

Bioinformatics 1: Lecture 3

- Pairwise alignment
- Substitution
- Dynamic Programming algorithm

Exact match scoring matrix

To prepare an alignment, we first consider the score for aligning (associating) any one character of the first sequence with any one character of the second sequence.

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

Exact match
1/0

The cost of mutation is not a constant

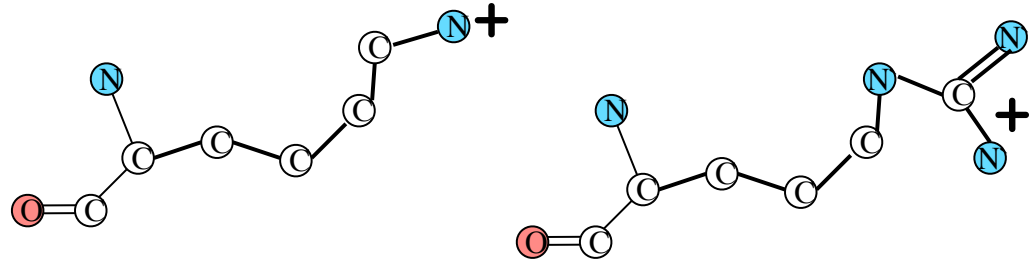
DNA: A change in the 3rd base in a codon, and sometimes the first base, sometimes conserves the amino acid. No selective pressure.

Protein: A change in amino acids that are in the same chemical class conserve their chemical environment. For example: Lys to Arg is conservative because both are positively charged.

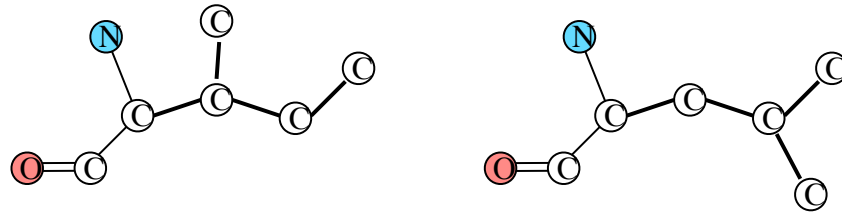
1st position	2nd position				3rd position
	U	C	A	G	
U	Phe	Ser	Tyr	Cys	U
	Phe	Ser	Tyr	Cys	C
	Leu	Ser	STOP	STOP	A
	Leu	Ser	STOP	Trp	G
C	Leu	Pro	His	Arg	U
	Leu	Pro	His	Arg	C
	Leu	Pro	Gln	Arg	A
	Leu	Pro	Gln	Arg	G
A	Ile	Thr	Asn	Ser	U
	Ile	Thr	Asn	Ser	C
	Ile	Thr	Lys	Arg	A
	Met	Thr	Lys	Arg	G
G	Val	Ala	Asp	Gly	U
	Val	Ala	Asp	Gly	C
	Val	Ala	Glu	Gly	A
	Val	Ala	Glu	Gly	G

Conservative amino acid changes

Lys <--> Arg



Ile <--> Leu



Ser <--> Thr

Asp <--> Glu

Asn <--> Gln

If the “chemistry” of the sidechain is conserved, then the mutation is less likely to change structure/function.

Amino acid substitution matrices

Two 20x20 substitution matrices are used: BLOSUM & PAM.

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

Each number is the score for aligning a single pair of amino acids.

Calculate the score for this alignment:

ACEPGAA
ASDDGTV

BLOSUM62

Scoring matrix

For protein alignments, first set up the scoring matrix by filling in the appropriate substitution score.

		→						
		A	C	E	P	G	A	A
↓	A	4	0	-1	0	-1	4	4
	S	1	-1	-4	-3	-3	0	0
	D	-2	-3	2	-1	-1	-2	-2
	D	-2	-3	2	-1	-1	-2	-2
	G	0	-3	-2	-2	6	0	0
	T	0	-1	-1	-1	-2	0	0
	V	0	-1	-2	-2	-3	0	0

Inexact match
using BLOSUM62
substitution
matrix:
score = 10

Pseudo code for setting up the alignment matrix

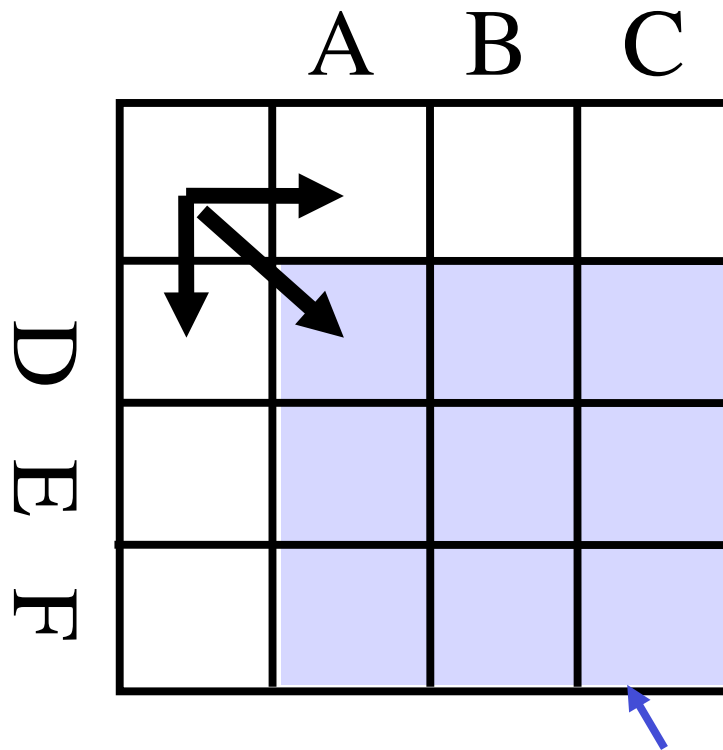
```
read blosum[1..20][1..20]
aa[1..20]={A,C,D,E,F,G,H,I,J,K,L,M,N,P,Q,R,S,T,V,W,Y}
read firstseq[1..N]
convert firstseq to numbers 1..20 using aa
read secondseq[1..M]
convert secondseq to numbers 1..20 using aa

alignmentmatrix[1..N][1..M]=0

for (i from 1 to N) do
  for (j from 1 to M) do
    alignmentmatrix[i][j] = blosum[firstseq[i]][secondseq[j]]
  enddo
enddo

write alignmentmatrix[1..N][1..M]
```

An Alignment as a Path through the Alignment Matrix



There are directions for each step: down, right, or diagonal.

The alignment is complete when we reach the lower right-hand corner box.

Imagine each of these boxes has a score in it. (i.e. from BLOSUM)

arrows



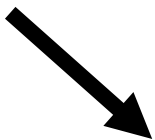
Upper seq advances by one,
Lower seq advances by zero.
Gap in lower seq.

X
~



Upper seq advances by zero,
Lower seq advances by one.
Gap in upper seq.

~
Y



Upper seq advances by one,
Lower seq advances by one.
Match.

X
Y

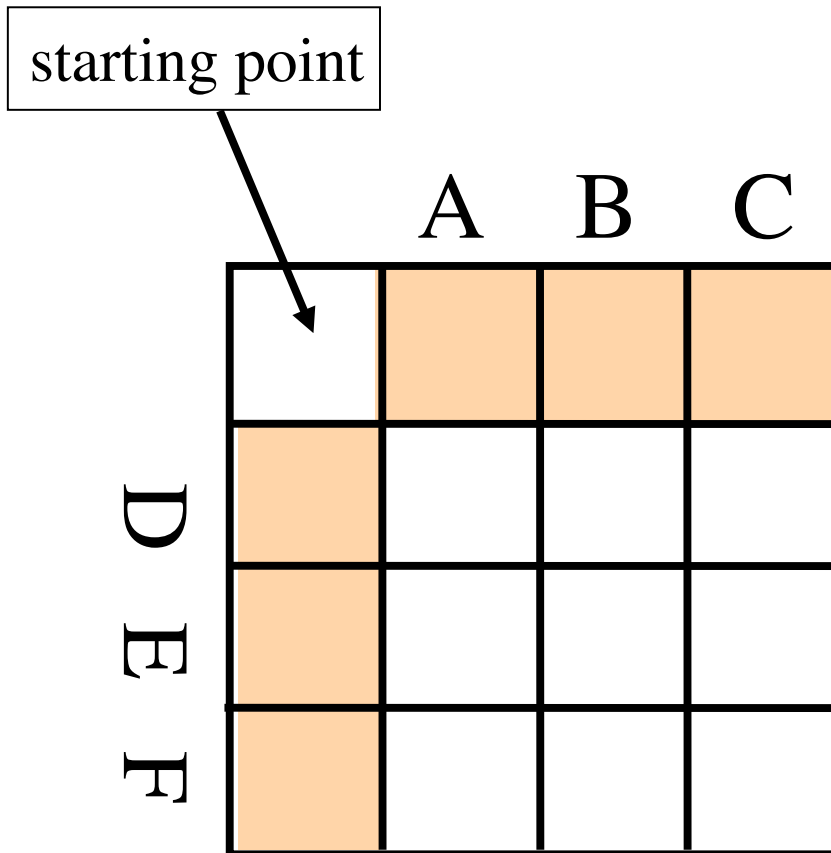
definitions

Gap

Insertion

Deletion

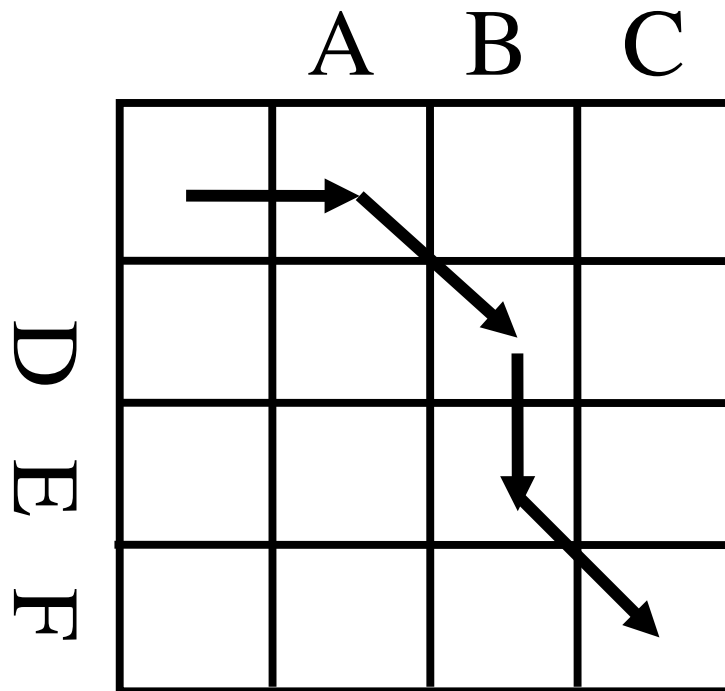
Gap rows



The size of the alignment matrix is $(N+1) \times (M+1)$, where N and M are the lengths of the two sequences.

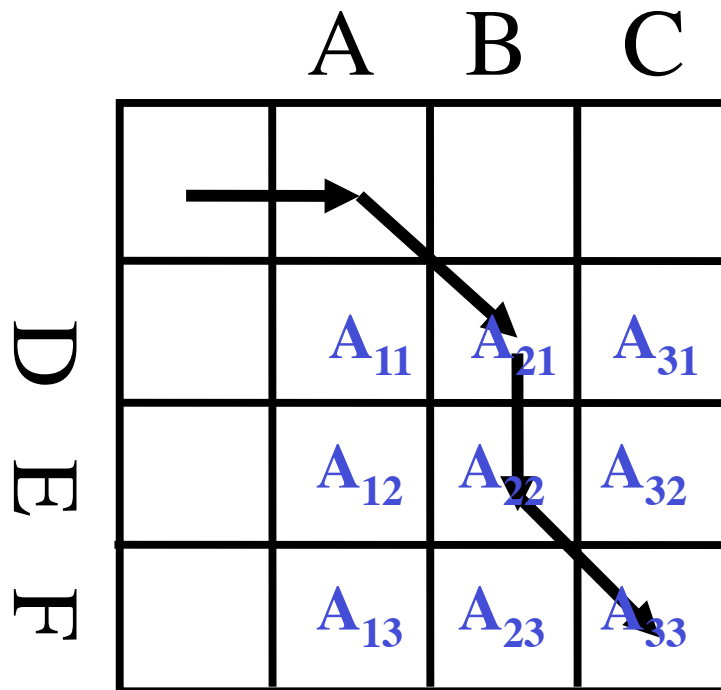
Rows on top and left ("gap" rows) have no scores.

An alignment



A B ~ C
~ D E F

Scoring an alignment with gaps



A B ~ C
~ D E F

Start with score = 0

For each step:

if **gap**: add a gap penalty

if **match**: add A_{ij} value.

Pseudocode for scoring an alignment with gaps

```
program scoremyalignment

right=1; down=2; diag=3;
Aseq=0; Bseq=0; gappenalty=2;
read myfile, A[0..3,0..3]
walk[1..4]=(right,diag,down,diag)
score = 0.
for i from 1 to len(walk) do
  if (walk[i]==right) then
    score = score + gappenalty
    Aseq = Aseq + 1
  elseif (walk[i]==down) then
    score = score + gappenalty
    Bseq = Bseq + 1
  elseif (walk[i]==diag)
then
    Aseq = Aseq + 1
    Bseq = Bseq + 1
    score = score + A[Aseq,Bseq]
  endif
enddo
write *, "Score =",score
end program scoremyalignment
```

right, down, diag are arbitrary constants in this case

Aseq, Bseq are counters

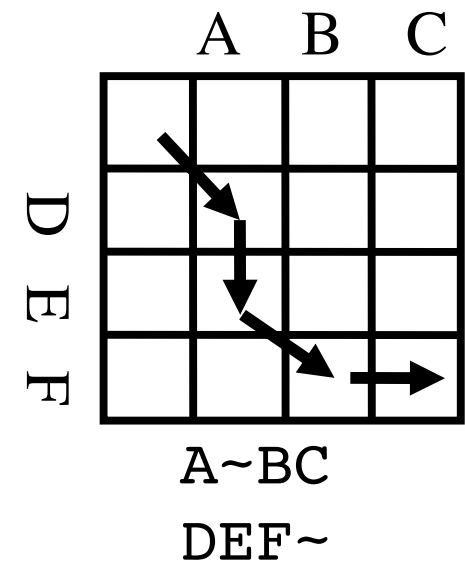
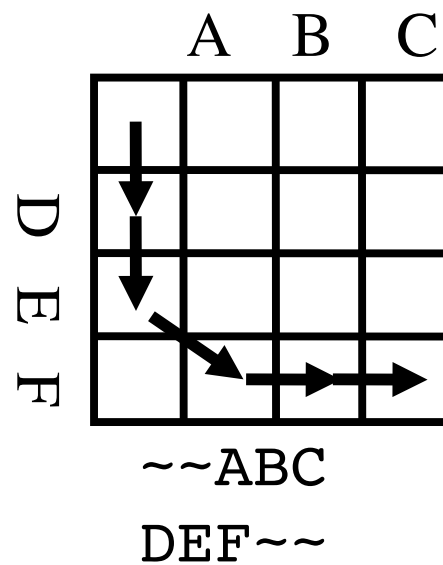
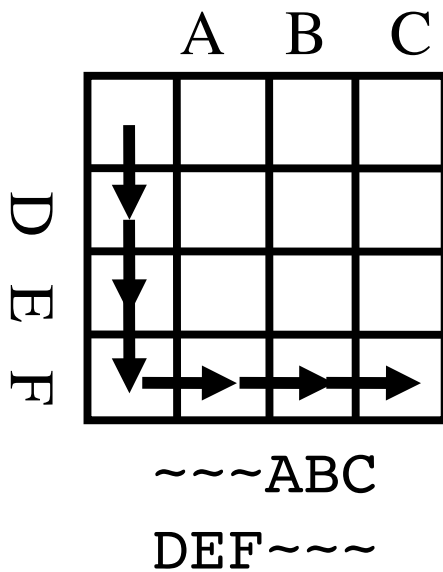
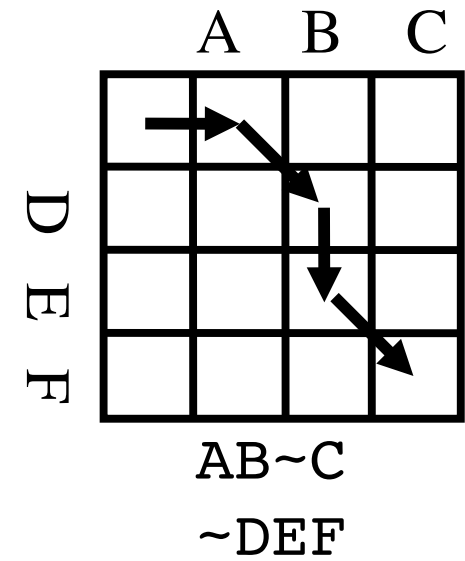
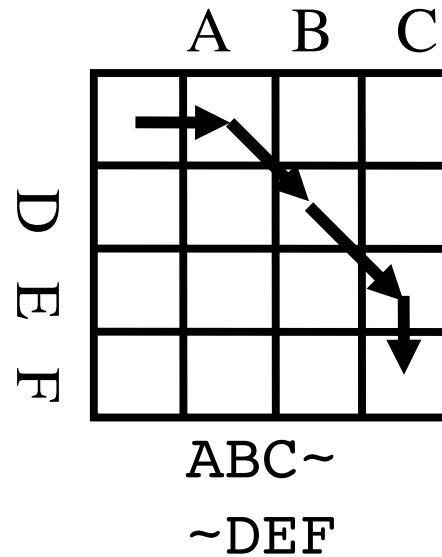
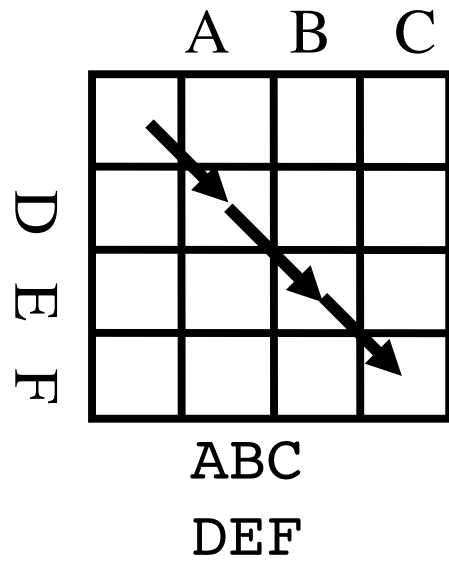
myfile contains precalculated alignment scores

walk is a series of arrows

add a penalty for gaps

add the alignment score to the score
only if the arrow was *diag*

A walk through the alignment matrix



All possible alignments for ABC versus DEF

ABC~~~

~~~DEF

ABC~~

~~DEF

ABC~

~DEF

ABC

DEF

~ABC

DEF~

~~ABC

DEF~~

AB~C

~DEF

A~BC

~DEF

A~BC

DE~F

A~BC

DEF~

AB~C

~DEF

AB~C

DEF~

A~~BC

DEF~~

ABC~~

~D~EF

A~B~C

~DEF~

A~~BC

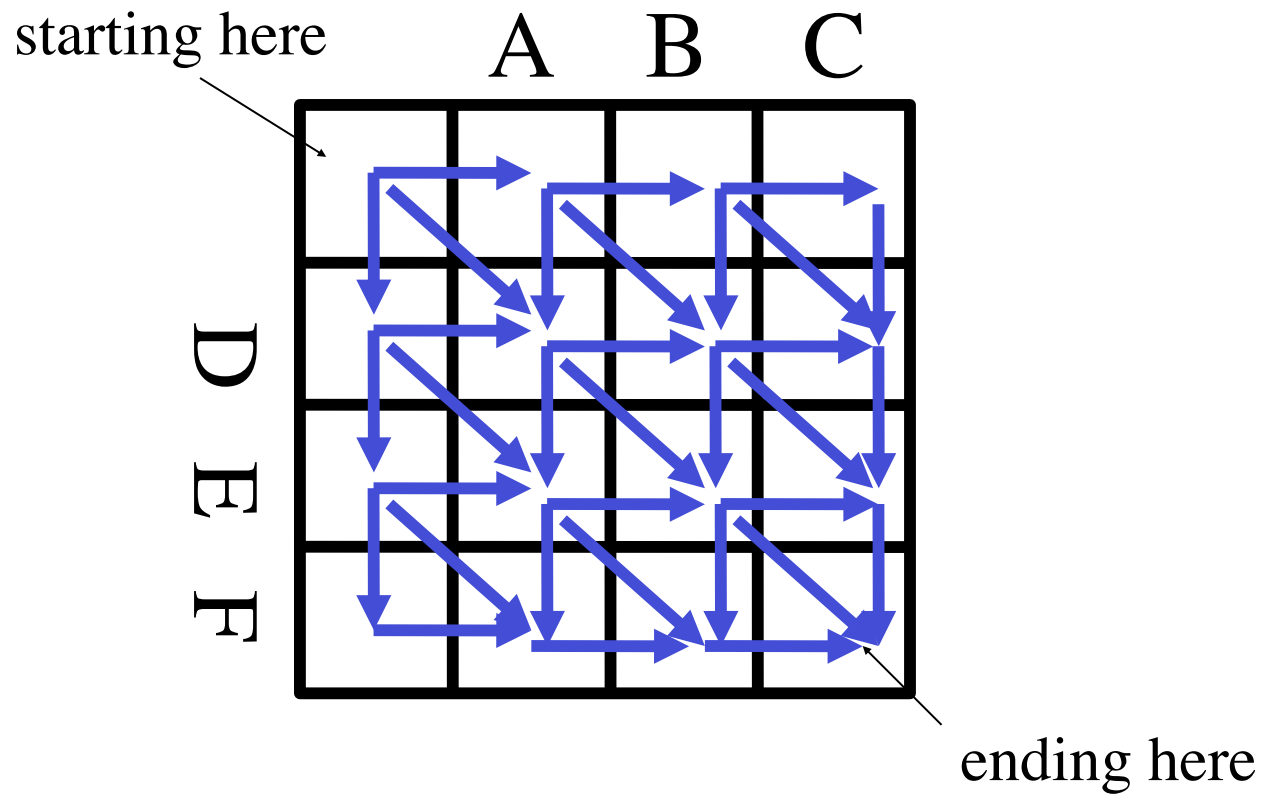
~DEF~

*no gaps*

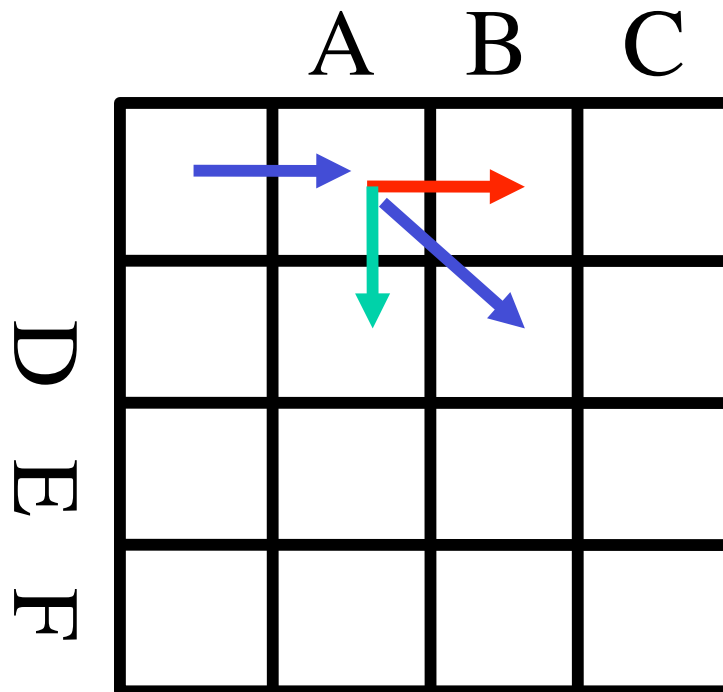
*one gaps*

*two gaps*

All possible alignments = all possible paths



Each box spawns three paths.



Total number of  
possible paths of  $n$   
arrows =  $3^n$

# Finding the optimal path

All "paths" through the alignment matrix end in the **lower right-hand corner**. Imagine there are  $3^n$  "walkers" each taking a different path to the lower right box and collecting scores and penalties as it goes. We can ask each walker as it enters the last box what its current score is. Then we pick the best one. Is that optimal? Did we need to ask all  $3^n$  walkers?

Consider the last *down* arrow before the last box. All alignments that traversed that arrow added the same amount -- a gap penalty. So the ranked scores of those walkers before the arrow are the same after the arrow. Whichever walker had the highest score in the previous box still has the highest score after all the walkers traverse the same arrow together. No change in ranking. So we could have ignored the suboptimal score walkers.

The same argument applies to all boxes back to the beginning. We only need to keep the optimal path. We can ignore the others, since they will always rank lower than the optimal.

# Dynamic Programming

- For each box, add up the scores of the three walks that end in that box. **Keep just the highest scoring one.**
- Draw an arrow (traceback) from the box that had the highest scoring walk.
- When you reach the last box, trace back along the arrows.
- Convert the arrows into an alignment. This is the optimal alignment.

# Dynamic programming algorithm

```
For all i=0..imax { For all j=0.. jmax {  
    Si,j = MAX { Si-1,j-1 + s(i,j), Si-1,j - wx, Si,j-1 - wy }  
    Ti,j = argmax { Si-1,j-1 + s(i,j), Si-1,j - wx, Si,j-1 - wy }  
} }
```

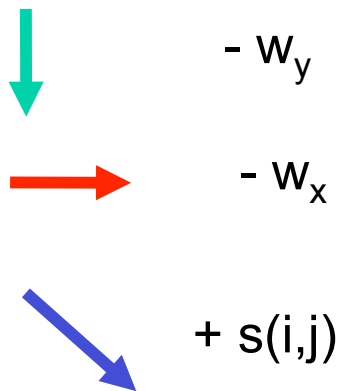
Traceback(T)

$S(i,j)$  is the score assigned to box  $(i,j)$ , composed to the maximum of the three values, the alignment score at  $(i-1,j-1)$  plus the match score  $s(i,j)$ , or the alignment score at  $(i-1,j)$  minus the gap penalty  $w_x$ , or the alignment score at  $(i,j-1)$  minus the gap penalty  $w_y$ . In practice  $w_x = w_y$ .

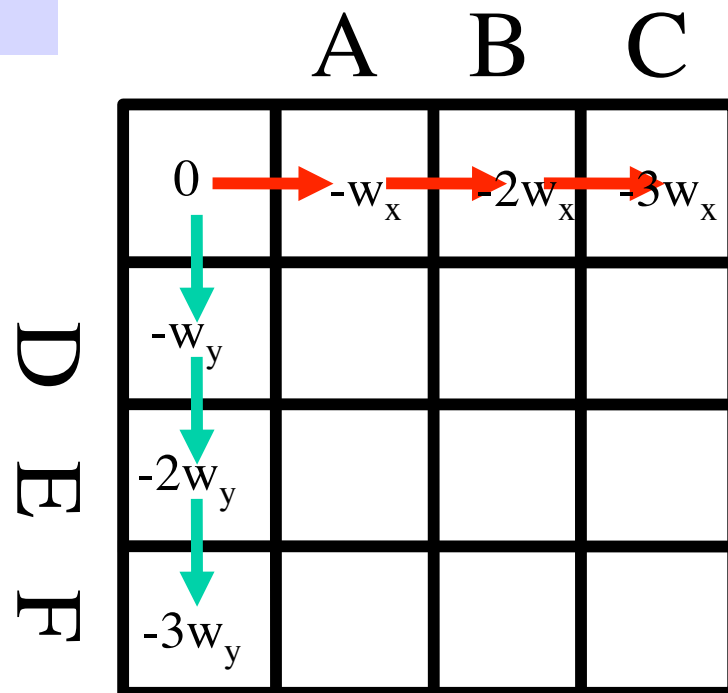
$T(i,j)$  is a flag pointing back along the arrow.

# Forward summation

$$S_{i,j} = \text{MAX} \left\{ \begin{array}{l} S_{i-1,j-1} + s(i,j), \\ S_{i-1,j} - w_x, \\ S_{i,j-1} - w_y \end{array} \right\}$$



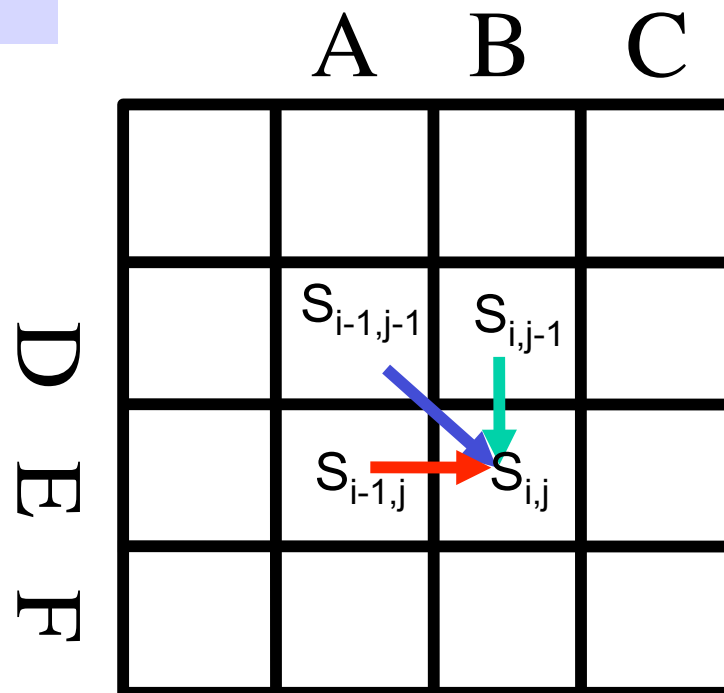
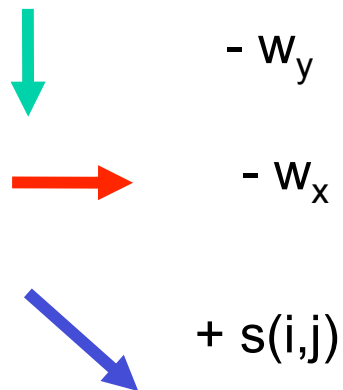
The first row and column (gap rows) are filled in using only one of the arrows, since the other two are out-of-bounds..



# Forward summation

$$S_{i,j} = \text{MAX} \left\{ \begin{array}{l} S_{i-1,j-1} + s(i,j), \\ S_{i-1,j} - w_x, \\ S_{i,j-1} - w_y \end{array} \right\}$$

Think of each arrow as adding a new term, either  $s(i,j)$  or a gap penalty.

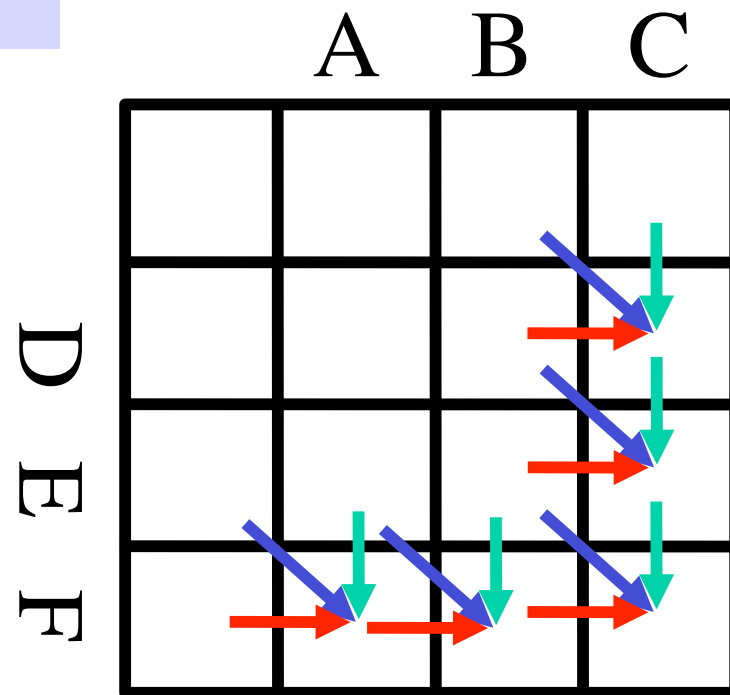
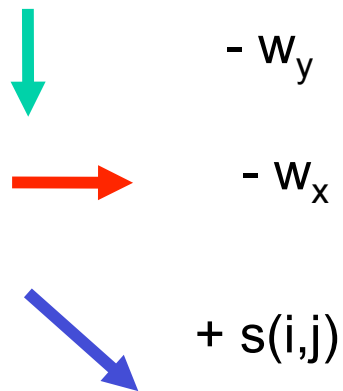


Filling in a box requires that the values in the three input boxes be already filled in.

# Forward summation

$$S_{i,j} = \text{MAX} \left\{ \begin{array}{l} S_{i-1,j-1} + s(i,j), \\ S_{i-1,j} - w_x, \\ S_{i,j-1} - w_y \end{array} \right\}$$

The last row and column are filled in normally

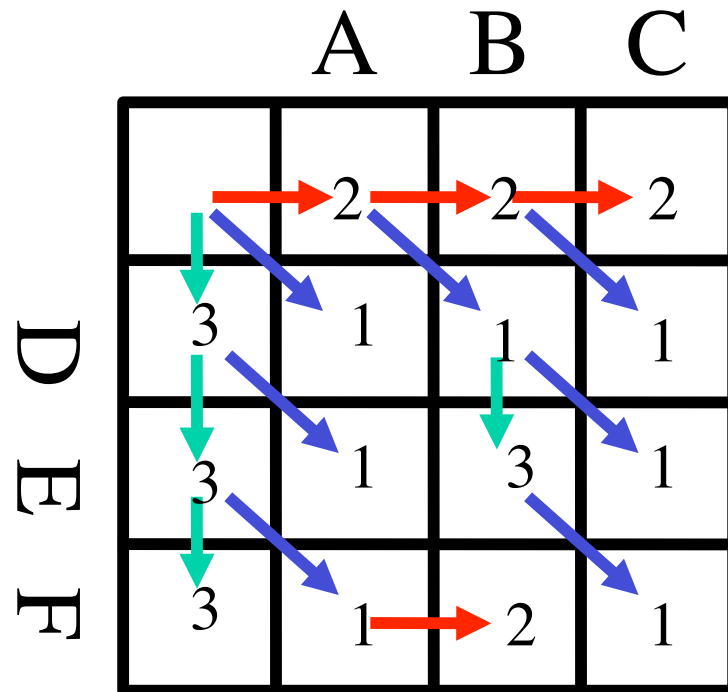
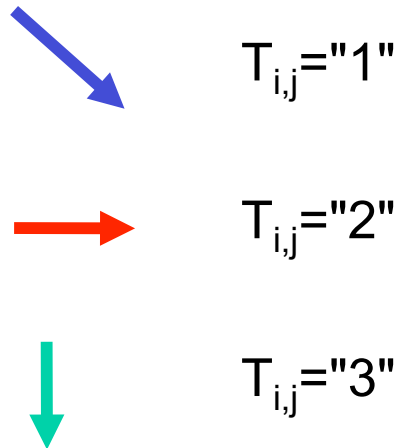


# Traceback

\*

$$T_{i,j} = \operatorname{argmax} \left\{ \begin{array}{l} S_{i-1,j-1} + s(i,j), \\ S_{i-1,j} - w_x, \\ S_{i,j-1} - w_y \end{array} \right\}$$

We save one *traceback* value in each box. This is a number, letter, or word that represents the arrow direction: i.e. **down**, **right**, or **diagonal**.



\* *argmax()* is a function that returns the *number* of the maximum argument, not the *value*.

# Traceback

$$T_{i,j} = \operatorname{argmax} \left\{ \begin{array}{l} S_{i-1,j-1} + s(i,j), \\ S_{i-1,j} - w_x, \\ S_{i,j-1} - w_y \end{array} \right\}$$

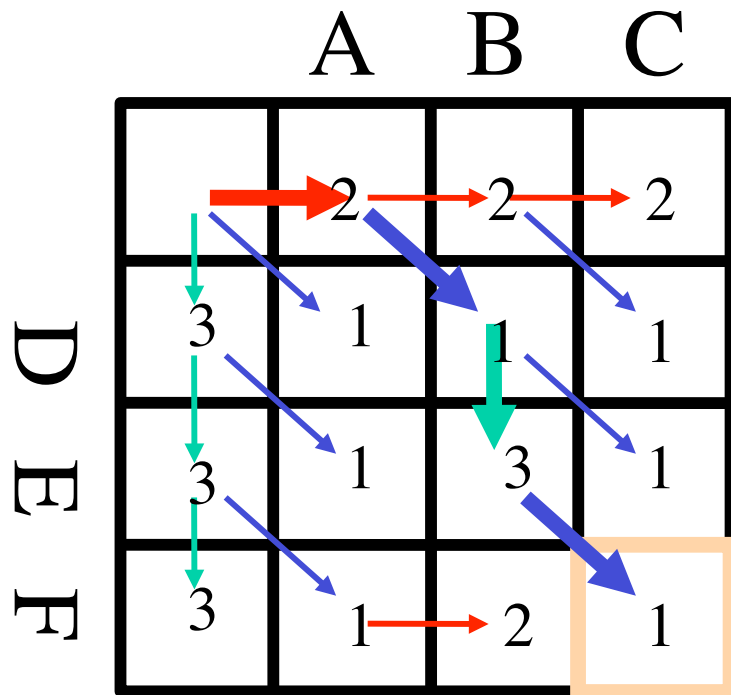
Traceback starts from the **last box** where  $i=\text{length of } x$ , and  $j=\text{length of } y$ .

Each "arrow" points back to the previous box. The result is a series of arrows in reverse order:

1312

These are reversed: 2131

...and translated to an alignment:



2 1 3 1  
  
 AB ~ C  
 ~ DEF

# From arrows to alignment.

Remember that each arrow is one "column" of the alignment.



Upper seq advances by one,  
Lower seq advances by zero.  
**Gap** in lower seq.



Upper seq advances by zero,  
Lower seq advances by one.  
**Gap** in upper seq.

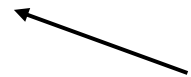


Upper seq advances by one,  
Lower seq advances by one.  
**Match.**

A  
~

~  
B

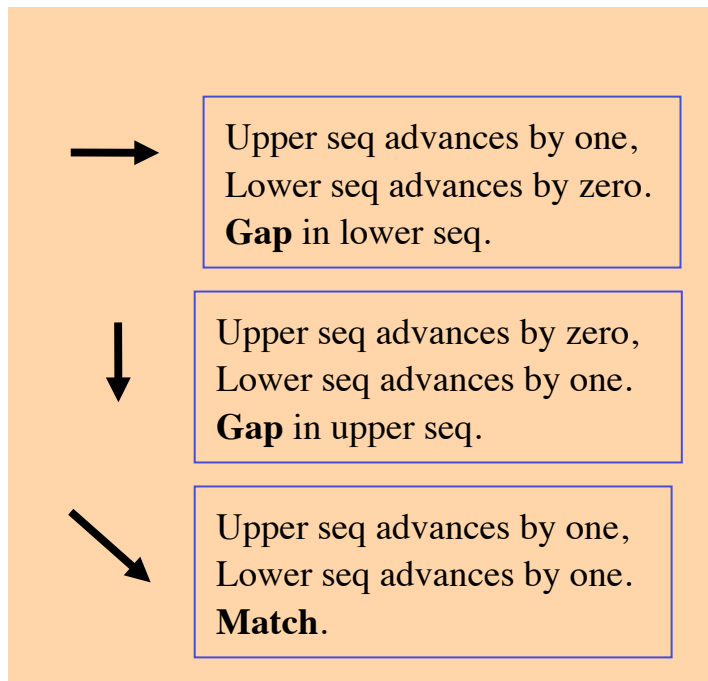
A  
B



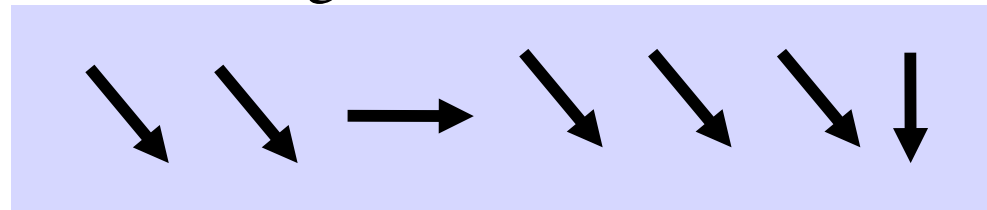
write the current sequence letter

# From arrows to alignment.

Remember that each arrow is one "column" of the alignment.



Align ADGTFR with  
ADTFRE using the  
following arrows:



|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| A | D | G | T | F | R | ~ |
| A | D | ~ | T | F | R | E |

## In class exercise: dynamic programming with proteins

Match=use BLOSUM score in lower right Gap penalty = -1

|   | A | D  | G  | T  | F  | R  | M  | G  | G  |    |
|---|---|----|----|----|----|----|----|----|----|----|
|   | 0 |    |    |    |    |    |    |    |    |    |
| D |   | -2 | 6  | -1 | -1 | -3 | -2 | -3 | -1 | -1 |
| G |   | 0  | -1 | 6  | -2 | -3 | -2 | -3 | 6  | 6  |
| Y |   | -2 | -3 | -3 | -2 | 3  | -2 | -1 | -3 | -3 |
| R |   | -1 | -2 | -2 | -1 | -3 | 5  | -1 | -2 | -2 |
| I |   | -2 | -3 | -4 | -1 | 0  | -3 | 1  | -4 | -4 |
| G |   | 6  | -1 | 6  | -2 | -3 | -2 | -3 | 6  | 6  |

# DP Instructions

- Prefill the boxes with the appropriate BLOSUM score.  
(*done. numbers in lower right.*)
- Calculate  $S_{i,j} = \max \{ S_{i-1,j-1} + s(i,j), S_{i-1,j} - 1, S_{i,j-1} - 1 \}$ . **Ignore arrows that are out-of-bounds.** Fill in boxes from upper left to lower right, as you read.
- As you fill in each  $S_{i,j}$  score, also draw one **arrow**( $\text{argmax} \{ S_{i-1,j-1} + s(i,j), S_{i-1,j} - 1, S_{i,j-1} - 1 \}$ ) coming into the box, where  $\text{arrow}(1) = \text{diagonal}$ ,  $\text{arrow}(2) = \text{right}$ ,  $\text{arrow}(3) = \text{down}$ .
- At the lower right, traceback and draw the arrows.
- Translate the arrows into an alignment.

# Alignment in Ugene

Enter the two short sequences into Geneious using File/New Document from text

Select the file names. Right click over one of the selected names and a menu pops up. Select Export/Export sequences as alignment. Give it a name. The sequences appear in an alignment, but the sequences are not aligned.

Right-click on the alignment. Select Align/Align with Kalign.

Set all gap penalties to 1. *Do you get the same answer as you did on paper?*