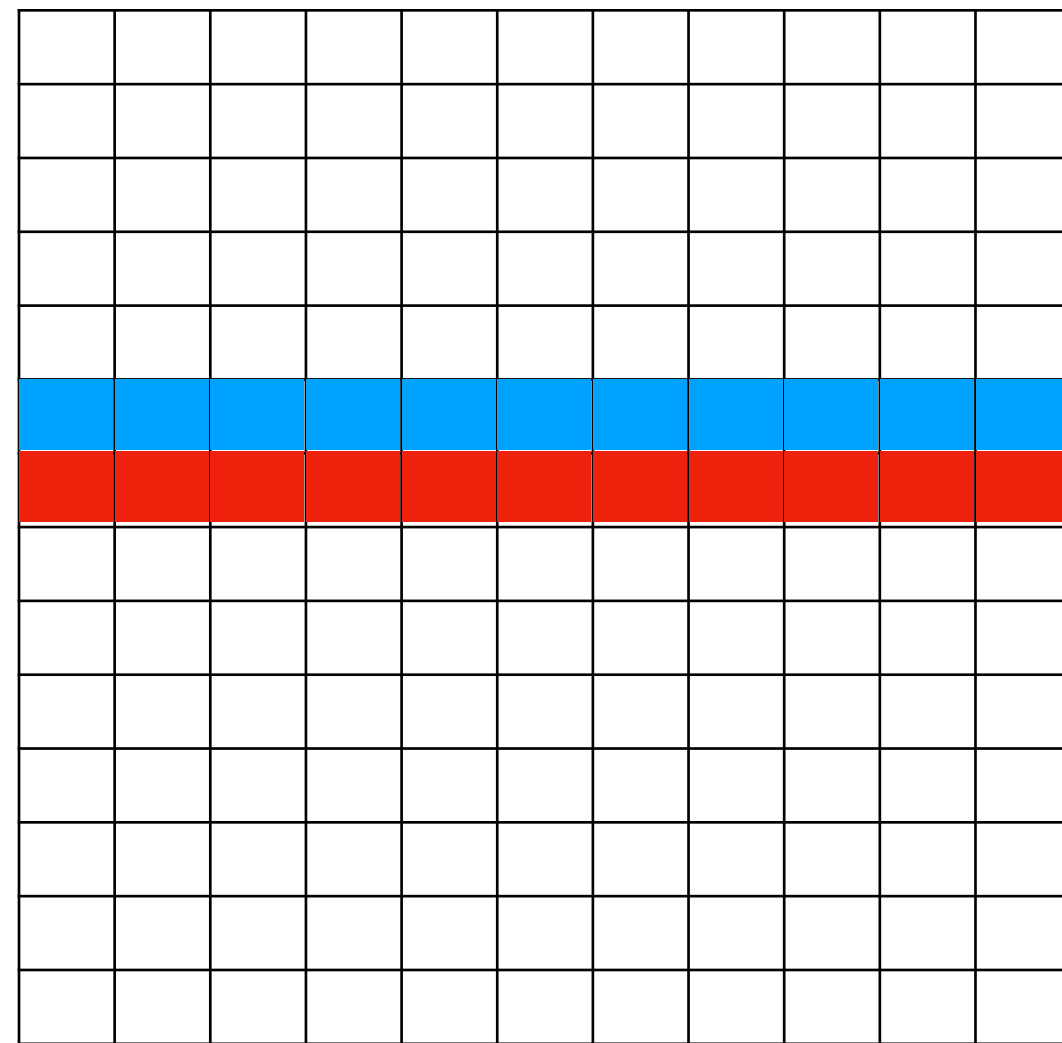


# Hirschberg's Algorithm

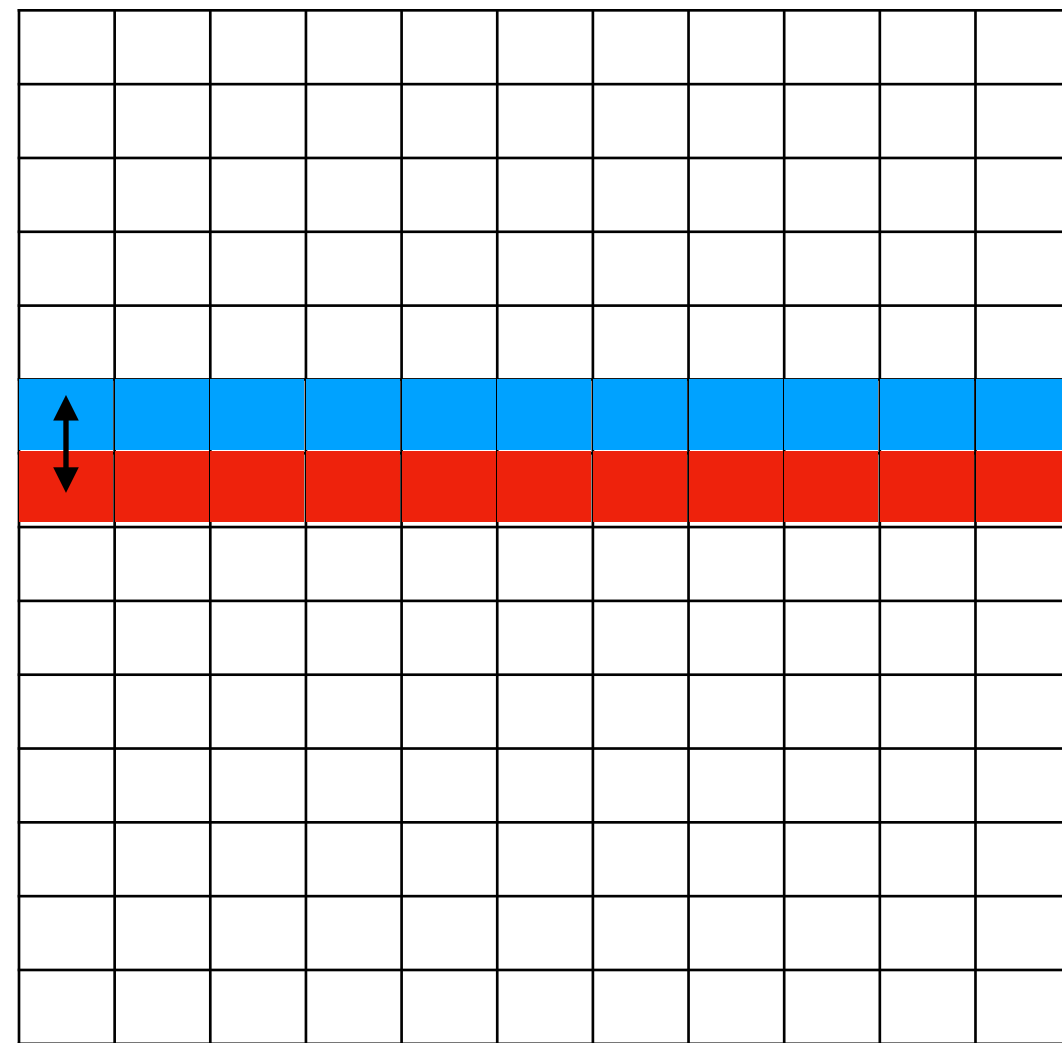




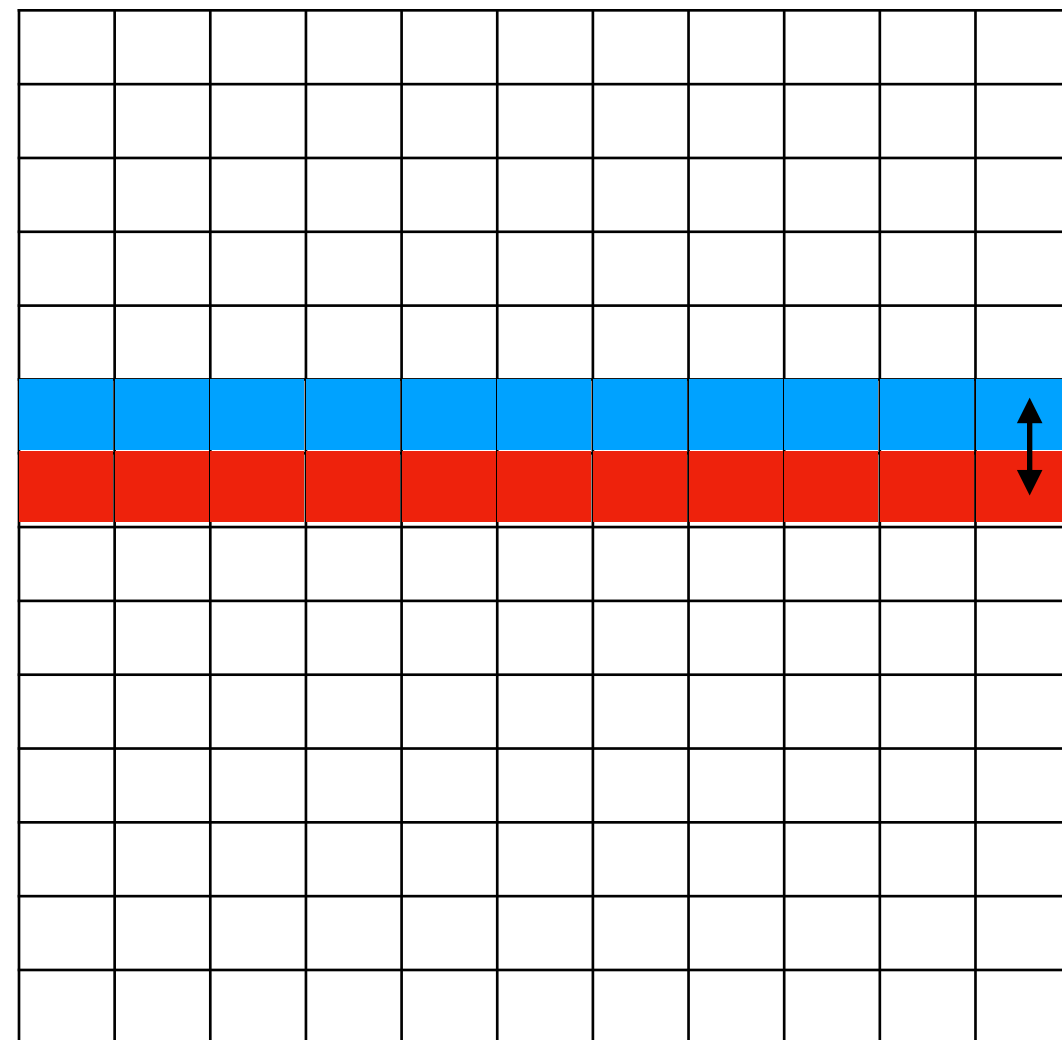
# Hirschberg's Algorithm



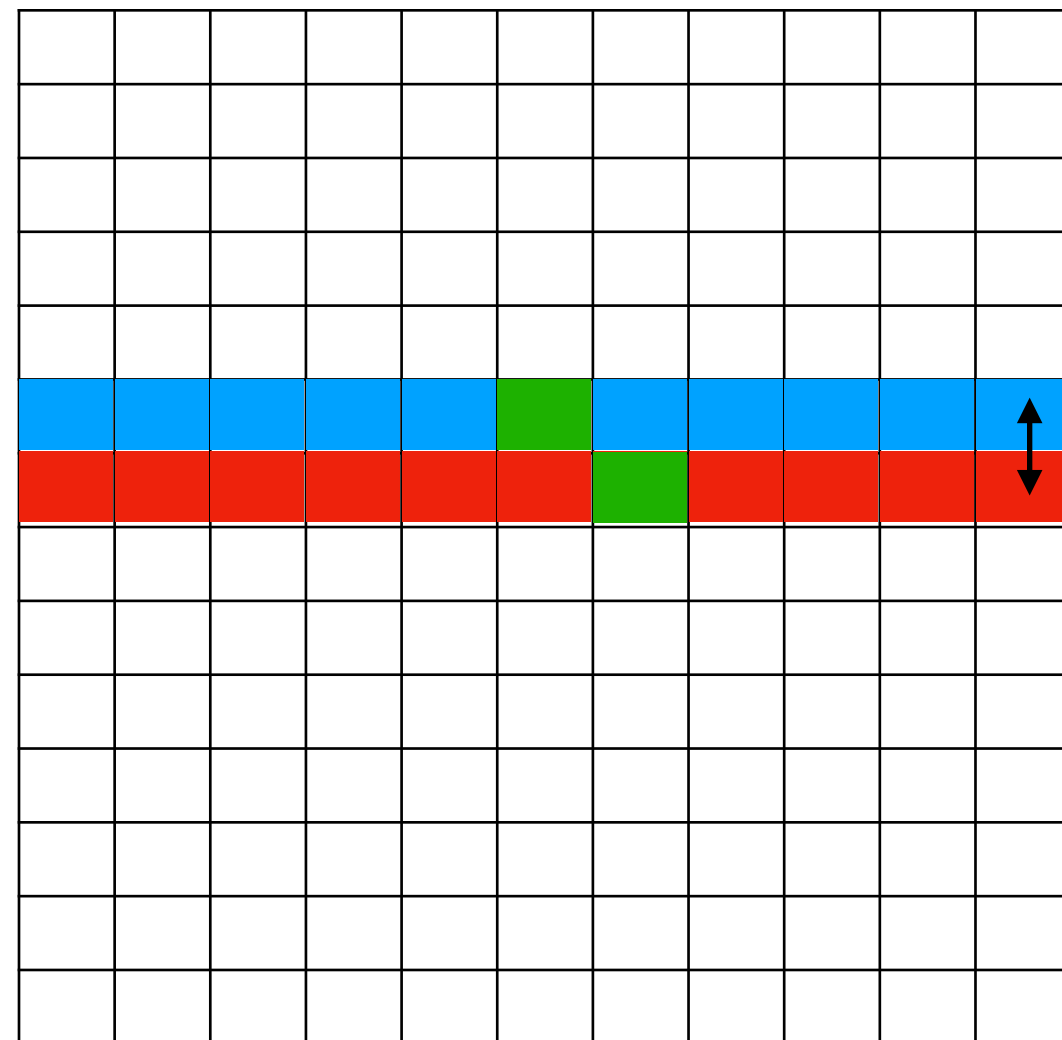
# Hirschberg's Algorithm



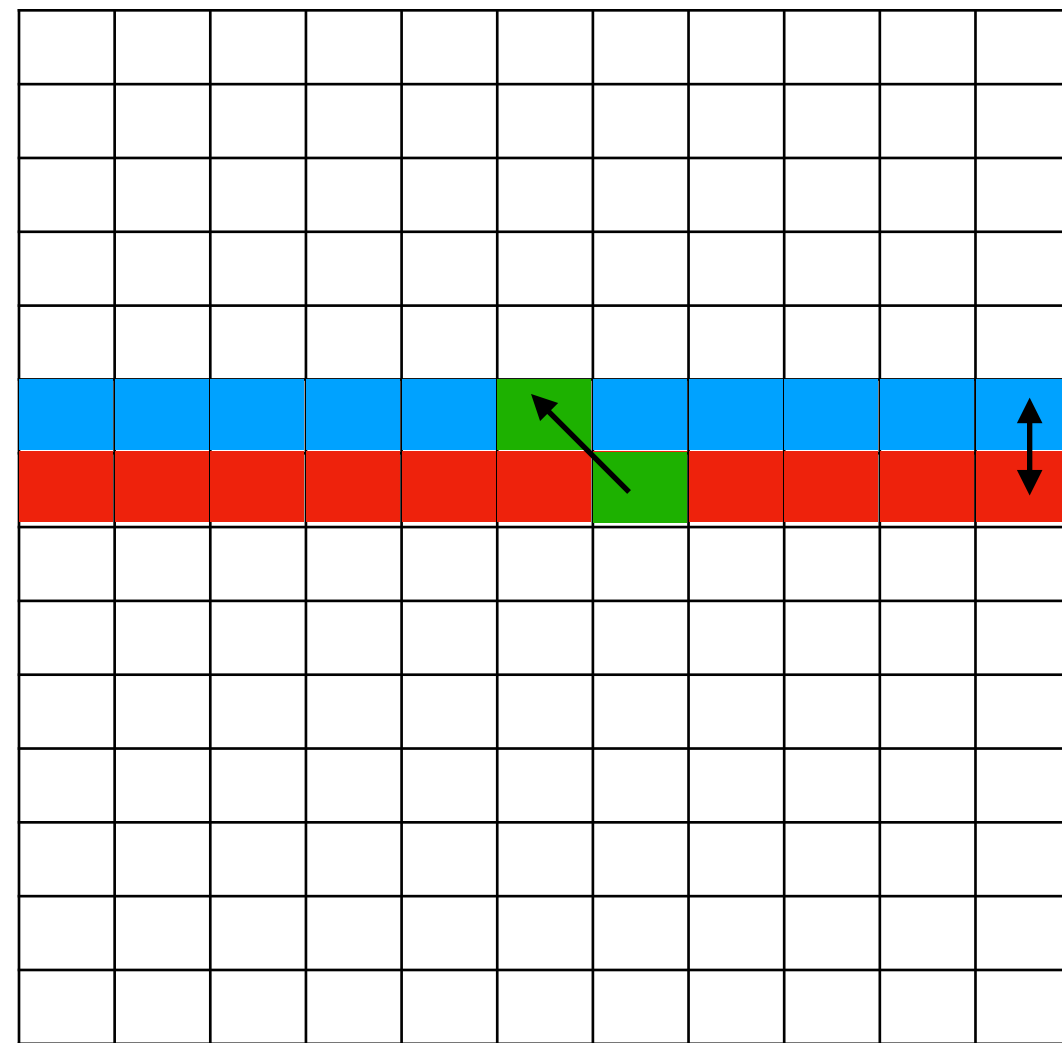
# Hirschberg's Algorithm



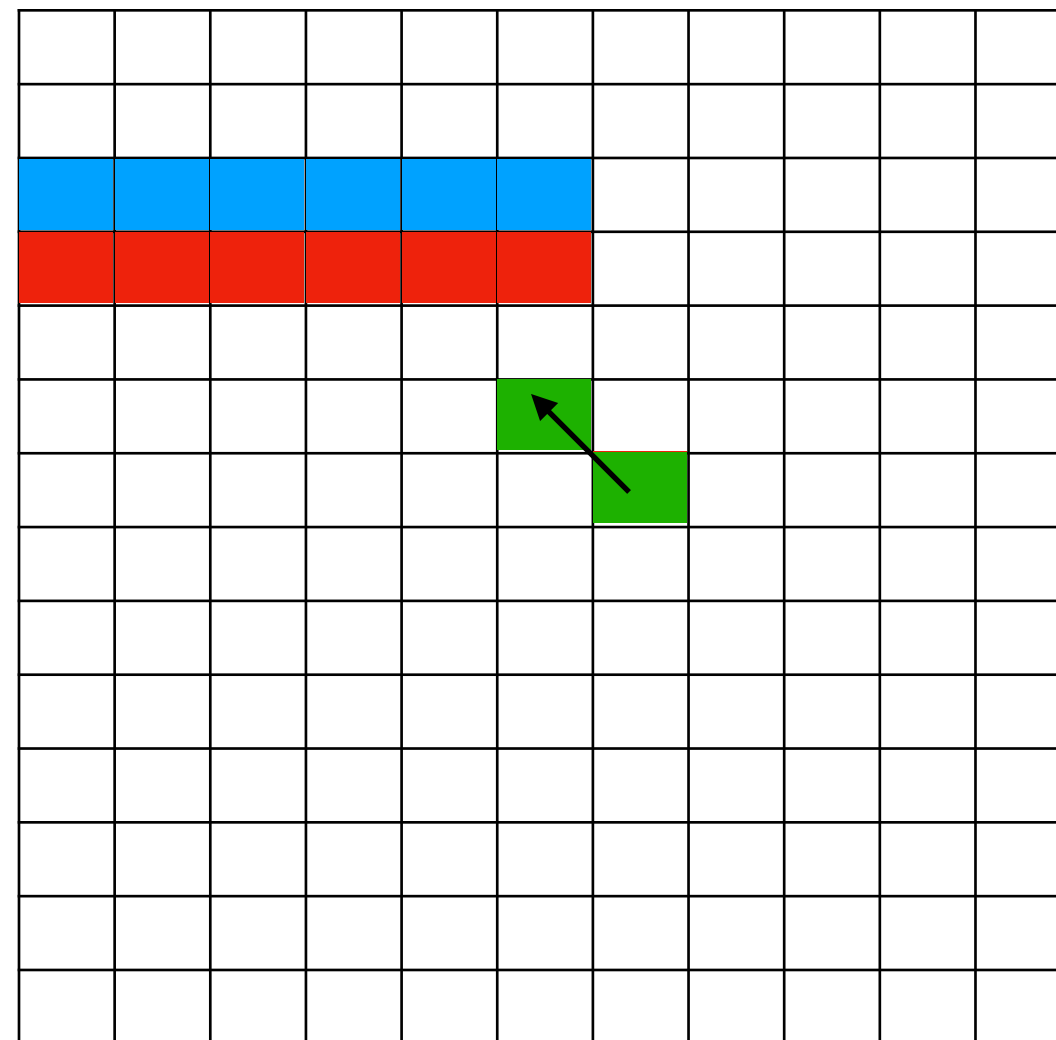
# Hirschberg's Algorithm



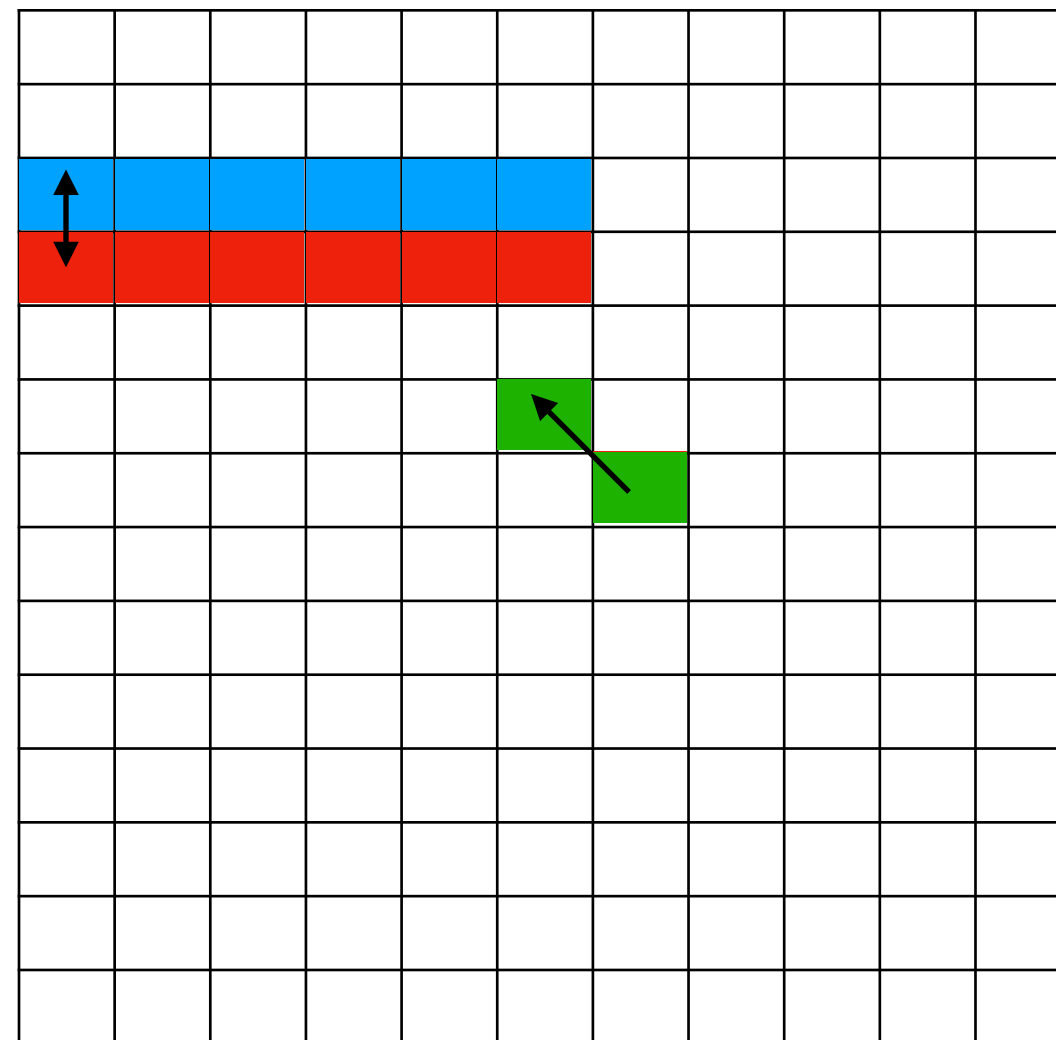
# Hirschberg's Algorithm



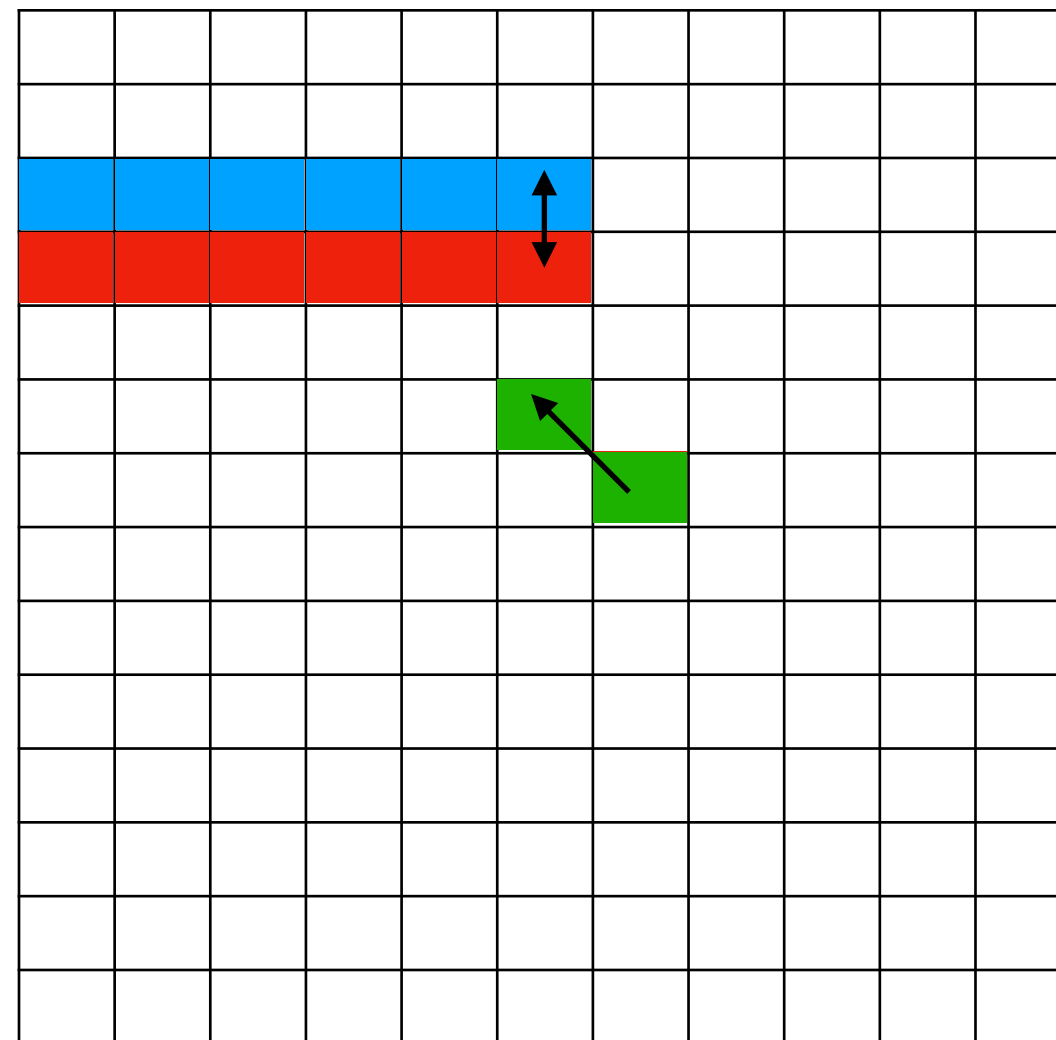
# Hirschberg's Algorithm



# Hirschberg's Algorithm

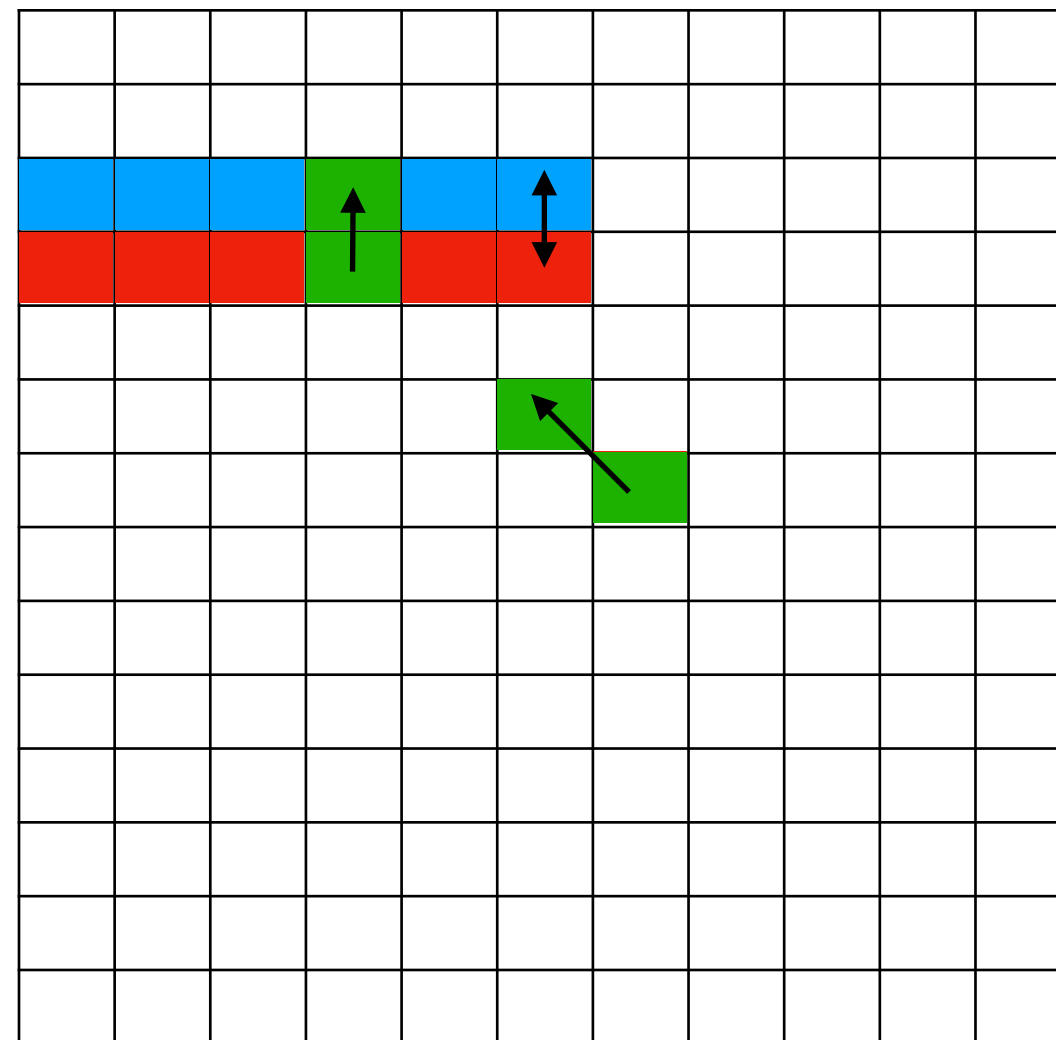


# Hirschberg's Algorithm





# Hirschberg's Algorithm



# Hirschberg's Algorithm

**FindMid**( $S[1\dots n], T[1\dots m]$ ):

$F = \text{CostOnlyNWForward}(S[1\dots n/2], T[1\dots m])$

$B = \text{CostOnlyNWBackward}(S[n/2+1\dots n], T[1\dots m])$

**return**  $\text{argmax}_j \{ F[j] + B[j] \}$

# Hirschberg's Algorithm

**FindMid**( $S[1\dots n], T[1\dots n]$ ):

$F = \text{CostOnlyNWForward}(S[1\dots n/2], T[1\dots m])$

$B = \text{CostOnlyNWBackward}(S[n/2+1\dots n], T[1\dots m])$

**return**  $\text{argmax}_j \{ F[j] + B[j] \}$

**HirschbergAlign**( $S[i\dots j], T[x\dots y]$ ):

**if**  $i=j$  **then** compute full Needleman-Wunsch and **return** the alignment

$mid = (i+j)/2$

$z = \text{FindMid}(S[1\dots n], T[1\dots n])$

**return**  $\text{HirschbergAlign}(S[i\dots mid], T[x\dots z]) \cdot \text{HirschbergAlign}(S[mid+1\dots j], T[z+1\dots y])$

Concatenation



# Hirschberg's Algorithm

**FindMid**( $S[1\dots n], T[1\dots m]$ ):

$F = \text{CostOnlyNWForward}(S[1\dots n/2], T[1\dots m])$

$B = \text{CostOnlyNWBackward}(S[n/2+1\dots n], T[1\dots m])$

**return**  $\text{argmax}_j \{ F[j] + B[j] \}$

$O(m)$ -space, freed after each call

**HirschbergAlign**( $S[i\dots j], T[x\dots y]$ ):

**if**  $i=j$  **then** compute full Needleman-Wunsch and **return** the alignment

$mid = (i+j)/2$

$z = \text{FindMid}(S[1\dots n], T[1\dots m])$

**return** **HirschbergAlign**( $S[i\dots mid], T[x\dots z]$ )  $\cdot$  **HirschbergAlign**( $S[mid+1\dots j], T[z+1\dots y]$ )

Concatenation

# Hirschberg's Algorithm

**FindMid**( $S[1\dots n], T[1\dots m]$ ):

$F = \text{CostOnlyNWForward}(S[1\dots n/2], T[1\dots m])$

$B = \text{CostOnlyNWBackward}(S[n/2+1\dots n], T[1\dots m])$

**return**  $\text{argmax}_j \{ F[j] + B[j] \}$

$O(m)$ -space, freed after each call

**HirschbergAlign**( $S[i\dots j], T[x\dots y]$ ):

**if**  $i=j$  **then** compute full Needleman-Wunsch and **return** the alignment

$mid = (i+j)/2$

$z = \text{FindMid}(S[1\dots n], T[1\dots m])$

**return** **HirschbergAlign**( $S[i\dots mid], T[x\dots z]$ ) **·** **HirschbergAlign**( $S[mid+1\dots j], T[z+1\dots y]$ )

$O(n)$ -space, to store the stack

Concatenation

# Hirschberg's Algorithm

**FindMid**( $S[1\dots n], T[1\dots n]$ ):

$F = \text{CostOnlyNWForward}(S[1\dots n/2], T[1\dots m])$

$B = \text{CostOnlyNWBackward}(S[n/2+1\dots n], T[1\dots m])$

**return**  $\text{argmax}_j \{ F[j] + B[j] \}$

**HirschbergAlign**( $S[i\dots j], T[x\dots y]$ ):

**if**  $i=j$  **then** compute full Needleman-Wunsch and **return** the alignment

$mid = (i+j)/2$

$z = \text{FindMid}(S[1\dots n], T[1\dots n])$

**return**  $\text{HirschbergAlign}(S[i\dots mid], T[x\dots z]) \cdot \text{HirschbergAlign}(S[mid+1\dots j], T[z+1\dots y])$

Concatenation



# Hirschberg's Algorithm

**FindMid**( $S[1\dots n], T[1\dots n]$ ):

$F = \text{CostOnlyNWForward}(S[1\dots n/2], T[1\dots m])$

$B = \text{CostOnlyNWBackward}(S[n/2+1\dots n], T[1\dots m])$

**return**  $\text{argmax}_j \{ F[j] + B[j] \}$

$O(mn)$ -time

**HirschbergAlign**( $S[i\dots j], T[x\dots y]$ ):

**if**  $i=j$  **then** compute full Needleman-Wunsch and **return** the alignment

$mid = (i+j)/2$

$z = \text{FindMid}(S[1\dots n], T[1\dots n])$

**return**  $\text{HirschbergAlign}(S[i\dots mid], T[x\dots z]) \cdot \text{HirschbergAlign}(S[mid+1\dots j], T[z+1\dots y])$

Concatenation



# Hirschberg's Algorithm

**FindMid**( $S[1\dots n], T[1\dots n]$ ):

$F = \text{CostOnlyNWForward}(S[1\dots n/2], T[1\dots m])$

$O(mn)$ -time

$B = \text{CostOnlyNWBackward}(S[n/2+1\dots n], T[1\dots m])$

$O(mn)$ -time

**return**  $\text{argmax}_j \{ F[j] + B[j] \}$

**HirschbergAlign**( $S[i\dots j], T[x\dots y]$ ):

**if**  $i=j$  **then** compute full Needleman-Wunsch and **return** the alignment

$mid = (i+j)/2$

$z = \text{FindMid}(S[1\dots n], T[1\dots n])$

**return** **HirschbergAlign**( $S[i\dots mid], T[x\dots z]$ ) **·** **HirschbergAlign**( $S[mid+1\dots j], T[z+1\dots y]$ )

Concatenation





# Hirschberg's Algorithm

**FindMid**( $S[1\dots n], T[1\dots n]$ ):

$F = \text{CostOnlyNWForward}(S[1\dots n/2], T[1\dots m])$

$B = \text{CostOnlyNWBackward}(S[n/2+1\dots n], T[1\dots m])$

**return**  $\text{argmax}_j \{ F[j] + B[j] \}$

$O(mn)$ -time

$O(mn)$ -time

$O(1)$ -time

**HirschbergAlign**( $S[i\dots j], T[x\dots y]$ ):

**if**  $i=j$  **then** compute full Needleman-Wunsch and **return** the alignment

$mid = (i+j)/2$

$z = \text{FindMid}(S[1\dots n], T[1\dots n])$

**return**  $\text{HirschbergAlign}(S[i\dots mid], T[x\dots z]) \cdot \text{HirschbergAlign}(S[mid+1\dots j], T[z+1\dots y])$

**Concatenation**



# Hirschberg's Algorithm

**FindMid**( $S[1\dots n], T[1\dots m]$ ):

$F = \text{CostOnlyNWForward}(S[1\dots n/2], T[1\dots m])$

$B = \text{CostOnlyNWBackward}(S[n/2+1\dots n], T[1\dots m])$

**return**  $\text{argmax}_j \{ F[j] + B[j] \}$

$O(mn)$ -time

$O(mn)$ -time

$O(1)$ -time

$O(mn)$ -time for each call

**HirschbergAlign**( $S[i\dots j], T[x\dots y]$ ):

**if**  $i=j$  **then** compute full Needleman-Wunsch and **return** the alignment

$mid = (i+j)/2$

$z = \text{FindMid}(S[1\dots n], T[1\dots m])$

**return** **HirschbergAlign**( $S[i\dots mid], T[x\dots z]$ )  $\cdot$  **HirschbergAlign**( $S[mid+1\dots j], T[z+1\dots y]$ )

Concatenation

# Hirschberg's Algorithm

**FindMid**( $S[1\dots n], T[1\dots n]$ ):

$F = \text{CostOnlyNWForward}(S[1\dots n/2], T[1\dots m])$

$B = \text{CostOnlyNWBackward}(S[n/2+1\dots n], T[1\dots m])$

**return**  $\text{argmax}_j \{ F[j] + B[j] \}$

$O(mn)$ -time

$O(mn)$ -time

$O(1)$ -time

$O(mn)$ -time for each call

**HirschbergAlign**( $S[i\dots j], T[x\dots y]$ ):

$\text{Time}(n, m)$

**if**  $i=j$  **then** compute full Needleman-Wunsch and **return** the alignment

$mid = (i+j)/2$

$z = \text{FindMid}(S[1\dots n], T[1\dots n])$

**return**  $\text{HirschbergAlign}(S[i\dots mid], T[x\dots z]) \cdot \text{HirschbergAlign}(S[mid+1\dots j], T[z+1\dots y])$

Concatenation

# Hirschberg's Algorithm

**FindMid**( $S[1\dots n], T[1\dots n]$ ):

$F = \text{CostOnlyNWForward}(S[1\dots n/2], T[1\dots m])$

$B = \text{CostOnlyNWBackward}(S[n/2+1\dots n], T[1\dots m])$

**return**  $\text{argmax}_j \{ F[j] + B[j] \}$

$O(mn)$ -time

$O(mn)$ -time

$O(1)$ -time

$O(mn)$ -time for each call

**HirschbergAlign**( $S[i\dots j], T[x\dots y]$ ):

**if**  $i=j$  **then** compute full Needleman-Wunsch and **return** the alignment

$mid = (i+j)/2$

$z = \text{FindMid}(S[1\dots n], T[1\dots n])$

**return**  $\text{HirschbergAlign}(S[i\dots mid], T[x\dots z]) \cdot \text{HirschbergAlign}(S[mid+1\dots j], T[z+1\dots y])$

$\text{Time}(n, m)$

$O(mn)$ -time

Concatenation

# Hirschberg's Algorithm

**FindMid**( $S[1\dots n], T[1\dots m]$ ):

$F = \text{CostOnlyNWForward}(S[1\dots n/2], T[1\dots m])$

$B = \text{CostOnlyNWBackward}(S[n/2+1\dots n], T[1\dots m])$

**return**  $\text{argmax}_j \{ F[j] + B[j] \}$

$O(mn)$ -time

$O(mn)$ -time

$O(1)$ -time

$O(mn)$ -time for each call

**HirschbergAlign**( $S[i\dots j], T[x\dots y]$ ):

**if**  $i=j$  **then** compute full Needleman-Wunsch and **return** the alignment

$mid = (i+j)/2$

$z = \text{FindMid}(S[1\dots n], T[1\dots m])$

**return**  $\text{HirschbergAlign}(S[i\dots mid], T[x\dots z]) \cdot \text{HirschbergAlign}(S[mid+1\dots j], T[z+1\dots y])$

$\text{Time}(n, m)$

$O(mn)$ -time

$\text{Time}(n/2, z) + \text{Time}(n/2, m-z)$

**Concatenation**

# Hirschberg's Algorithm

**FindMid**( $S[1\dots n], T[1\dots m]$ ):

$F = \text{CostOnlyNWForward}(S[1\dots n/2], T[1\dots m])$

$B = \text{CostOnlyNWBackward}(S[n/2+1\dots n], T[1\dots m])$

**return**  $\text{argmax}_j \{ F[j] + B[j] \}$

$O(mn)$ -time

$O(mn)$ -time

$O(1)$ -time

$O(mn)$ -time for each call

**HirschbergAlign**( $S[i\dots j], T[x\dots y]$ ):

**if**  $i=j$  **then** compute full Needleman-Wunsch and **return** the alignment

$mid = (i+j)/2$

$z = \text{FindMid}(S[1\dots n], T[1\dots m])$

**return**  $\text{HirschbergAlign}(S[i\dots mid], T[x\dots z]) \cdot \text{HirschbergAlign}(S[mid+1\dots j], T[z+1\dots y])$

$\text{Time}(n, m)$

$O(mn)$ -time

$O(mn)$ -time total

$\text{Time}(n/2, z) + \text{Time}(n/2, m-z)$

**Concatenation**

# Alignment Scores

- In all of the examples we have been given  $\delta$ ,  $a$ ,  $b$ , etc. and using fixed values we are able to find the optimal alignment in  $O(mn)$ -time
- Since different values of those parameter induce different alignments, how do we know which parameter values are best?
- This is a problem known as *inverse parametric alignment* (or just parametric alignment) where you want to find ***all*** possible alignments of the two sequences.
- How many *optimal* alignments do you think there are? (notice that optimal is emphasized)