CS 364 COMPUTATIONAL BIOLOGY

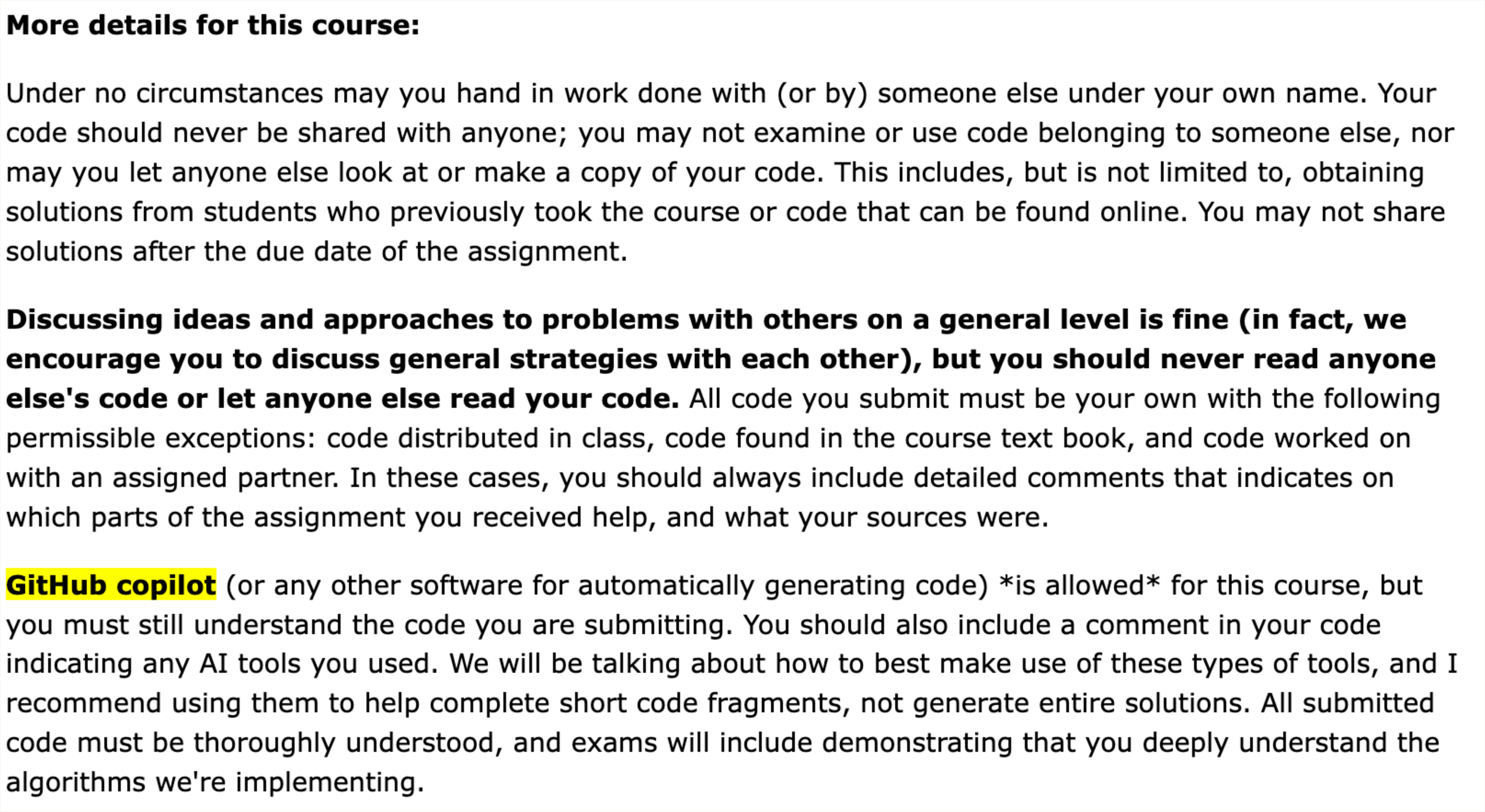
Sara Mathieson
Haverford College

Feedback and schedule notes

- More time for labs, midnight deadline
- More board work, more practice problems
- Random partners, mixed feelings
- Exam: Thursday Oct 10 (in-class)
- More office hours (may not have another official time but feel free to make appointments!)

Academic Integrity notes

From the syllabus:



More details for this course:

Under no circumstances may you hand in work done with (or by) someone else under your own name. Your code should never be shared with anyone; you may not examine or use code belonging to someone else, nor may you let anyone else look at or make a copy of your code. This includes, but is not limited to, obtaining solutions from students who previously took the course or code that can be found online. You may not share solutions after the due date of the assignment.

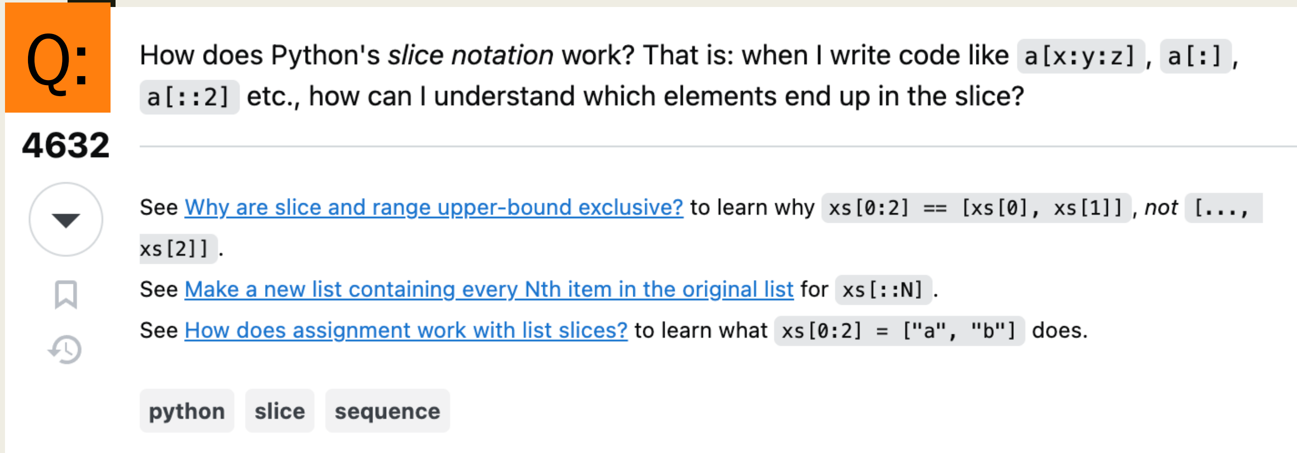
Discussing ideas and approaches to problems with others on a general level is fine (in fact, we encourage you to discuss general strategies with each other), but you should never read anyone else's code or let anyone else read your code. All code you submit must be your own with the following permissible exceptions: code distributed in class, code found in the course text book, and code worked on with an assigned partner. In these cases, you should always include detailed comments that indicates on which parts of the assignment you received help, and what your sources were.

GitHub copilot (or any other software for automatically generating code) *is allowed* for this course, but you must still understand the code you are submitting. You should also include a comment in your code indicating any AI tools you used. We will be talking about how to best make use of these types of tools, and I recommend using them to help complete short code fragments, not generate entire solutions. All submitted code must be thoroughly understood, and exams will include demonstrating that you deeply understand the algorithms we're implementing.

Academic Integrity notes

Example of citing stack overflow:

Q:



4632

How does Python's *slice notation* work? That is: when I write code like `a[x:y:z]`, `a[:]`, `a[::2]` etc., how can I understand which elements end up in the slice?

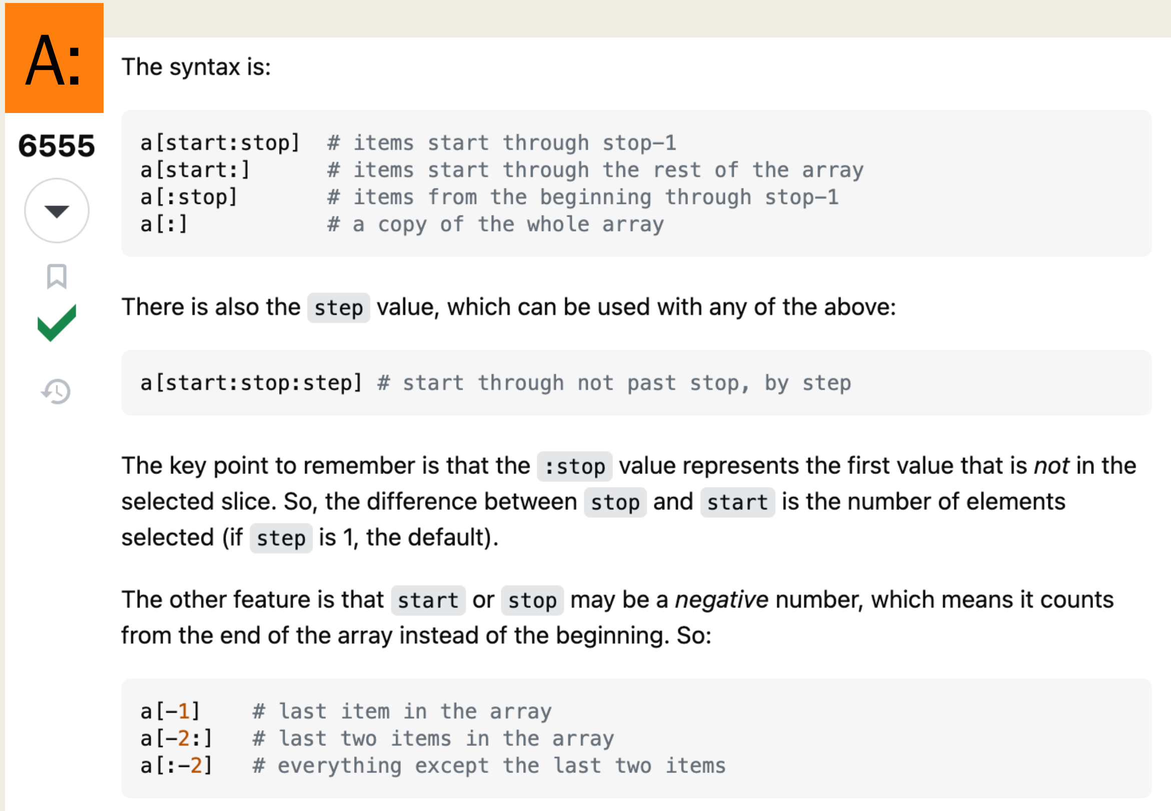
See Why are slice and range upper-bound exclusive? to learn why `xs[0:2] == [xs[0], xs[1]]`, *not* `[..., xs[2]]`.

See Make a new list containing every Nth item in the original list for `xs[::N]`.

See How does assignment work with list slices? to learn what `xs[0:2] = ["a", "b"]` does.

python slice sequence

A:



6555

The syntax is:

```
a[start:stop]  # items start through stop-1
a[start:]      # items start through the rest of the array
a[:stop]       # items from the beginning through stop-1
a[:]           # a copy of the whole array
```

There is also the `step` value, which can be used with any of the above:

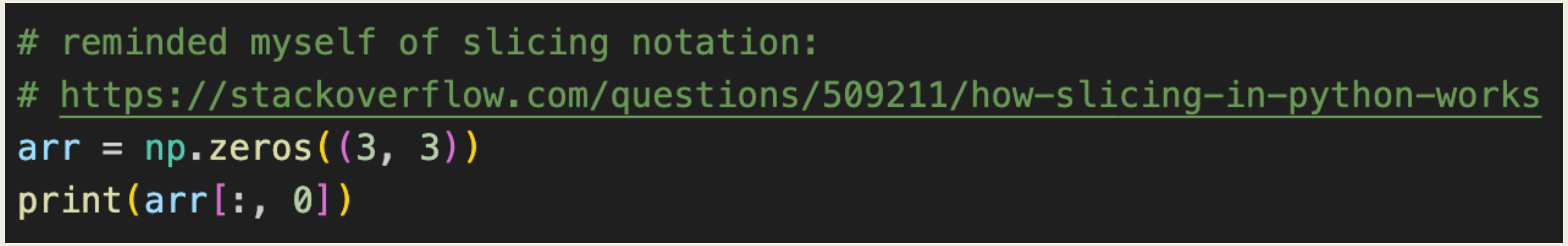
```
a[start:stop:step] # start through not past stop, by step
```

The key point to remember is that the `:stop` value represents the first value that is *not* in the selected slice. So, the difference between `stop` and `start` is the number of elements selected (if `step` is 1, the default).

The other feature is that `start` or `stop` may be a *negative* number, which means it counts from the end of the array instead of the beginning. So:

```
a[-1]    # last item in the array
a[-2:]   # last two items in the array
a[:-2]   # everything except the last two items
```

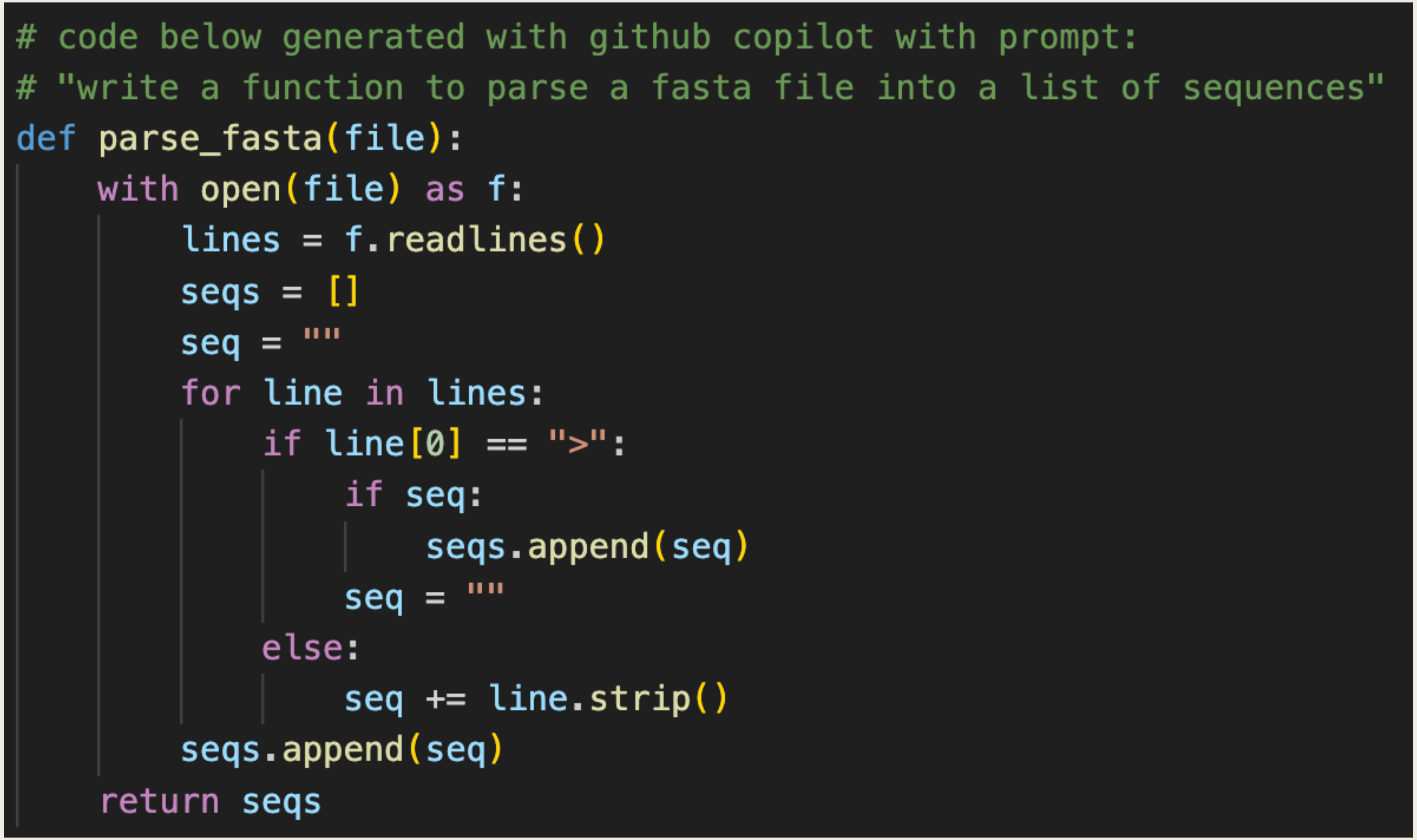
In your code:



```
# reminded myself of slicing notation:
# https://stackoverflow.com/questions/509211/how-slicing-in-python-works
arr = np.zeros((3, 3))
print(arr[:, 0])
```

Academic Integrity notes

Example of citing github copilot:



```python
# code below generated with github copilot with prompt:
# "write a function to parse a fasta file into a list of sequences"
def parse_fasta(file):
    with open(file) as f:
        lines = f.readlines()
        seqs = []
        seq = ""
        for line in lines:
            if line[0] == ">":
                if seq:
                    seqs.append(seq)
                seq = ""
            else:
                seq += line.strip()
        seqs.append(seq)
    return seqs
```

Outline

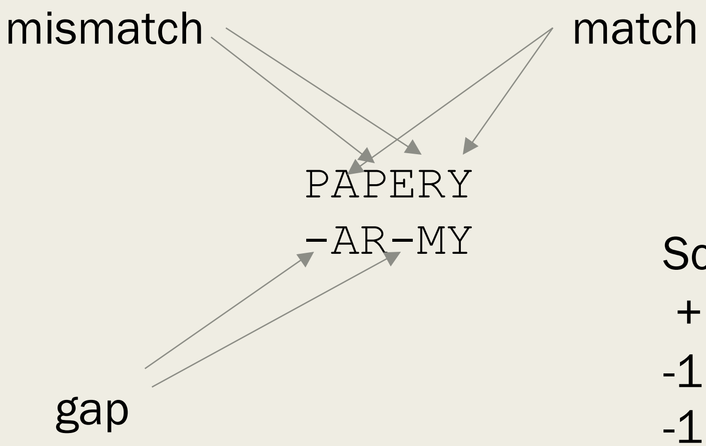
- Global sequence alignment (Needleman-Wunsch)
- Local sequence alignment (Smith-Waterman)
- Alignment variations

Reading: Durbin 2.1-2.3
(*on hold in the library*)

Global Sequence Alignment

Concept: Alignment score

How good is a particular alignment?



mismatch

match

```
PAPERY
-AR-MY
```

gap

Score:

+1 for a match

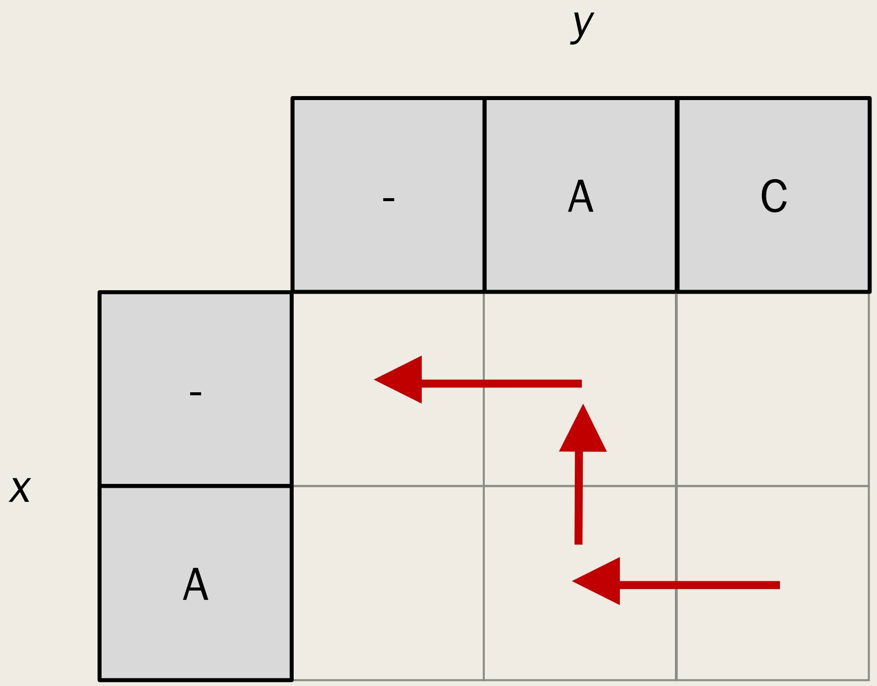
-1 for a mismatch

-1 for a gap

Score = -1 +1 -1 -1 -1 +1 = -2

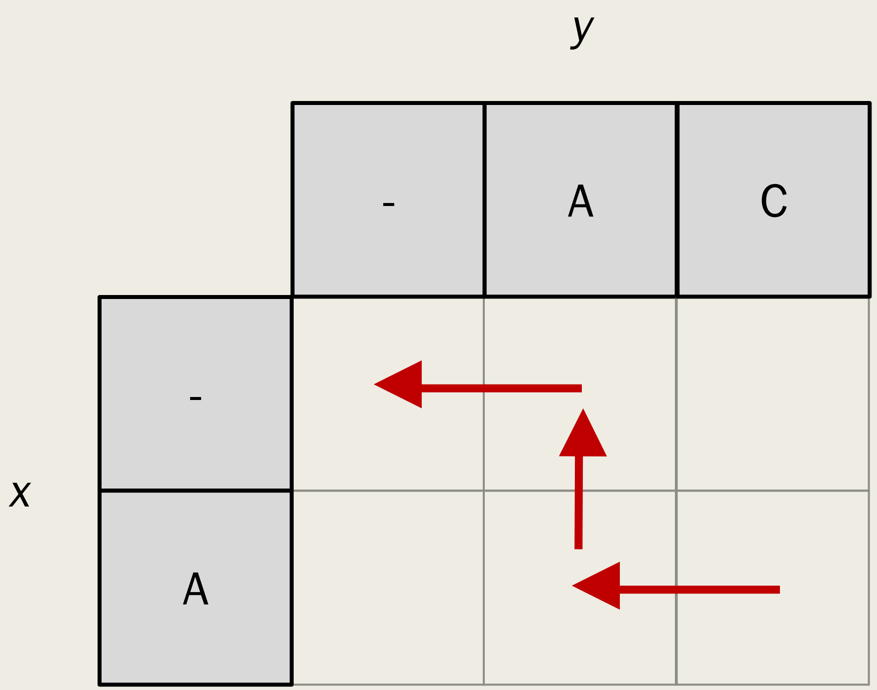
Handout 7, first question

Extra exercise (discuss with a partner)



What alignment does the given trace-back represent?

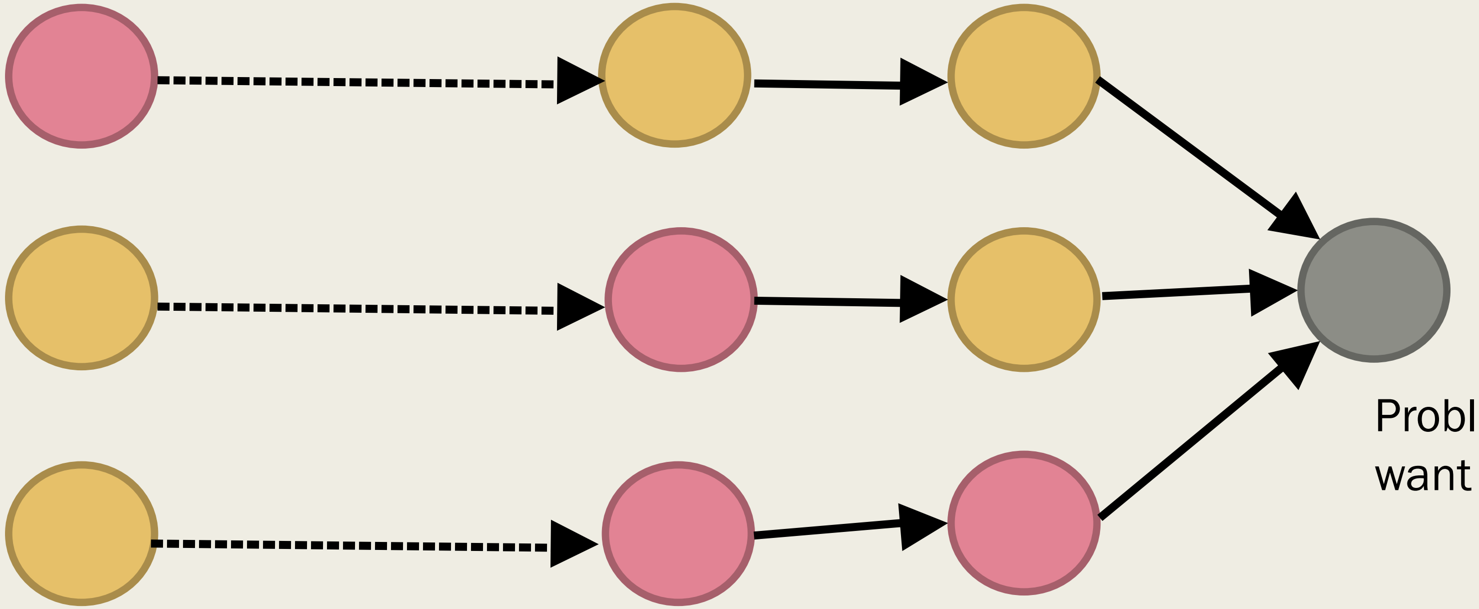
Extra exercise (discuss with a partner)



What alignment does the given trace-back represent?

```
x: -A-
y: A-C
```

Dynamic programming



Problem we
want to solve

The optimal solution to problem n+1 can be expressed in terms of the optimal solution to problem n.

Lots of the smaller problems are **actually the same problem** so as long as we **remember the solution**, we only have to solve them once.

Global alignment (Needleman-Wunsch)

- *S(i,j)* = best alignment score for *x[1...i]* and *y[1...j]* (inclusive), with gap penalty *g* (usually negative) and matching table *m*

- Base case:

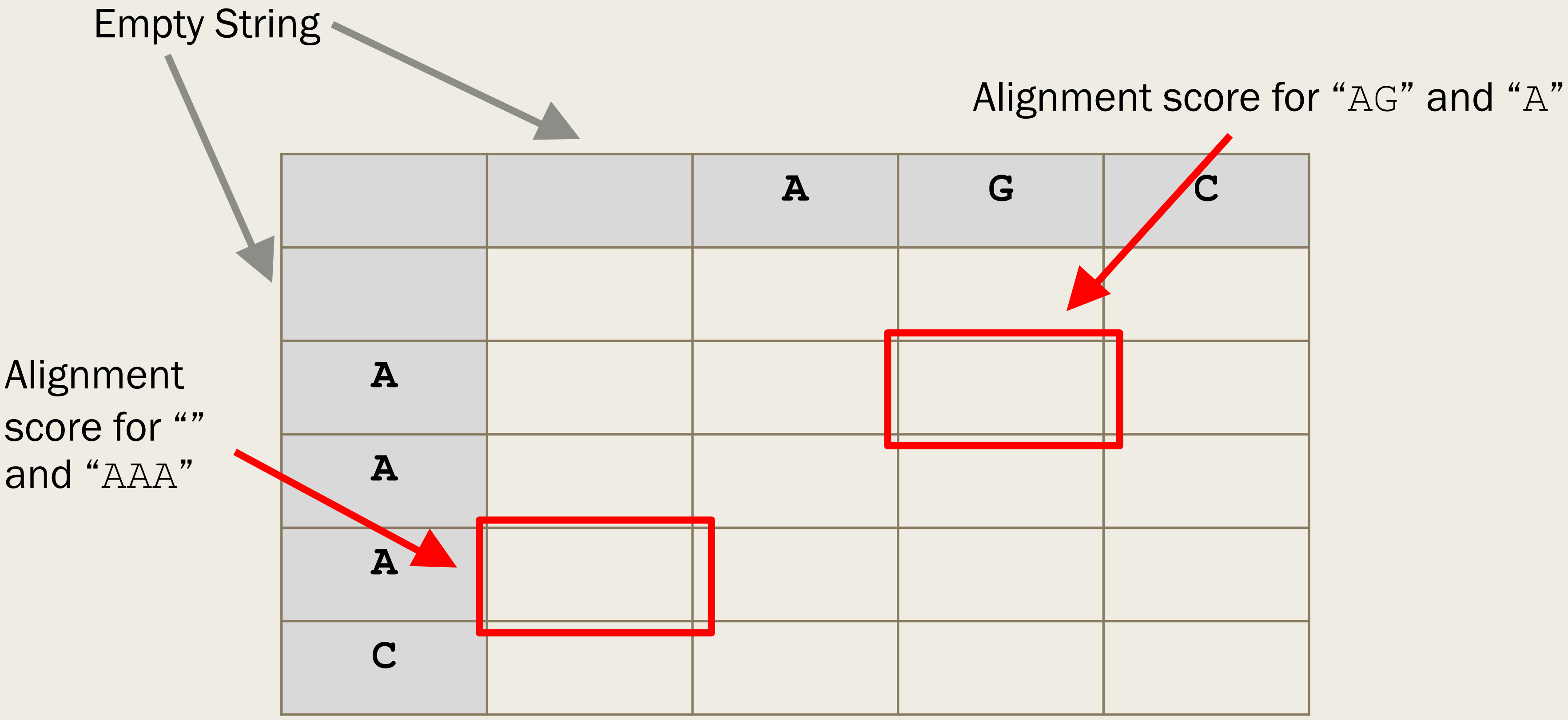
$$\begin{aligned} S(i,0) &= g \cdot i \\ S(0,j) &= g \cdot j \end{aligned}$$

- Recursion:

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

Dynamic programing for alignment

Align the strings AAAC and AGC



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

Empty String

Alignment score for “AG” and “A”

Alignment score for “” and “AAA”

Dynamic programing for alignment

Align the strings AAAC and AGC

e.g. this is the cost to align them empty string “” and “AG”

		A	**G**	**C**
	0	−2	−4	−6
A	−2			
A	−4			
A	−6			
C	−8			

Initialization step

Dynamic programing for alignment

Align the strings `AAAC` and `AGC`

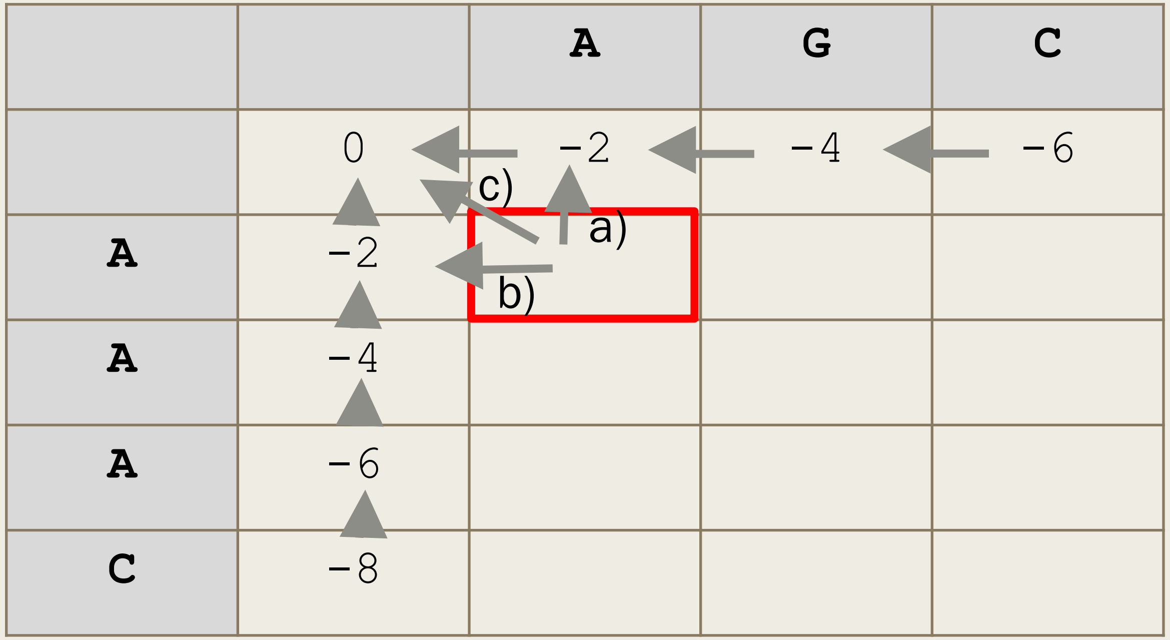
		A	**G**	**C**
	0	← -2	← -4	← -6
A	↑ -2			
A	↑ -4			
A	↑ -6			
C	↑ -8			

Initialization step: traceback

Dynamic programing for alignment

Align the strings AAAC and AGC

a) Add a gap to y; -2-2 = -4
b) Add a gap to x; -2 -2 = -4
c) Extend alignment; 0+1=+1



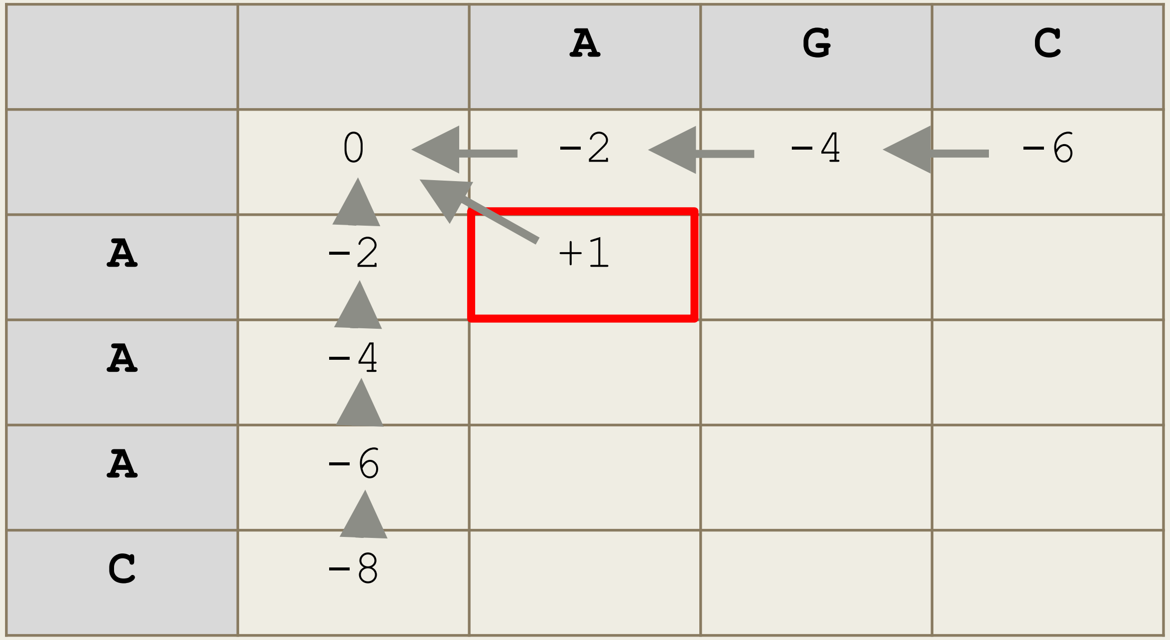
		A	**G**	**C**
	0	-2	-4	-6
A	-2			
A	-4			
A	-6			
C	-8			

Recursive step

Dynamic programing for alignment

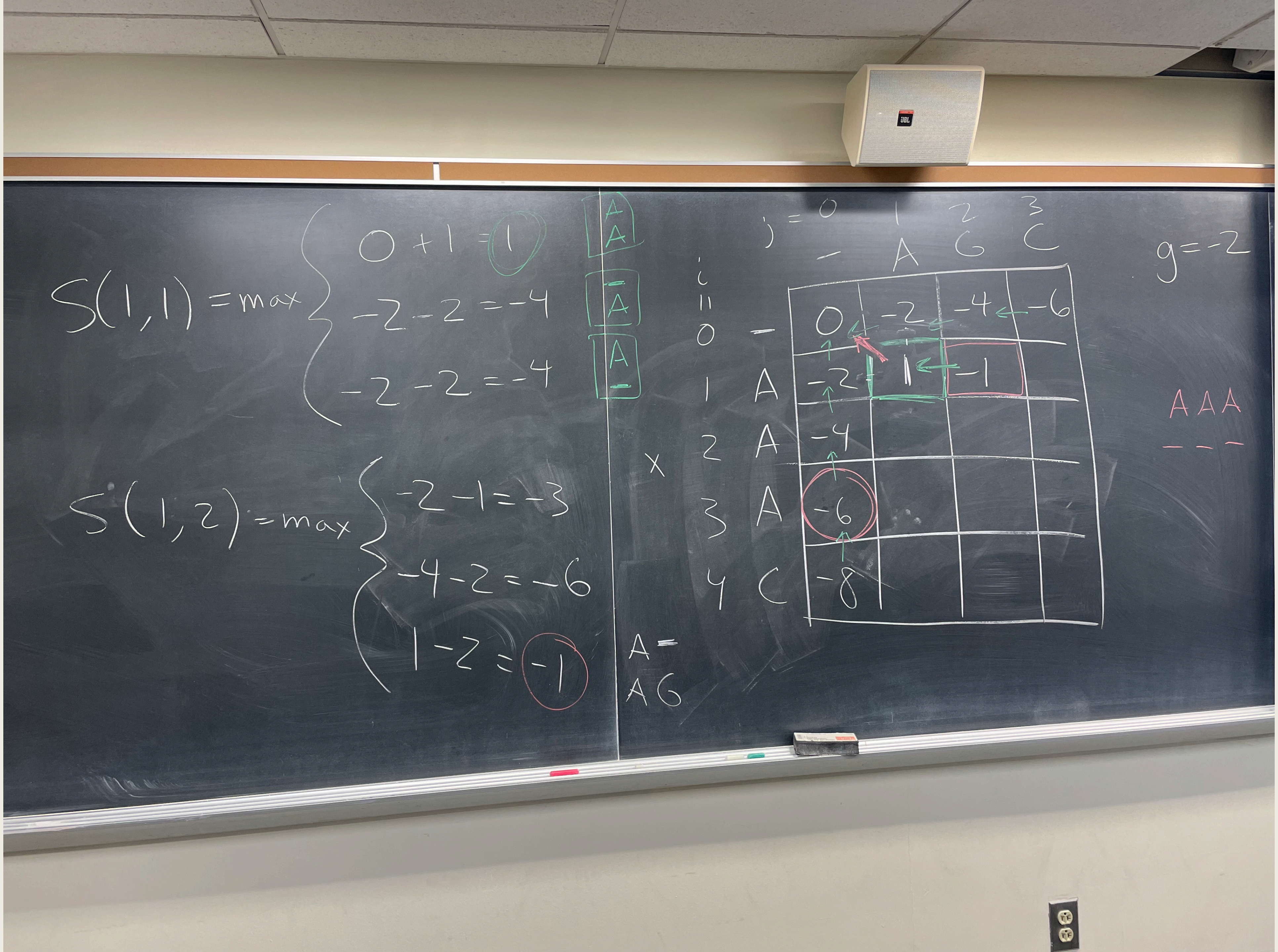
Align the strings `AAAC` and `AGC`

a) Add a gap to y; -2-2 = -4
b) Add a gap to x; -2 -2 = -4
c) Extend alignment; 0+1=+1



		A	**G**	**C**
	0	-2	-4	-6
A	-2	+1		
A	-4			
A	-6			
C	-8			

Recursive step



$$S(1,1) = \max \begin{cases} 0+1 = \boxed{1} \\ -2-2 = -4 \\ -2-2 = -4 \end{cases}$$

$\boxed{\begin{matrix} A \\ A \end{matrix}}$ $\boxed{\begin{matrix} - \\ A \end{matrix}}$ $\boxed{\begin{matrix} A \\ - \end{matrix}}$

$$S(1,2) = \max \begin{cases} -2-1 = -3 \\ -4-2 = -6 \\ 1-2 = \boxed{-1} \end{cases}$$

A-
AG

$$\begin{array}{c|c|cccc} & j= & 0 & 1 & 2 & 3 \\ i= & & - & A & G & C \\ \hline 0 & - & 0 & -2 & -4 & -6 \\ 1 & A & -2 & 1 & -1 & \\ 2 & A & -4 & & & \\ 3 & A & -6 & & & \\ 4 & C & -8 & & & \end{array}$$

x

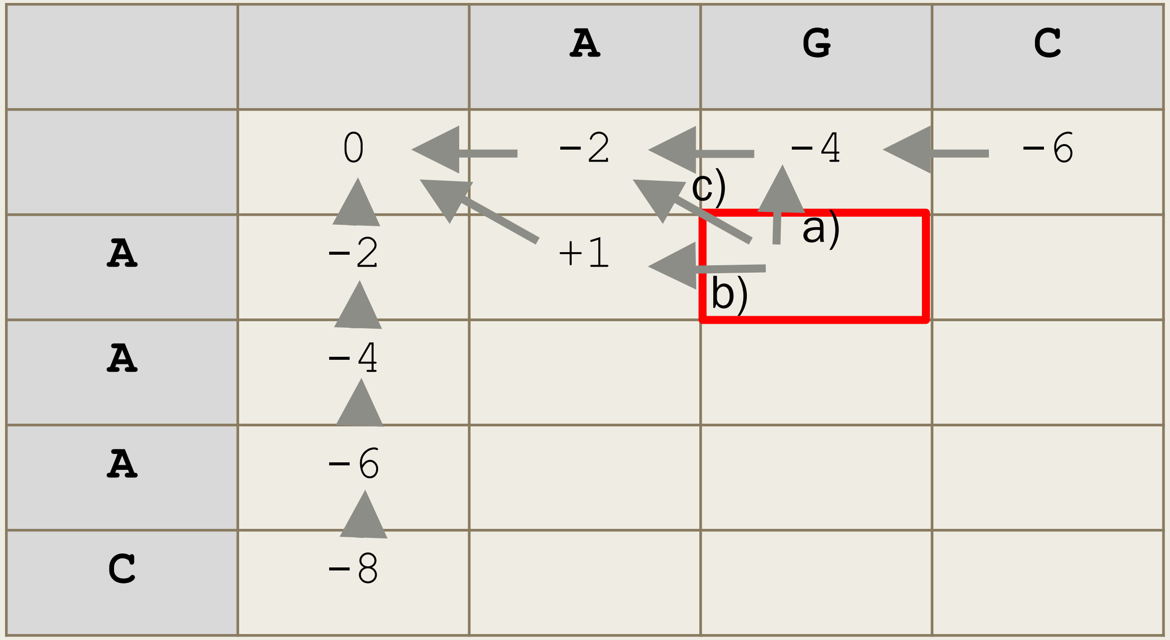
$g = -2$

AAA
- - -

Dynamic programing for alignment

Align the strings `AAAC` and `AGC`

a) Add a gap to y; -4-2=-6
b) Add a gap to x; +1-2=-1
c) Extend alignment; -2-1=-3



		A	**G**	**C**
	0	-2	-4	-6
A	-2	+1		
A	-4			
A	-6			
C	-8			

Recursive step

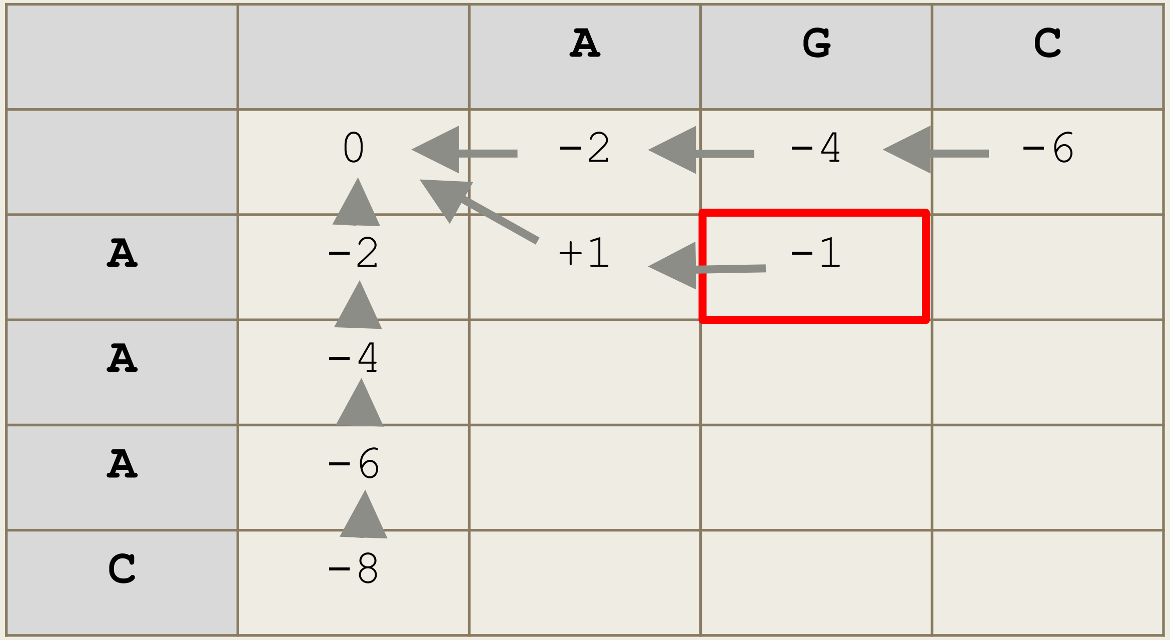
Dynamic programing for alignment

Align the strings `AAAC` and `AGC`

a) Add a gap to y; -4-2=-6

b) Add a gap to x; +1-2=-1

c) Extend alignment; -2-1=-3

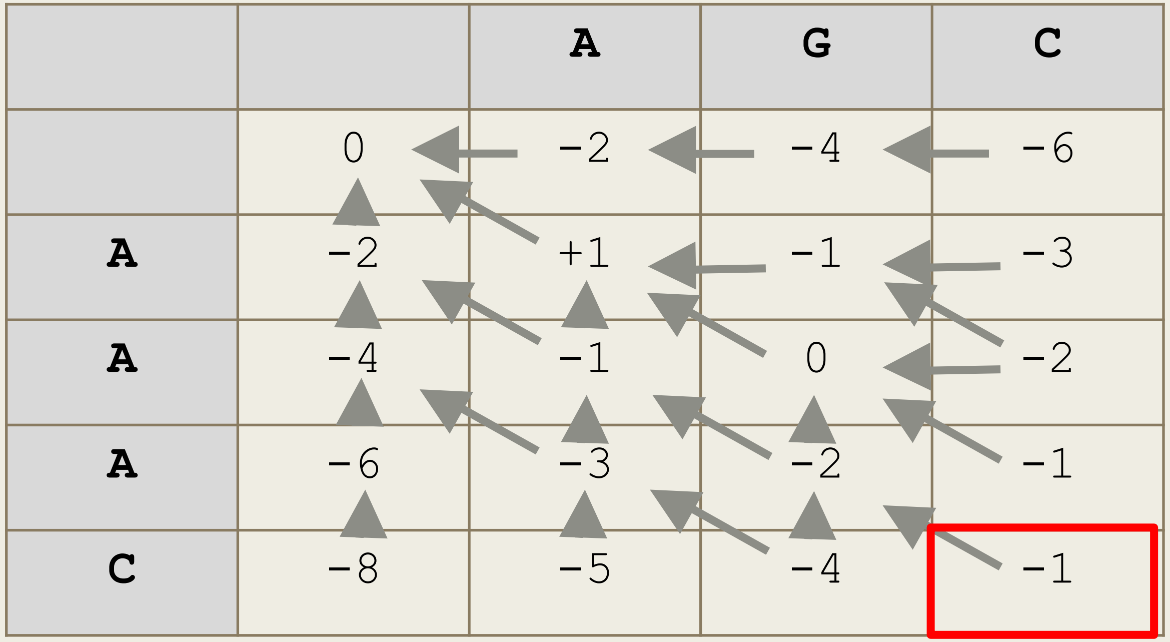


		A	G	C
	0	-2	-4	-6
A	-2	+1	-1	
A	-4			
A	-6			
C	-8			

Recursive step

Dynamic programing for alignment

Align the strings `AAAC` and `AGC`



		A	**G**	**C**
	0	-2	-4	-6
A	-2	+1	-1	-3
A	-4	-1	0	-2
A	-6	-3	-2	-1
C	-8	-5	-4	-1

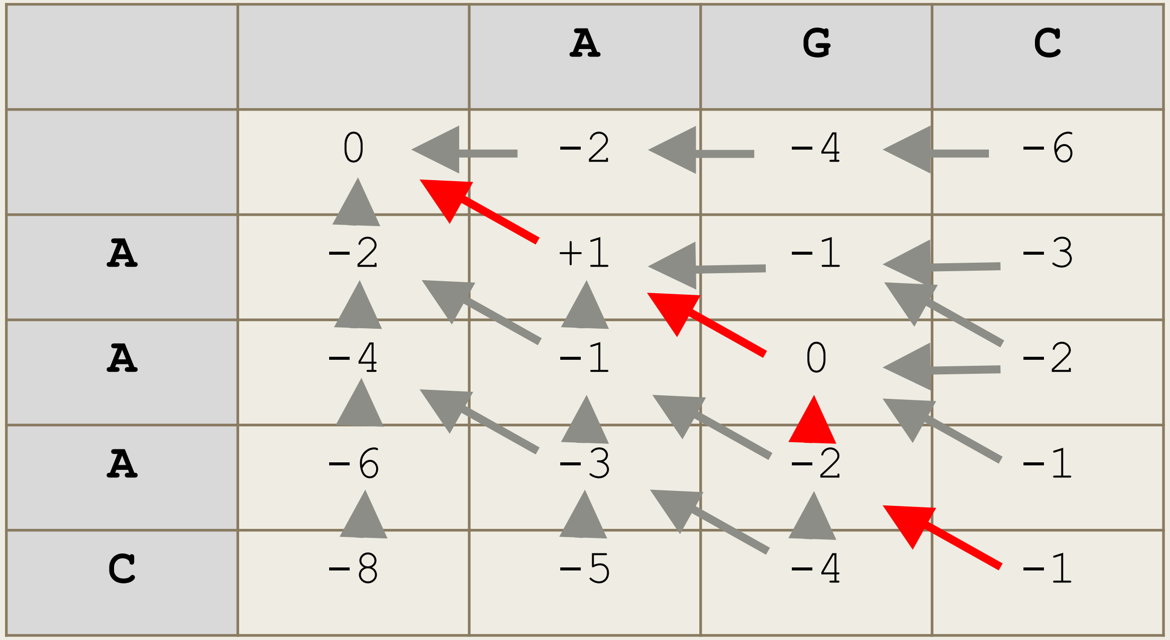
Recursive step

Dynamic programing for alignment

Align the strings `AAAC` and `AGC`

One optimal alignment:

```
AG−C
AAAC
```



		A	G	C
	0	-2	-4	-6
A	-2	+1	-1	-3
A	-4	-1	0	-2
A	-6	-3	-2	-1
C	-8	-5	-4	-1

Traceback step

Dynamic programing for alignment

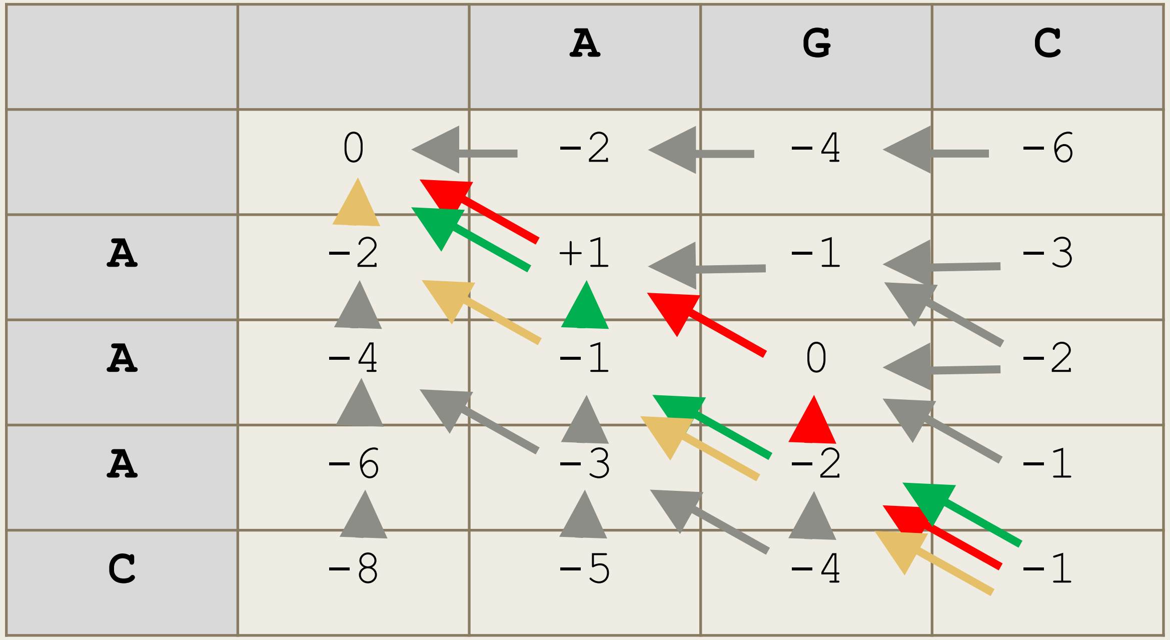
Align the strings `AAAC` and `AGC`

Optimal alignments:

```
AG-C
AAAC
```

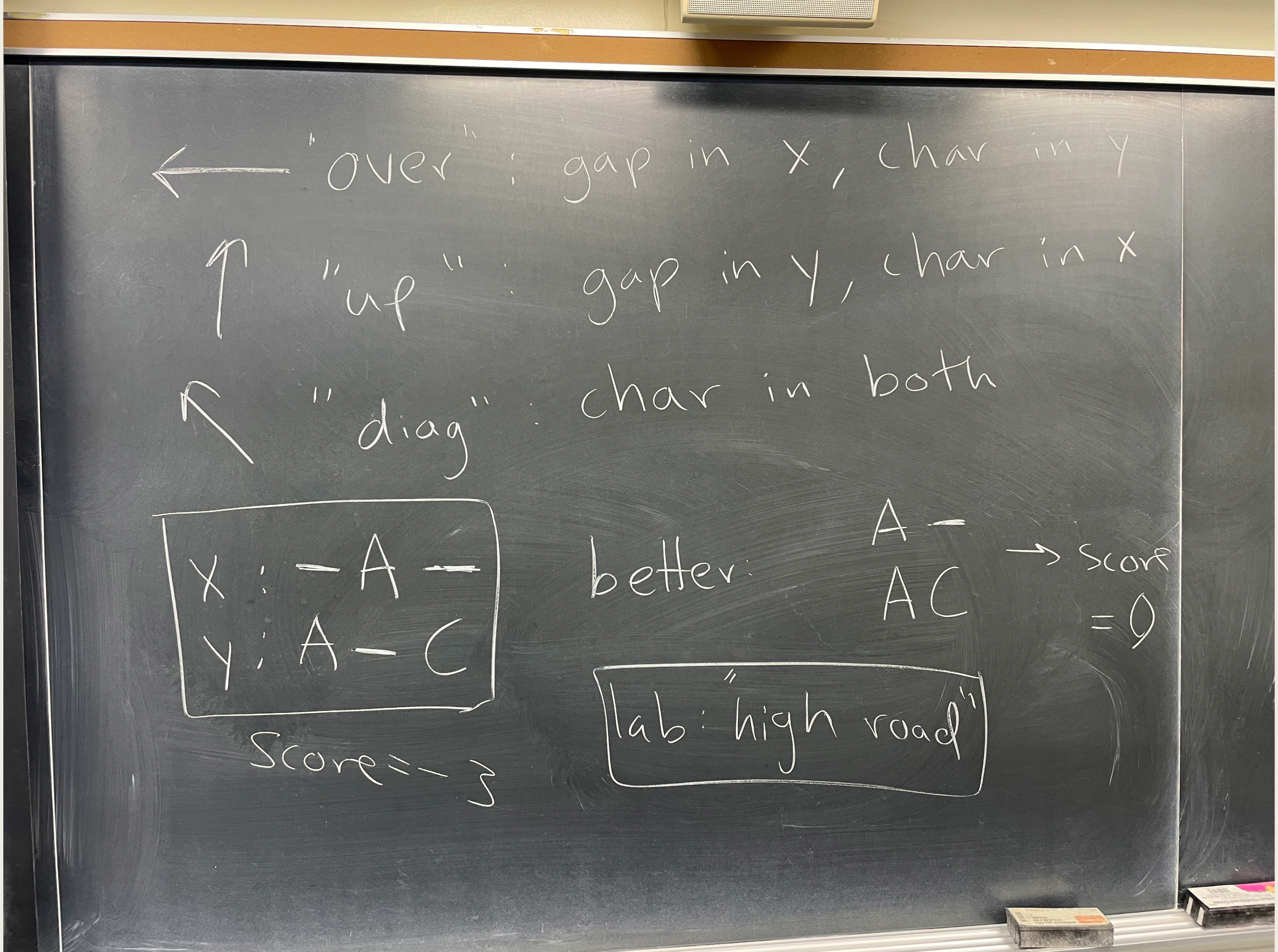
```
A-GC
AAAC
```

```
-AGC
AAAC
```



		A	G	C
	0	-2	-4	-6
A	-2	+1	-1	-3
A	-4	-1	0	-2
A	-6	-3	-2	-1
C	-8	-5	-4	-1

Traceback step



← "over": gap in x, char in y

↑ "up": gap in y, char in x

↖ "diag": char in both

```
x: – A –
y: A – C
```

Score = -3

better:

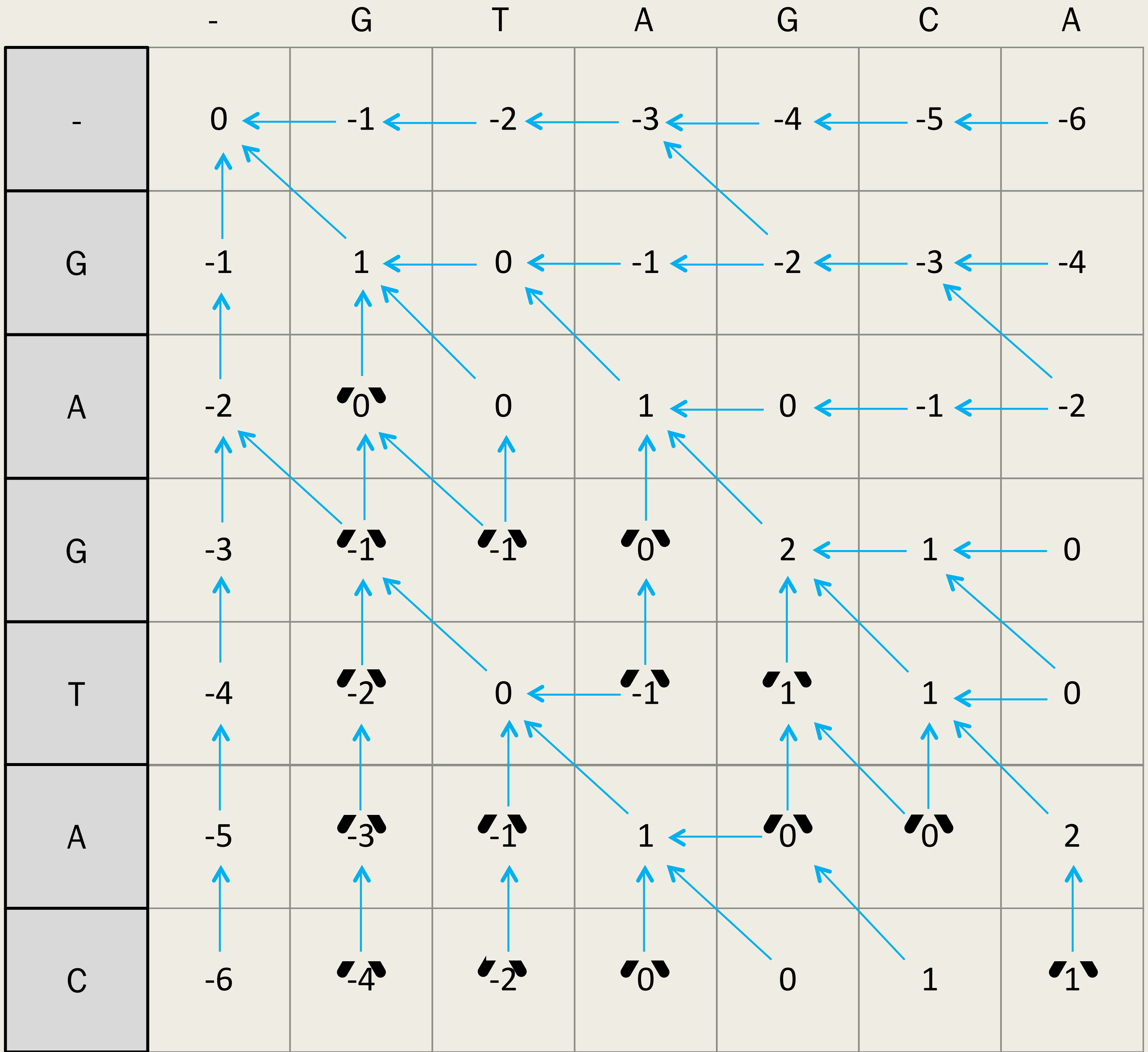
```
A –
A C
```

→ score = 0

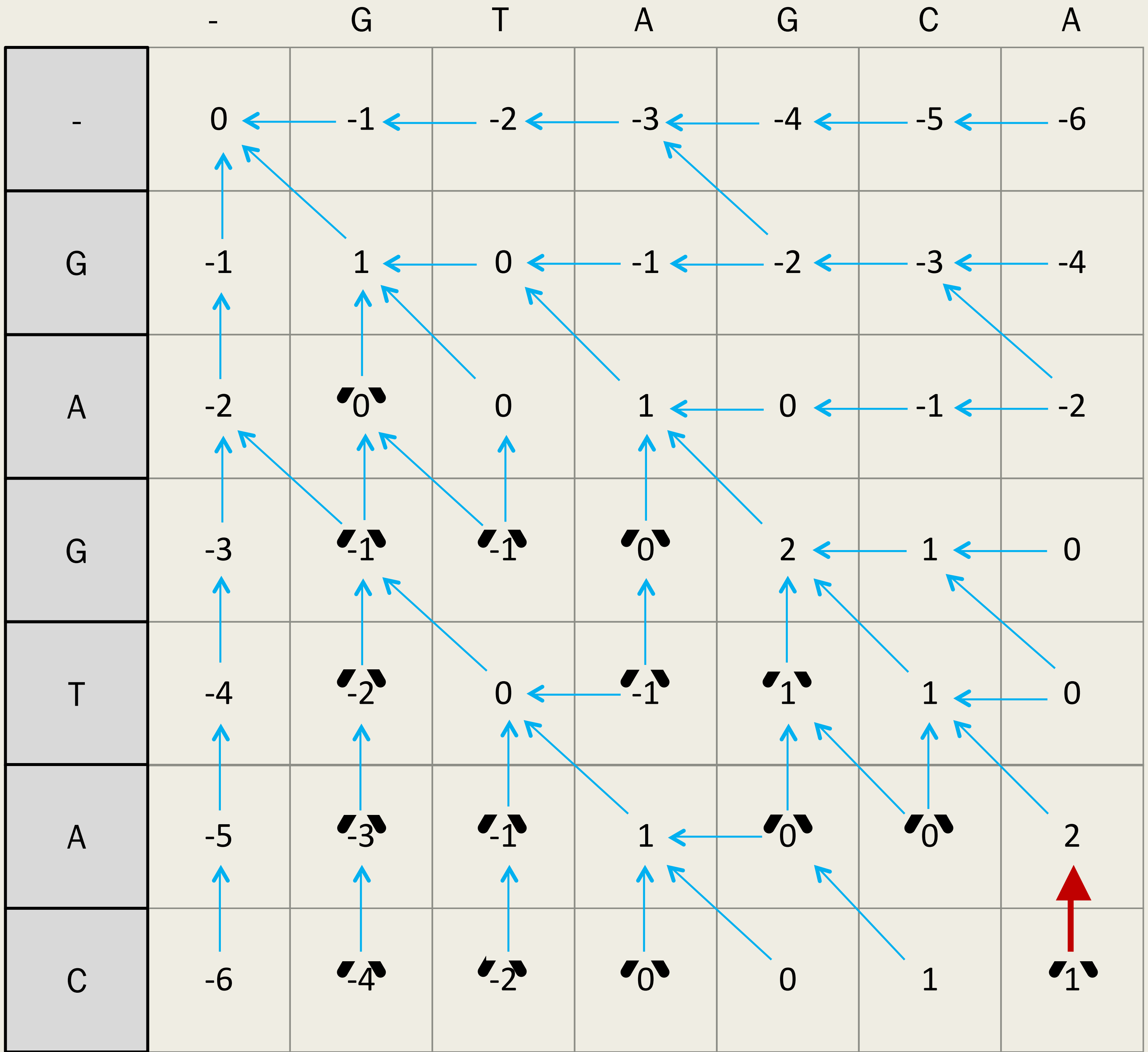
lab: "high road"

Handout 7, first page

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



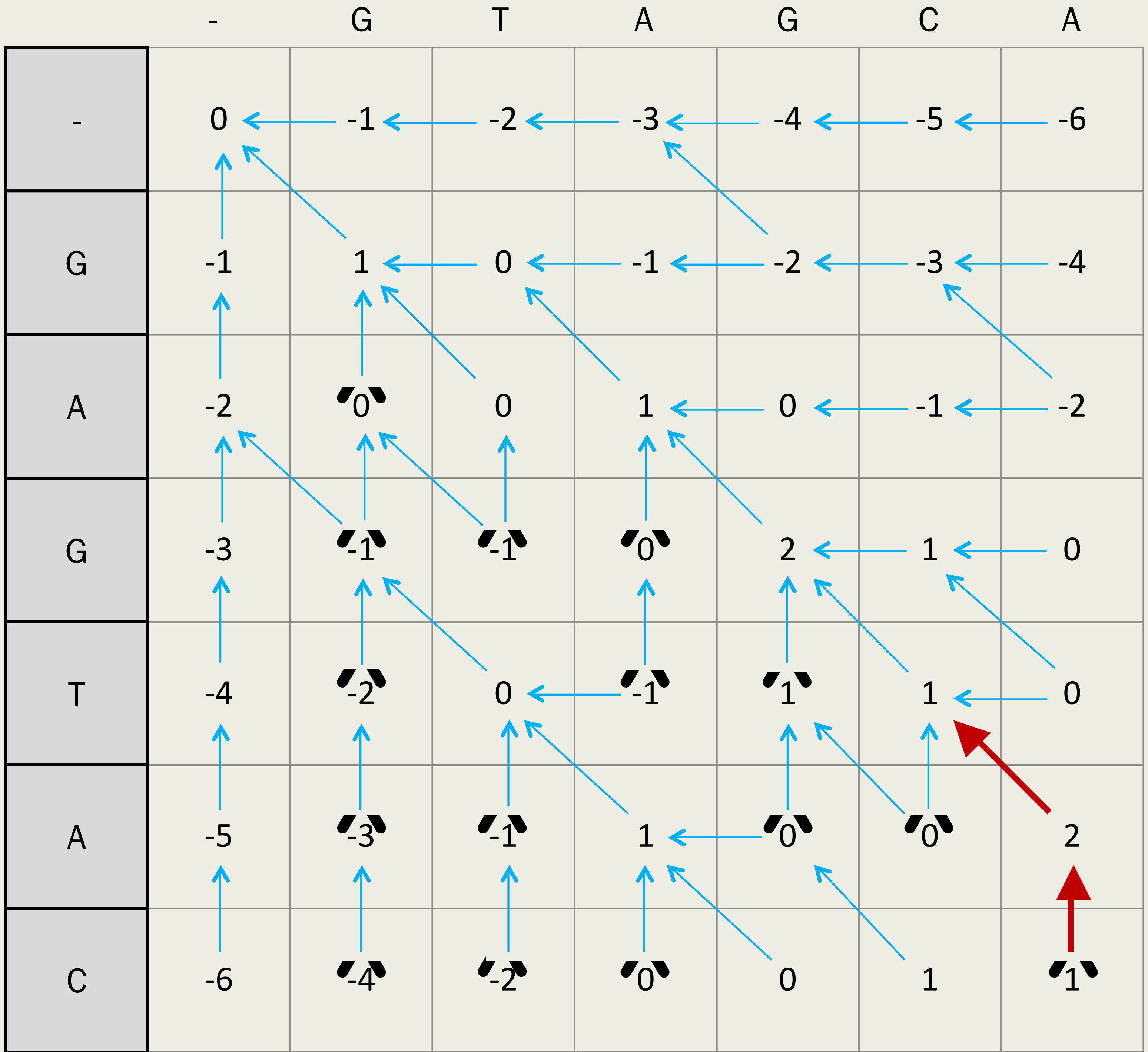
	-	G	T	A	G	C	A
-	0	-1	-2	-3	-4	-5	-6
G	-1	1	0	-1	-2	-3	-4
A	-2	0	0	1	0	-1	-2
G	-3	-1	-1	0	2	1	0
T	-4	-2	0	-1	1	1	0
A	-5	-3	-1	1	0	0	2
C	-6	-4	-2	0	0	1	1



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

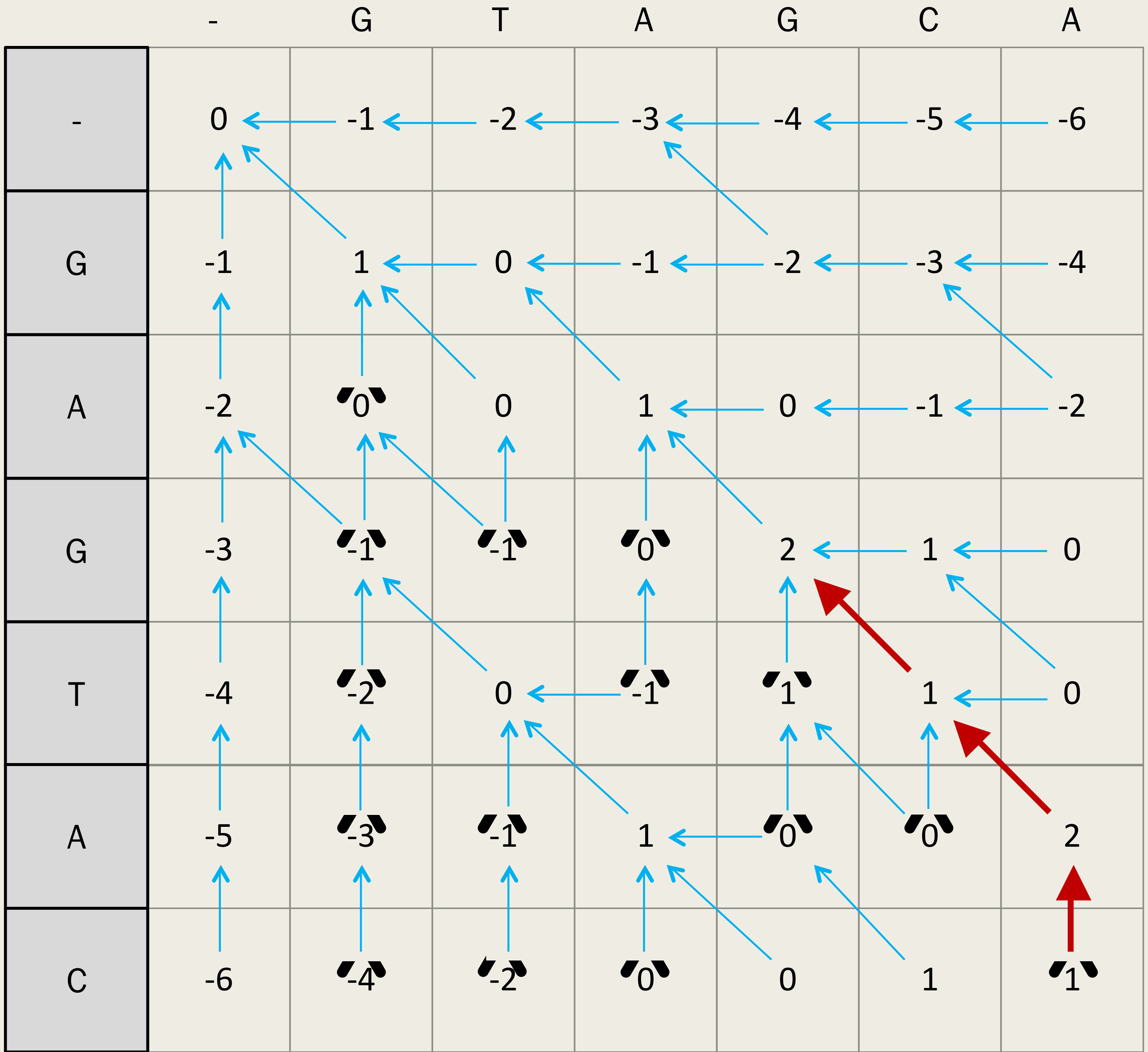
C

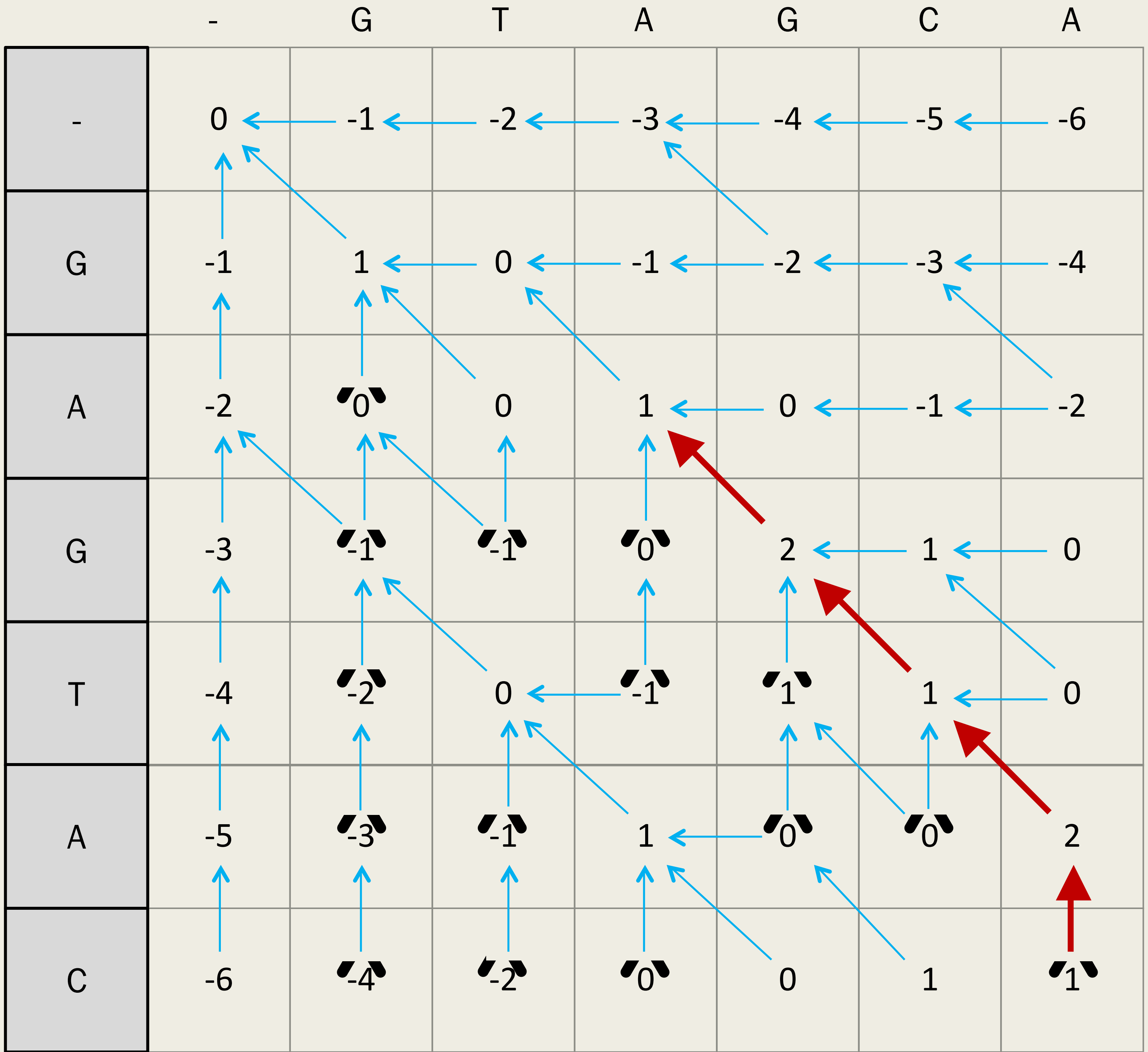
–



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

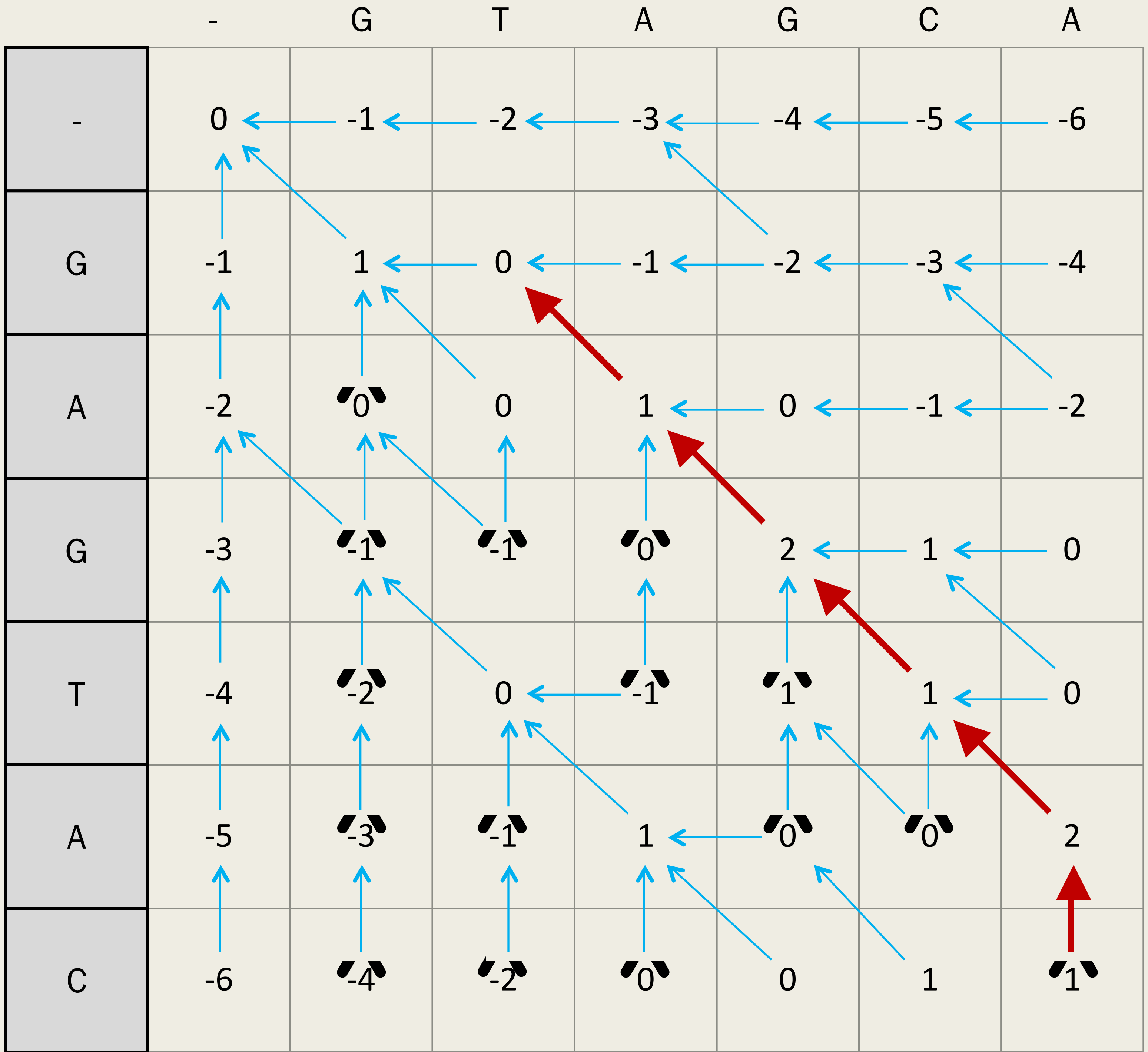
AC
A-





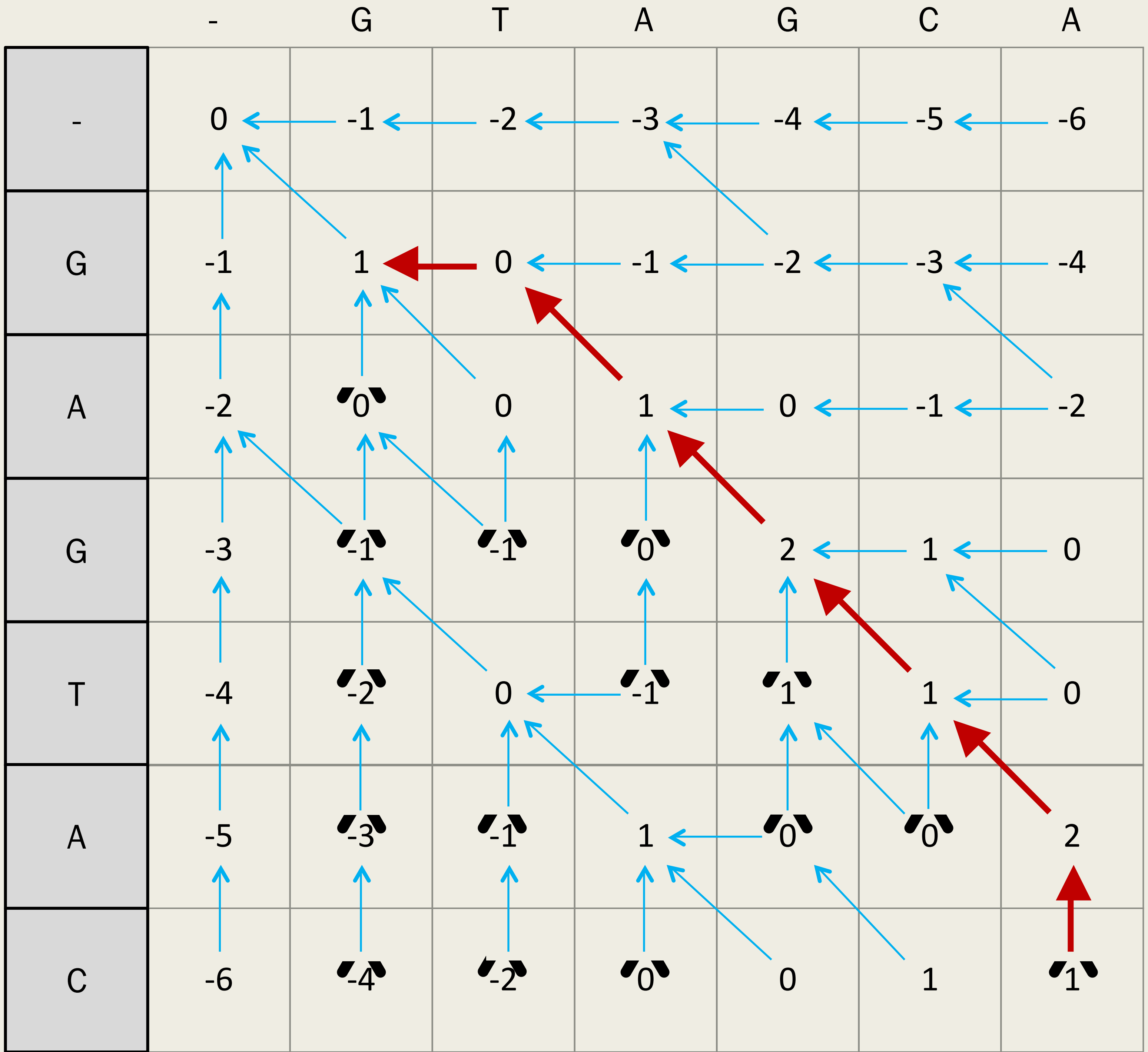
	-	G	T	A	G	C	A
-	0	-1	-2	-3	-4	-5	-6
G	-1	1	0	-1	-2	-3	-4
A	-2	0	0	1	0	-1	-2
G	-3	-1	-1	0	2	1	0
T	-4	-2	0	-1	1	1	0
A	-5	-3	-1	1	0	0	2
C	-6	-4	-2	0	0	1	1

GTAC
GCA-



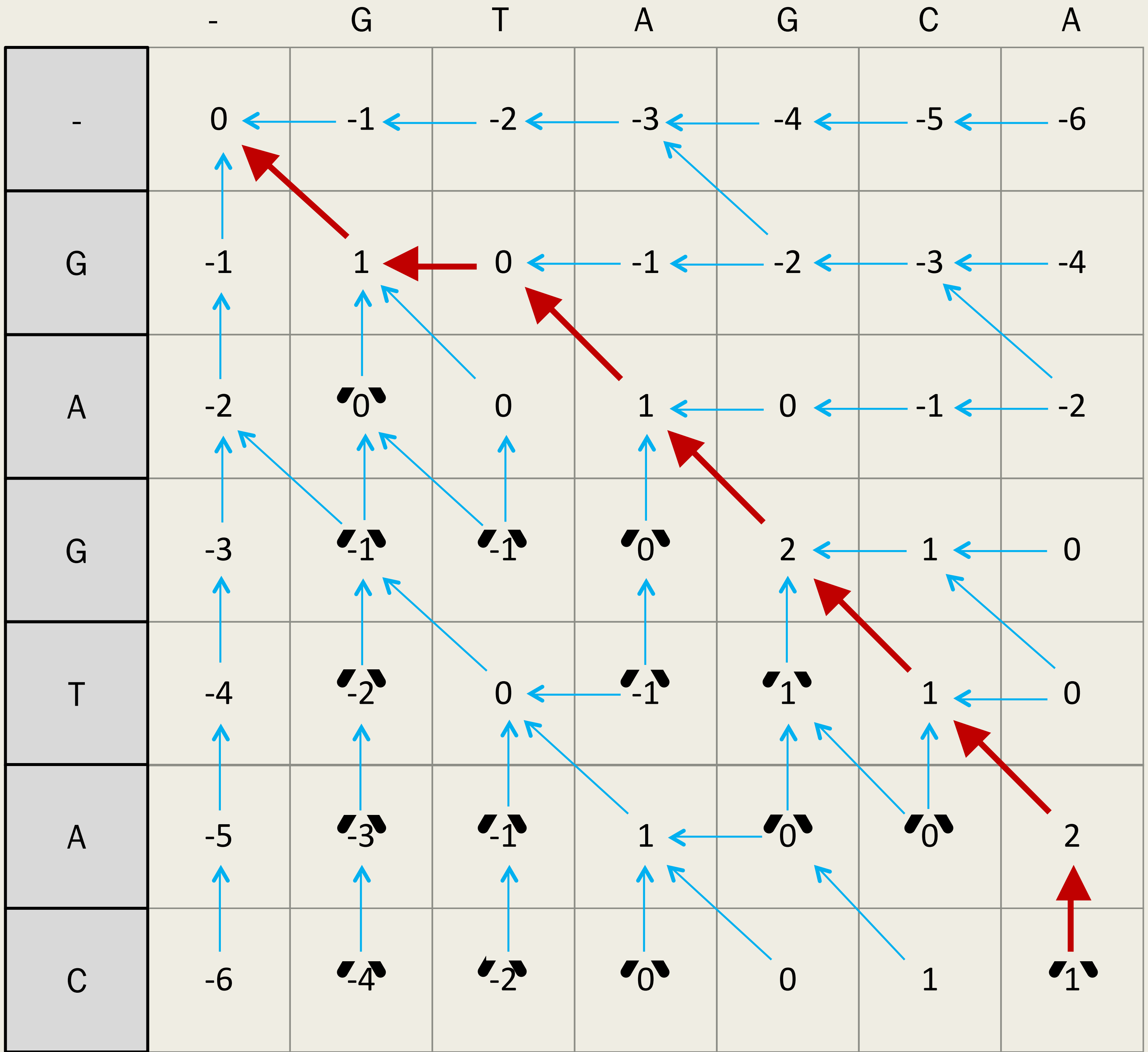
	-	G	T	A	G	C	A
-	0	-1	-2	-3	-4	-5	-6
G	-1	1	0	-1	-2	-3	-4
A	-2	0	0	1	0	-1	-2
G	-3	-1	-1	0	2	1	0
T	-4	-2	0	-1	1	1	0
A	-5	-3	-1	1	0	0	2
C	-6	-4	-2	0	0	1	1

AGTAC
AGCA-



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

```
-AGTAC
TAGCA-
```



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

```
G-AGTAC
GTAGCA-
```

Alignment with Protein Sequences

Central Dogma of Molecular Biology

- More correctly stated: “*The central dogma states that information in nucleic acid can be perpetuated or transferred but the transfer of information into protein is irreversible.*” (B. Lewin, 2004)

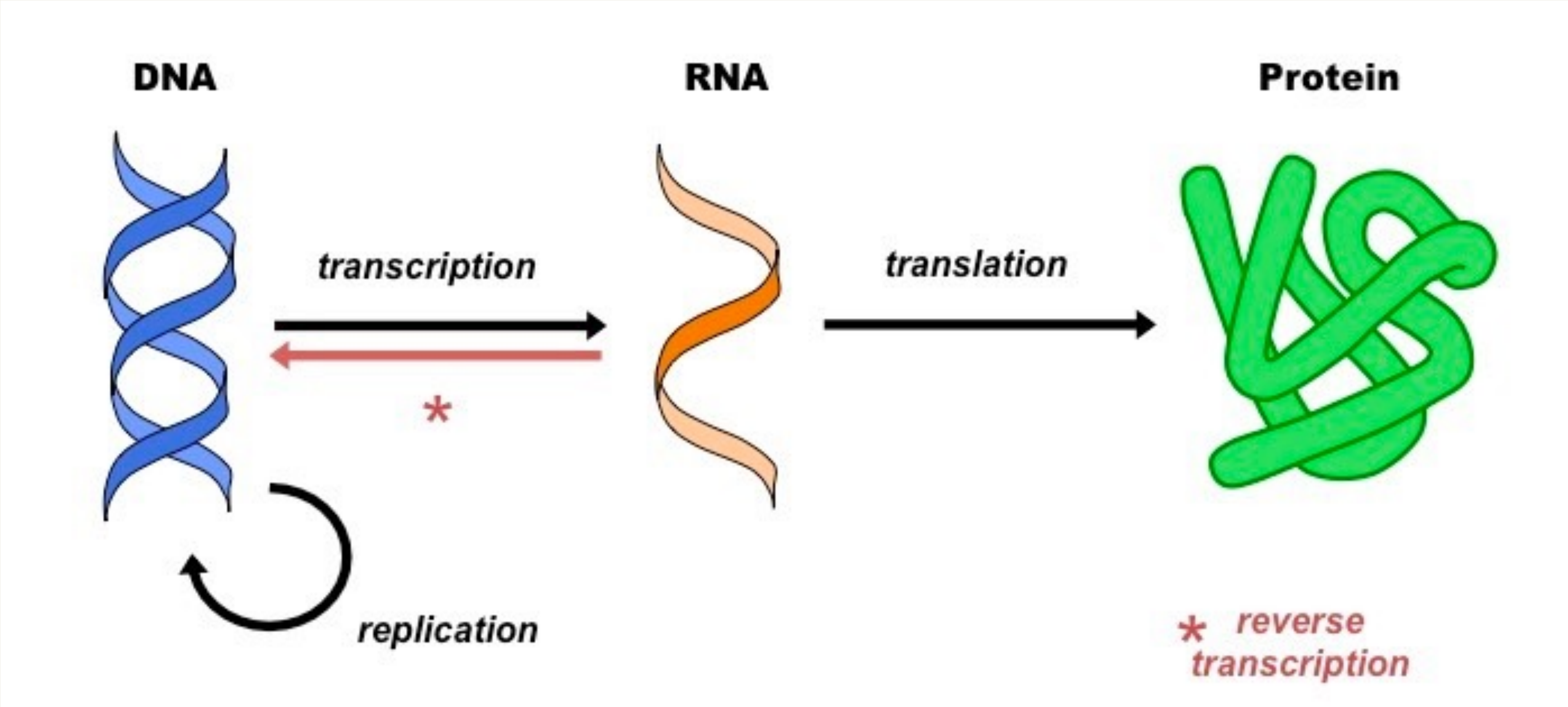


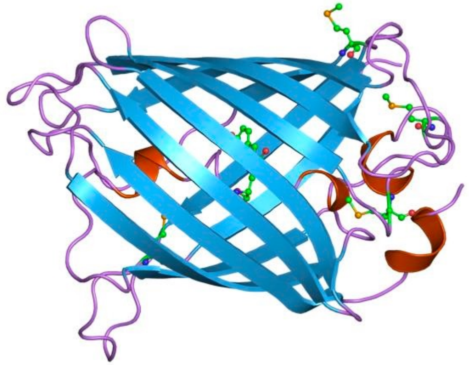
Image: http://ib.bioninja.com.au/

Central Dogma of Molecular Biology

- More correctly stated: “*The central dogma states that information in nucleic acid can be perpetuated or transferred but the transfer of information into protein is irreversible.”* (B. Lewin, 2004)

DNA	ATGCAATCAGATTAG
RNA	CAAUCAGAU
protein	Q S D

GFP protein example



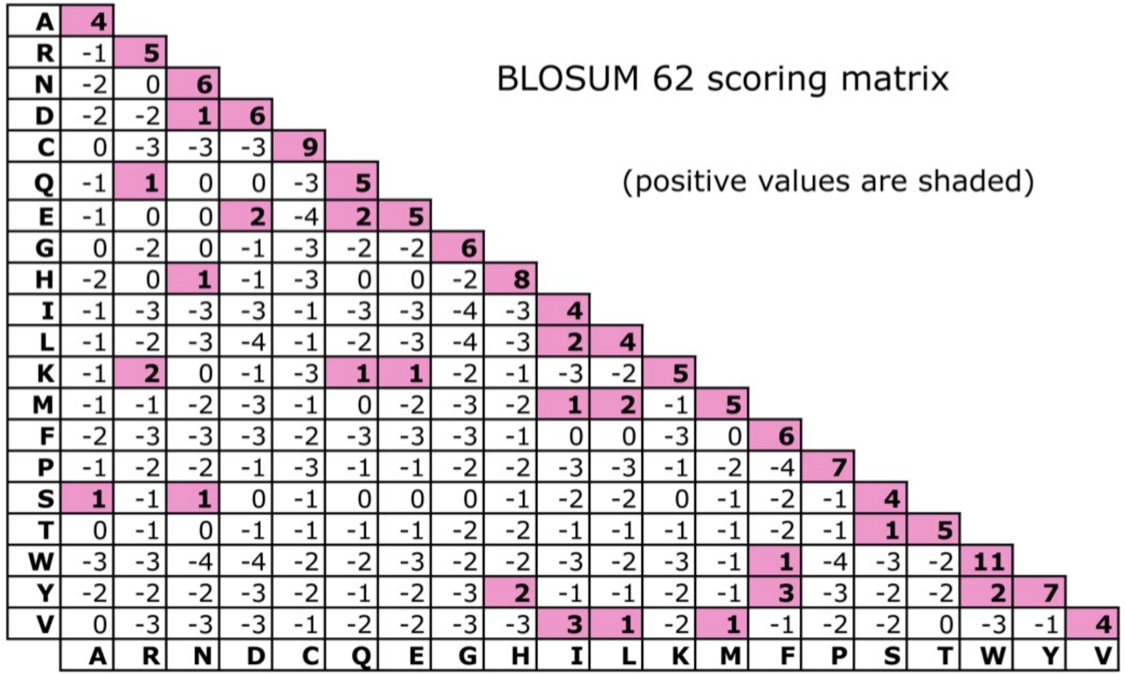
Codon table

A,GCU,GCC,GCA,GCG,AGA
R,CGU,CGC,CGA,CGG,AGG
N,AAU,AAC
D,GAU,GAC
C,UGU,UGC
Q,CAA,CAG
E,GAA,GAG
G,GGU,GGC,GGA,GGG
H,CAU,CAC
I,AUU,AUC,AUA
L,UUA,UUG,CUU,CUC,CUA,CUG
K,AAA,AAG
M,AUG
F,UUU,UUC
P,CCU,CCC,CCA,CCG
S,UCU,UCC,UCA,UCG,AGU,AGC
T,ACU,ACC,ACA,ACG
W,UGG
Y,UAU,UAC
V,GUU,GUC,GUA,GUG

Scoring alignments

Simple case: Each mismatch/gap scores -1, each match +1

More biologically relevant, different operations have different costs



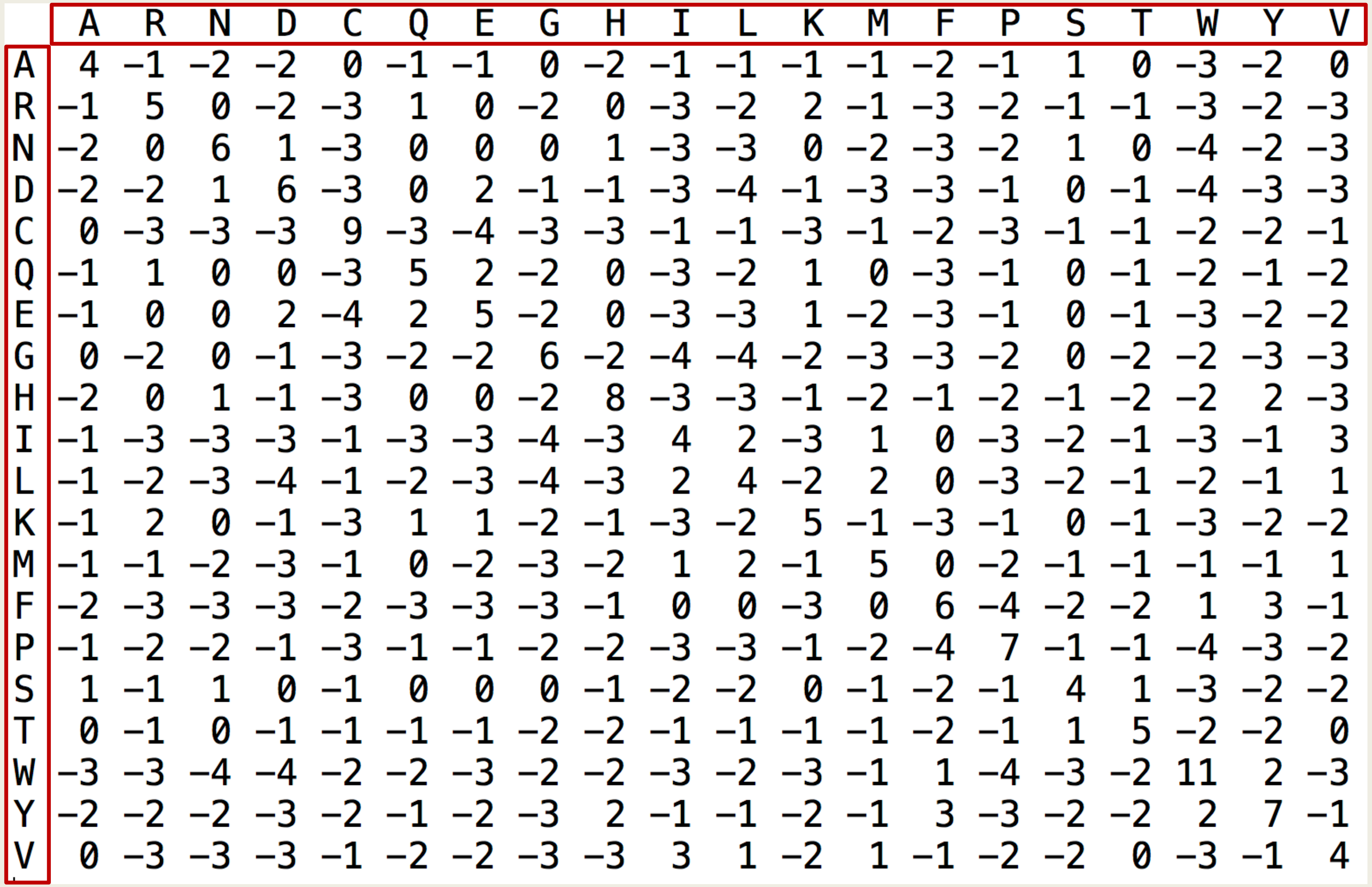
BLOSUM 62 scoring matrix

(positive values are shaded)

A	**4**																			
R	-1	**5**																		
N	-2	0	**6**																	
D	-2	-2	**1**	**6**																
C	0	-3	-3	-3	**9**															
Q	-1	**1**	0	0	-3	**5**														
E	-1	0	0	**2**	-4	**2**	**5**													
G	0	-2	0	-1	-3	-2	-2	**6**												
H	-2	0	**1**	-1	-3	0	0	-2	**8**											
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	**4**										
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	**2**	**4**									
K	-1	**2**	0	-1	-3	**1**	**1**	-2	-1	-3	-2	**5**								
M	-1	-1	-2	-3	-1	0	-2	-3	-2	**1**	**2**	-1	**5**							
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	**6**						
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	**7**					
S	**1**	-1	**1**	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	**4**				
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	**1**	**5**			
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	**1**	-4	-3	-2	**11**		
Y	-2	-2	-2	-3	-2	-1	-2	-3	**2**	-1	-1	-2	-1	**3**	-3	-2	-2	**2**	**7**	
V	0	-3	-3	-3	-1	-2	-2	-3	-3	**3**	**1**	-2	**1**	-1	-2	-2	0	-3	-1	**4**
	A	**R**	**N**	**D**	**C**	**Q**	**E**	**G**	**H**	**I**	**L**	**K**	**M**	**F**	**P**	**S**	**T**	**W**	**Y**	**V**

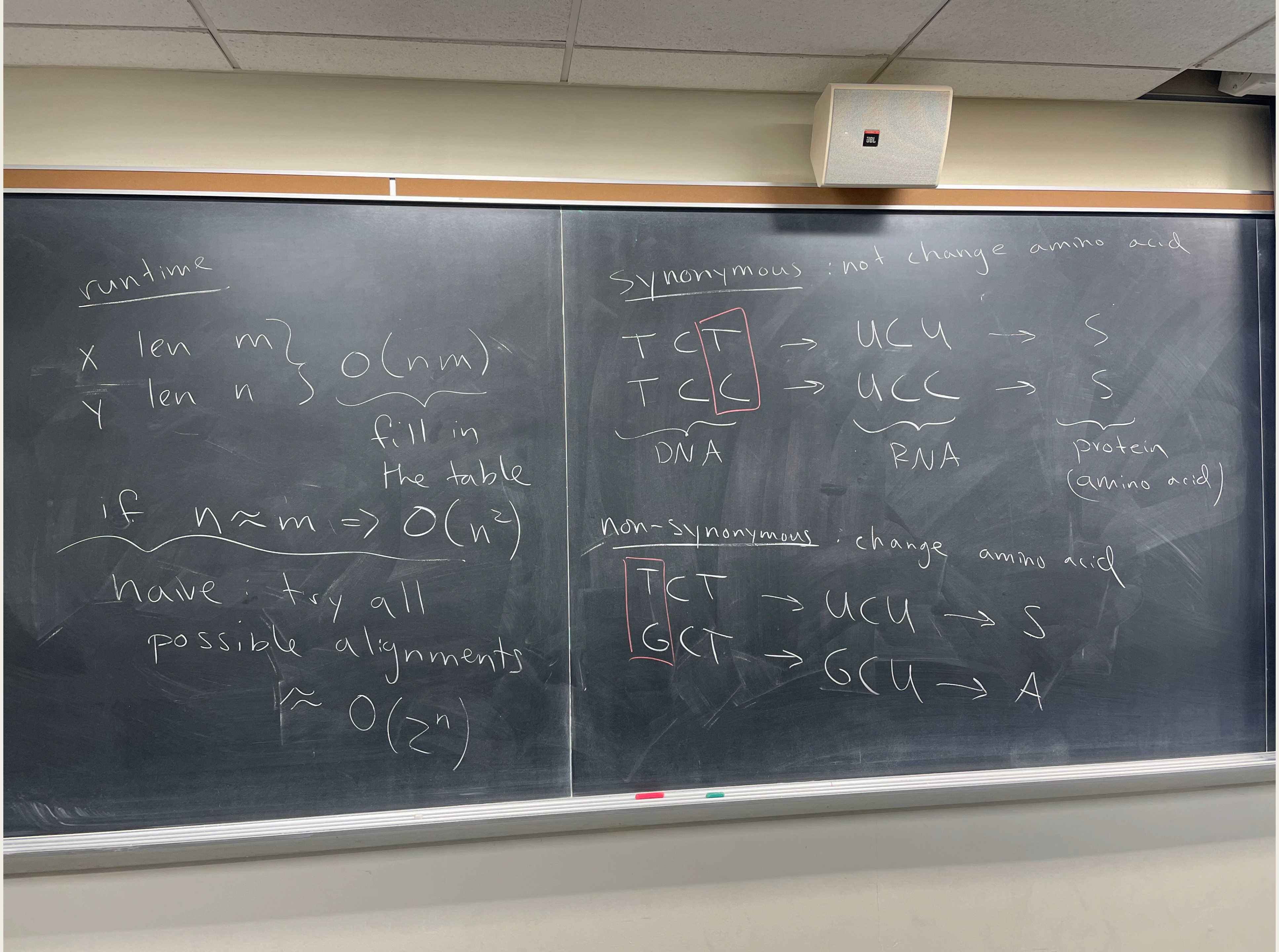
The values for amino acid substitutions were obtained from Henikoff S & Henikoff JG (1992) Amino acid substitutions matrices from protein blocks. *Proc. Natl. Acad. Sci.* **89**: 10915-10919.

Moving to proteins: BLOSUM match/mismatch matrix



	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	4	−1	−2	−2	0	−1	−1	0	−2	−1	−1	−1	−1	−2	−1	1	0	−3	−2	0
R	−1	5	0	−2	−3	1	0	−2	0	−3	−2	2	−1	−3	−2	−1	−1	−3	−2	−3
N	−2	0	6	1	−3	0	0	0	1	−3	−3	0	−2	−3	−2	1	0	−4	−2	−3
D	−2	−2	1	6	−3	0	2	−1	−1	−3	−4	−1	−3	−3	−1	0	−1	−4	−3	−3
C	0	−3	−3	−3	9	−3	−4	−3	−3	−1	−1	−3	−1	−2	−3	−1	−1	−2	−2	−1
Q	−1	1	0	0	−3	5	2	−2	0	−3	−2	1	0	−3	−1	0	−1	−2	−1	−2
E	−1	0	0	2	−4	2	5	−2	0	−3	−3	1	−2	−3	−1	0	−1	−3	−2	−2
G	0	−2	0	−1	−3	−2	−2	6	−2	−4	−4	−2	−3	−3	−2	0	−2	−2	−3	−3
H	−2	0	1	−1	−3	0	0	−2	8	−3	−3	−1	−2	−1	−2	−1	−2	−2	2	−3
I	−1	−3	−3	−3	−1	−3	−3	−4	−3	4	2	−3	1	0	−3	−2	−1	−3	−1	3
L	−1	−2	−3	−4	−1	−2	−3	−4	−3	2	4	−2	2	0	−3	−2	−1	−2	−1	1
K	−1	2	0	−1	−3	1	1	−2	−1	−3	−2	5	−1	−3	−1	0	−1	−3	−2	−2
M	−1	−1	−2	−3	−1	0	−2	−3	−2	1	2	−1	5	0	−2	−1	−1	−1	−1	1
F	−2	−3	−3	−3	−2	−3	−3	−3	−1	0	0	−3	0	6	−4	−2	−2	1	3	−1
P	−1	−2	−2	−1	−3	−1	−1	−2	−2	−3	−3	−1	−2	−4	7	−1	−1	−4	−3	−2
S	1	−1	1	0	−1	0	0	0	−1	−2	−2	0	−1	−2	−1	4	1	−3	−2	−2
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	1	5	−2	−2	0
W	−3	−3	−4	−4	−2	−2	−3	−2	−2	−3	−2	−3	−1	1	−4	−3	−2	11	2	−3
Y	−2	−2	−2	−3	−2	−1	−2	−3	2	−1	−1	−2	−1	3	−3	−2	−2	2	7	−1
V	0	−3	−3	−3	−1	−2	−2	−3	−3	3	1	−2	1	−1	−2	−2	0	−3	−1	4

Substitution scores between amino acids



runtime

$$\left.\begin{array}{l} X \text{ len } m \\ Y \text{ len } n \end{array}\right\} \underbrace{O(nm)}_{\text{fill in the table}}$$

$$\text{if } \underbrace{n \approx m \Rightarrow O(n^2)}$$

naive : try all possible alignments $\approx O(2^n)$

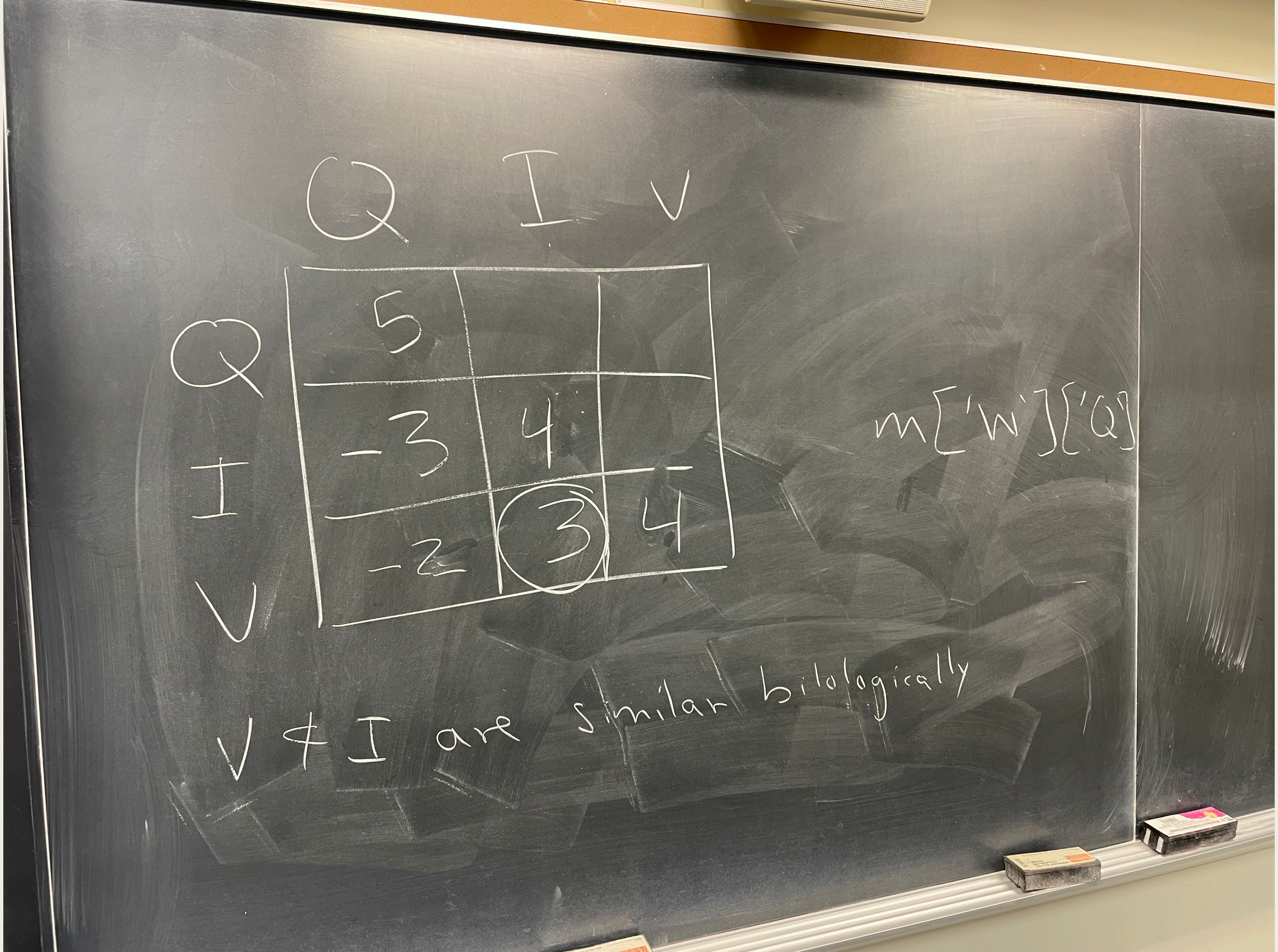
Synonymous : not change amino acid

$$\underbrace{\begin{array}{l} TCT \\ TCC \end{array}}_{\text{DNA}} \begin{array}{c} \rightarrow \\ \rightarrow \end{array} \underbrace{\begin{array}{l} UCU \\ UCC \end{array}}_{\text{RNA}} \begin{array}{c} \rightarrow \\ \rightarrow \end{array} \underbrace{\begin{array}{l} S \\ S \end{array}}_{\text{protein (amino acid)}}$$

non-synonymous : change amino acid

TCT → UCU → S

GCT → GCU → A



	Q	I	V
Q	5		
I	-3	4	
V	-2	3	4

m['W']['Q']

V & I are similar bilologically

Local Sequence Alignment

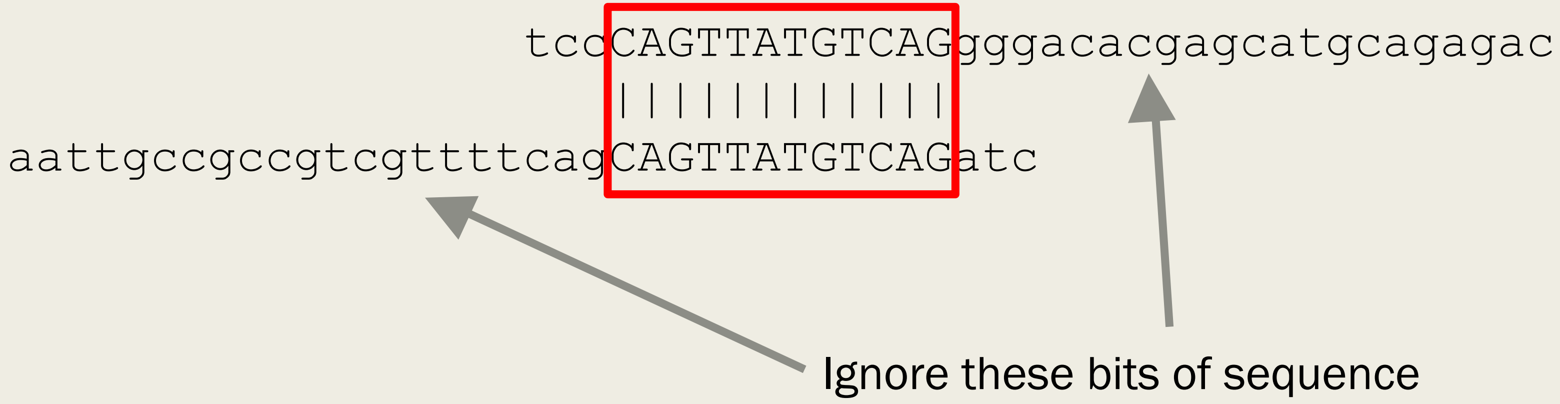
Local alignment

Find best aligning subsequences, but do not need to align the whole sequence

Global alignment

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

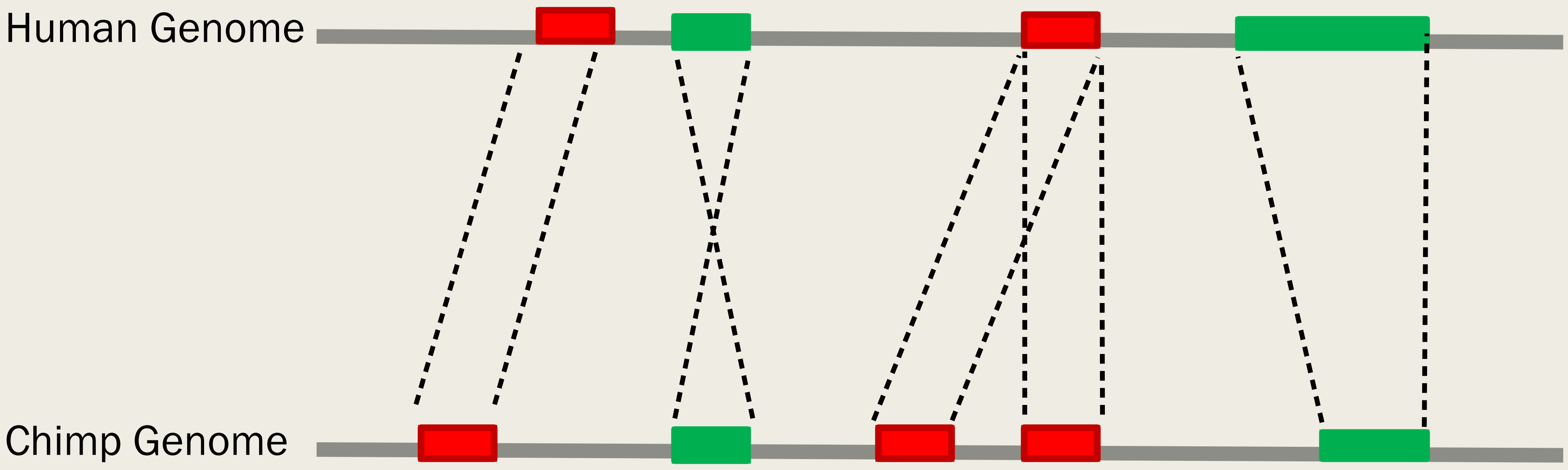
Local alignment



```
                  tccCAGTTATGTCAGgggacacgagcatgcagagac
                     ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

Ignore these bits of sequence

Local alignment



Exons (protein coding sequence) ~1%

Other conserved sequence (noncoding RNA, regulatory etc…) 5-10%

“Junk” not evolutionarily conserved ~90%

Local alignment algorithm

Three differences with global alignment:

		A	A	G	A
T					
T					
A					
A					
G					

Local alignment algorithm

Three differences with global alignment: **1) Initialize edges to 0**

		A	**A**	**G**	**A**
	0	0	0	0	0
T	0				
T	0				
A	0				
A	0				
G	0				

Local alignment algorithm

Three differences with global alignment:

1) Initialize edges to 0
2) **Do not allow scores to go negative**

		A	A	G	A
	0	0	0	0	0
T	0	-1			
T	0				
A	0				
A	0				
G	0				

Local alignment algorithm

Three differences with global alignment:

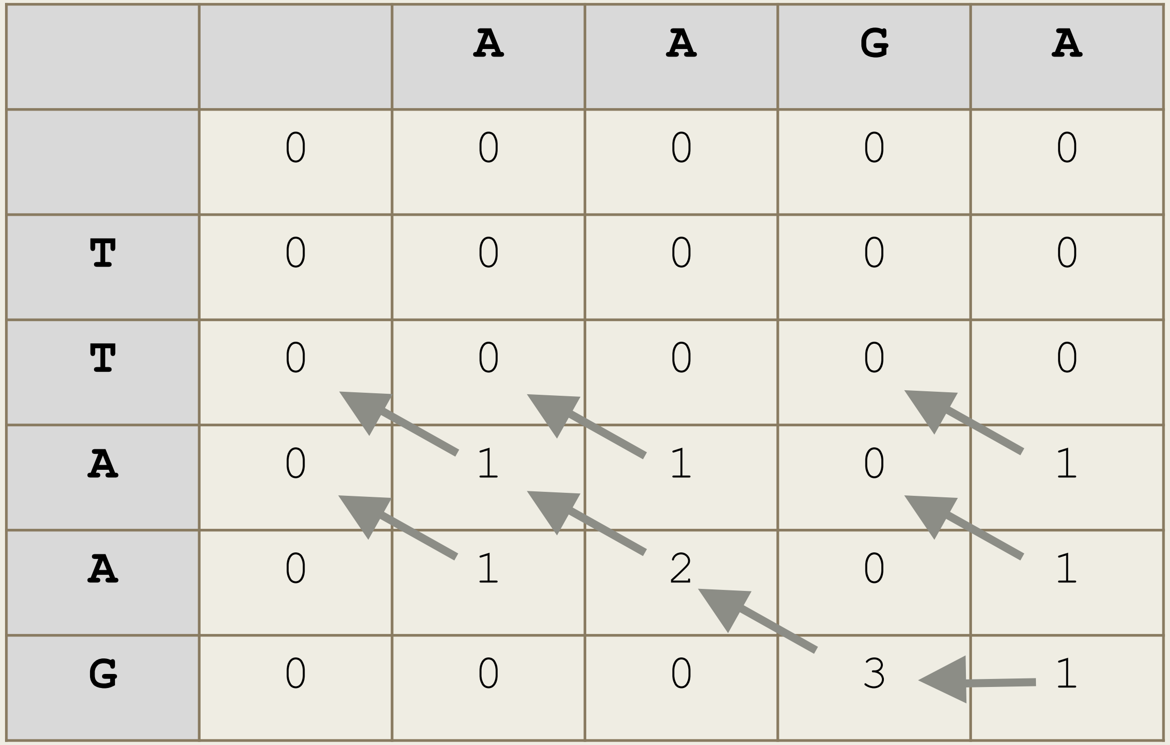
1) Initialize edges to 0
2) **Do not allow scores to go negative**

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

Local alignment algorithm

Three differences with global alignment:

1) Initialize edges to 0
2) Do not allow scores to go negative
3) **Find alignment from best score in the table (not necessarily bottom right)**



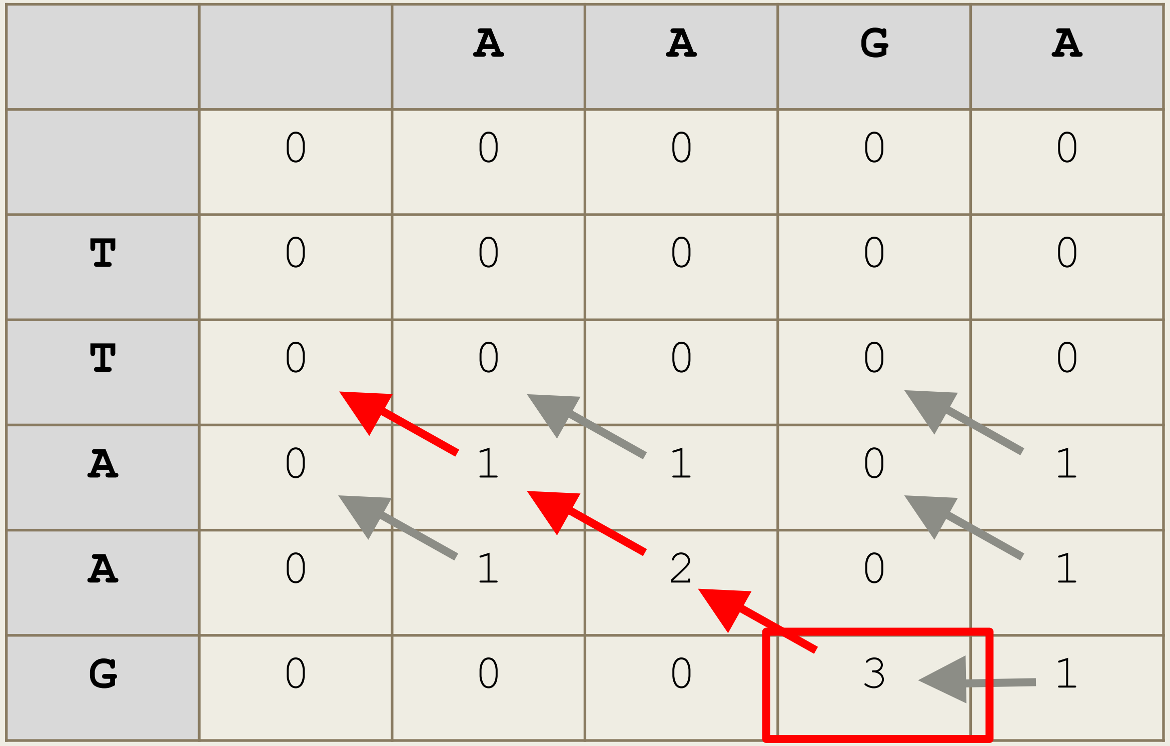
		A	A	G	A
	0	0	0	0	0
T	0	0	0	0	0
T	0	0	0	0	0
A	0	1	1	0	1
A	0	1	2	0	1
G	0	0	0	3	1

Local alignment algorithm

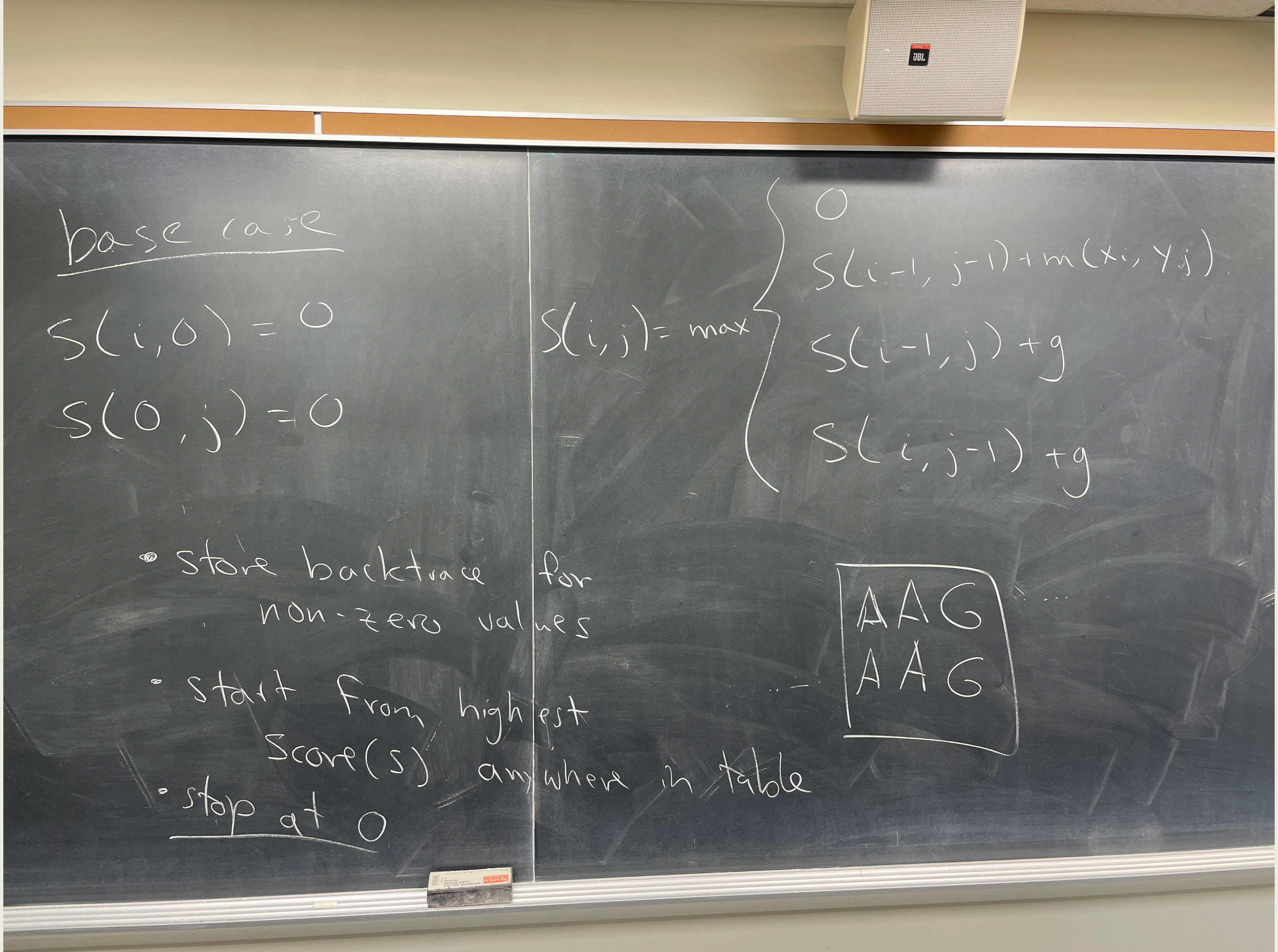
Three differences with global alignment:

1) Initialize edges to 0
2) Do not allow scores to go negative
3) **Find alignment from best score in the table (not necessarily bottom right)**

```
  AAGa
ttAAG
```



		A	A	G	A
	0	0	0	0	0
T	0	0	0	0	0
T	0	0	0	0	0
A	0	1	1	0	1
A	0	1	2	0	1
G	0	0	0	3	1



base case

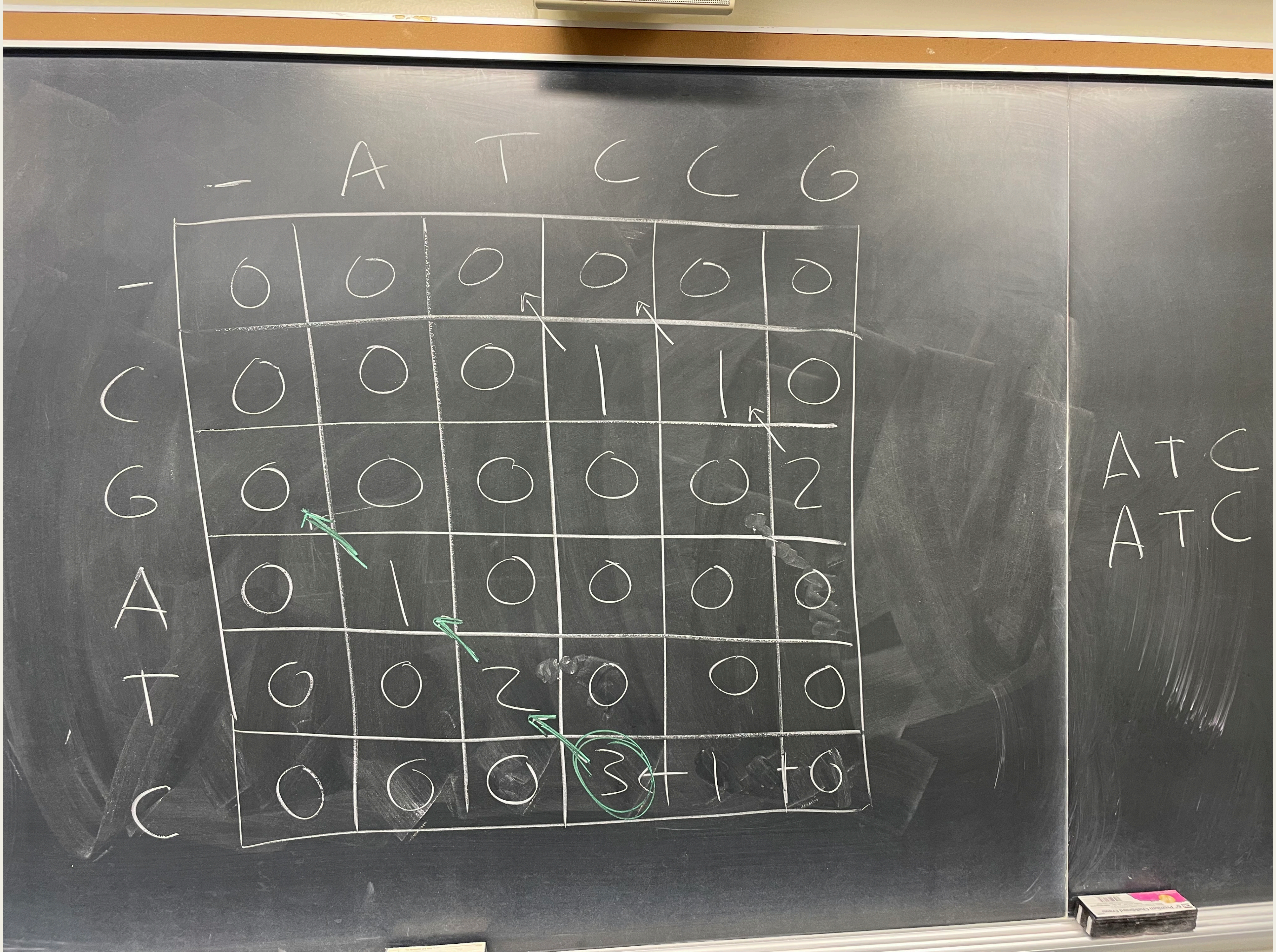
$S(i,0) = 0$

$S(0,j) = 0$

$$S(i,j) = \max \begin{cases} 0 \\ S(i-1,j-1) + m(x_i, y_j) \\ S(i-1,j) + g \\ S(i,j-1) + g \end{cases}$$

- store backtrace for non-zero values
- start from highest score(s) anywhere in table
- stop at 0

$$\boxed{\begin{matrix} \text{AAG} \\ \text{AAG} \end{matrix}}$$



	-	A	T	C	C	G
-	0	0	0	0	0	0
C	0	0	0	1	1	0
G	0	0	0	0	0	2
A	0	1	0	0	0	0
T	0	0	2	0	0	0
C	0	0	0	3	← 1	← 0

ATC
ATC