Melbourne Bioinformatics
BIOINFORMATICS + DATA SERVICES + INFRASTRUCTURE, FOR LIFE SCIENCES TODAY

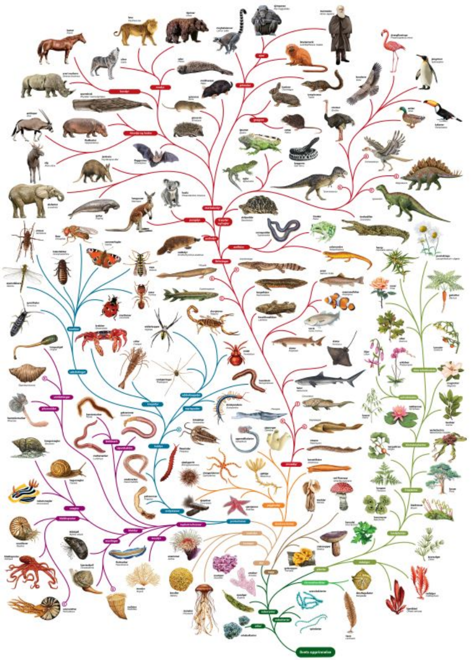
THE UNIVERSITY OF MELBOURNE

COMP90014

Algorithms for Bioinformatics

Week 2A - Sequence Alignment and Mapping I

Sequence Alignment and Mapping I



The Open University

Why align sequences?

1 Comparing sequences

2 Pairwise alignment

- Global alignment
- Local alignment
- Semi-global alignment

3 BLAST

sequence alignment（序列比对）是一种分析和比较两个或多个生物学序列（如DNA、RNA或蛋白质序列）的方法。这些序列可能来自不同物种、个体或同一物种中不同基因或蛋白质。

BLAST代表Basic Local Alignment Search Tool（基本局部比对搜索工具）。BLAST是一种广泛使用的生物信息学工具，用于比对和寻找生物学序列之间的相似性。

BLAST的主要目标是在一个给定的生物学数据库中快速找到一个查询序列的近似匹配或同源序列。该工具能够在数据库中高效地搜索相似的序列，如DNA、RNA或蛋白质序列

全局比对：全局比对试图找到两个序列之间的最佳匹配，从序列的一端到另一端进行比对。全局比对适用于相似度较高的序列，可以帮助研究相似的基因或蛋白质。

局部比对：局部比对仅关注两个序列中相似的部分，而忽略不相似的区域。这种比对适用于序列中存在插入或缺失的情况，例如寻找同源蛋白质中保守的结构域或模体。

Assignment 1

Released today (midnight)

Involves

- Indexing
- Kmers
- Sequence distance measurements
- Alignment

This lecture forms large portion of assignment questions

Why Align Sequences?

```
Before alignment

AFGIVHKLIVS
AFGIHKIVS

After alignment

AFGIVHKLIVS
AFGI-HK-IVS
```

Why align sequences?

Comparative analysis

Assess similarity

Similarity & Function

Discover functional and evolutionary relationships

-> Similar sequences suggest an evolutionary relationship

-> Evolutionary relationship suggests related function

-> Homology

Using biological sequences

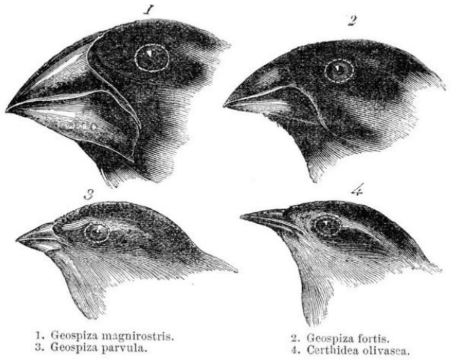
Compare organisms at molecular level

Find evolutionary relationships

Identify functionally conserved sequences

-> Infer function

-> Understand their evolution



Charles Darwin observed different species of finch in the Galápagos Islands

Wikipedia Commons

同源基因（Homologs）：指的是那些共享一个共同祖先的序列。这些序列可以是直系同源基因（orthologs）或旁系同源基因（paralogs）。

直系同源基因（Orthologs）：是由物种形成（speciation）事件分隔开的基因。例如，在不同物种中的基因，这些基因来源于同一个祖先基因。
旁系同源基因（Paralogs）：是由基因复制（duplication）事件分隔开的。比如，在一个物种中衍生自同一个基因复制事件的两个基因。

Why align sequences?

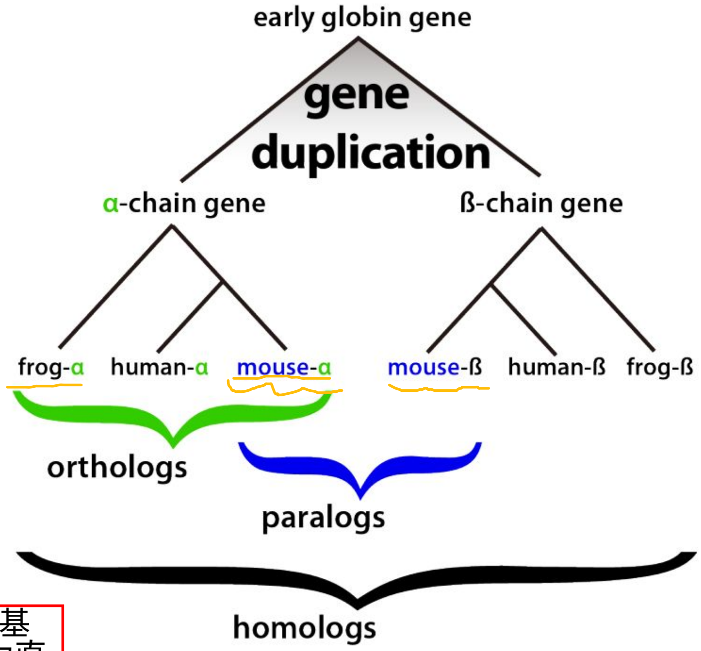
Homologs: Sequences that share a common ancestor.
Can be *orthologs* or *paralogs*

Orthologs: Separated by a speciation event.
e.g. genes in separate species derived from the same ancestral gene.

Paralogs: Separated by a duplication event.
e.g. two genes in a species derived from a gene duplication.

All orthologs and paralogs are homologs

Both can functionally diverge



Popo H. Liao via Wikimedia Commons

早期的珠蛋白基因（early globin gene）经历了基因复制，形成了α链基因和β链基因。在这个例子中，青蛙、人类、和老鼠的α链基因被标记为直系同源基因（因为它们来源于同一个祖先基因但存在于不同物种），而老鼠的α链和β链基因被标记为旁系同源基因（因为它们来源于同一个物种中的基因复制）。这些关系在图的下方通过不同颜色的曲线相互连接标示出来。

Why align sequences?

Bioinformatic uses of sequence alignments

DNA, RNA and protein sequences are strings

Processed to derive information

Alignment is a common starting point

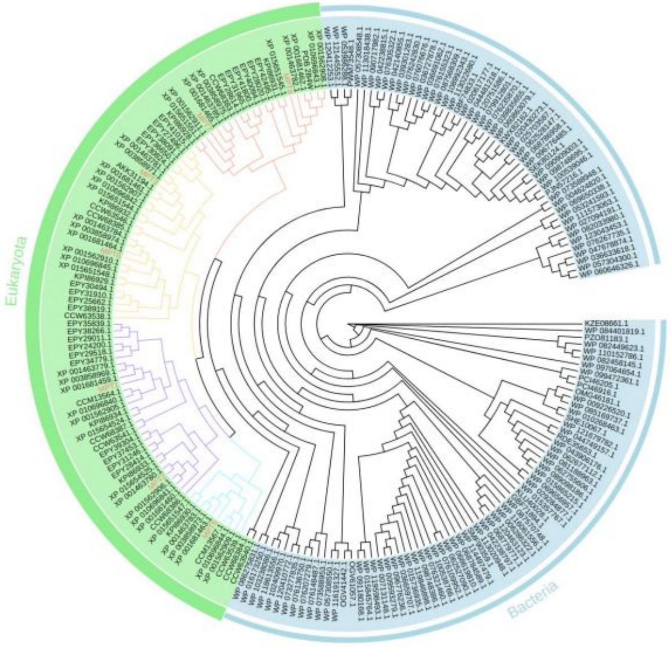
Used in different analyses

e.g. phylogenetic trees

Possible goals

Inference

Identification / mapping



Sernee et al. 2019. DOI: 10.1016/j.chom.2019.08.009

Why align sequences?

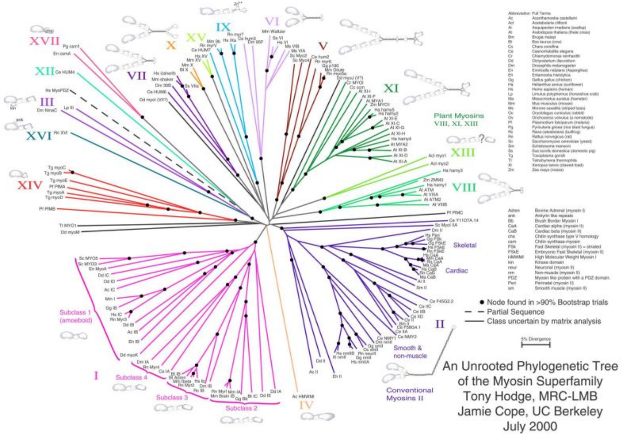
Bioinformatic uses of sequence alignments (ctd.)

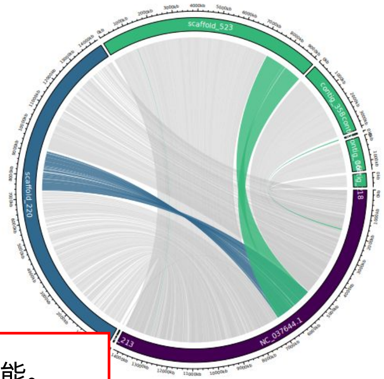
Phylogeny: Given a set of related (homologous) sequences, infer the evolutionary relationship between them.

Protein function: Given a newly identified sequence, find regions which are similar to proteins with known function, to infer similar structure and function.

Conservation: Given a set of related (homologous) sequences, find conserved regions.
e.g. regulatory elements in DNA
e.g. binding sites in proteins
e.g. find structural rearrangements in genomes.

系统发育学：通过比对一组相关（同源的）序列，推断它们之间的进化关系。
蛋白质功能：对于新识别的序列，找到与已知功能蛋白质相似的区域，从而推断其可能的结构和功能。
保守性：通过比对一组相关（同源的）序列，找出保守的区域。这些保守区域可能包括DNA中的调节元件、蛋白质中的结合位点或基因组中的结构重排。





Above
Phylogenetic tree of myosin superfamily

Left
Structural rearrangements in 3 related organisms

T. Hodge and M.J.T.V. Cope via Wikimedia Commons

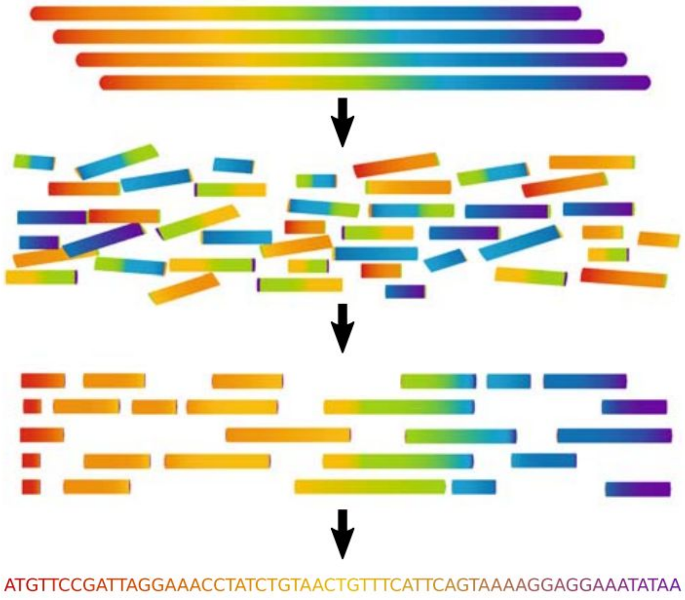
Why align sequences?

Bioinformatic uses of sequence alignments (ctd. ctd.)

Database searching: Given a query sequence, find similar sequences in a large database.
What species of bacteria are in my sample?

De novo assembly: Reconstruct a single large sequence from many small pieces of that sequence.
e.g. DNA, RNA.

Read Mapping: Given a reference sequence, align many small reads to it.
e.g. infer variants.



Commins et al., 2009. DOI: 10.1007/s12575-009-9004-1

Comparing Sequences

Comparing Sequences

Depends on your goal!

Sequences are strings - we use string distance algorithms

eg. DNA:	`"TGAGATTACA"`
eg. Protein:	`"LVCGERGFFY"`

Bioinformatics requires special variants

Type	Examples
Small to small	Comparing homologs Different types of substitutions / indels?
Small to big	Find gene in a genome Can’t align to whole genome?
Big to big	Comparing two genomes Handling structural variation?
Special cases	Aligning RNAseq reads to genome Handling split alignments?

pollev.com/gracehall381

Comparing Sequences

Only substitutions

Hamming Distance

For sequences of same length, count the number of mismatches at each position

Hamming distance between seq1 & seq2?

Hamming distance between seq1 & seq3?

```
Seq1: AGTAGCACTGA

Seq2: AGTAACACTGA

Seq3: AGTAGCCTGAT
           ▲    ▲
          del  ins
```

Comparing Sequences

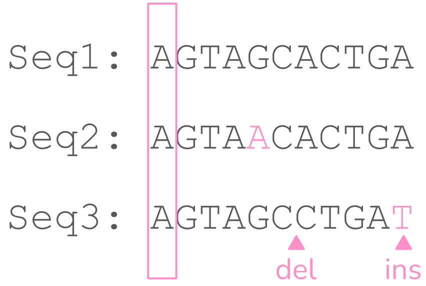
Only substitutions

Hamming Distance

For sequences of same length, count the number of mismatches at each position

Hamming distance between seq1 & seq2?

Hamming distance between seq1 & seq3?



```
Seq1: AGTAGCACTGA

Seq2: AGTAACACTGA

Seq3: AGTAGCCTGAT
```

del

ins

Comparison	Distance
Seq1 vs Seq2	0
Seq1 vs Seq3	0

Comparing Sequences

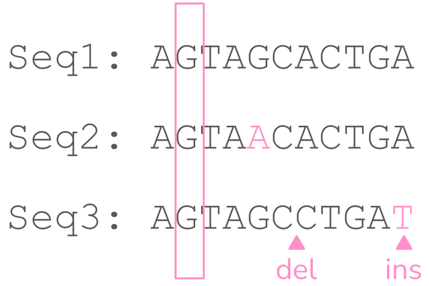
Only substitutions

Hamming Distance

For sequences of same length, count the number of mismatches at each position

Hamming distance between seq1 & seq2?

Hamming distance between seq1 & seq3?



```
Seq1: AGTAGCACTGA
Seq2: AGTAACACTGA
Seq3: AGTAGCCTGAT
```

Comparison	Distance
Seq1 vs Seq2	0
Seq1 vs Seq3	0

Comparing Sequences

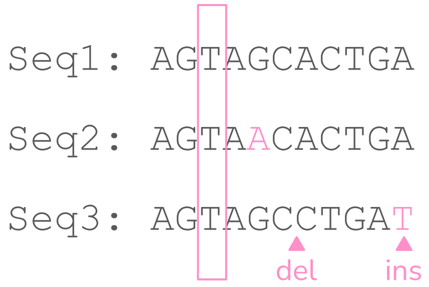
Only substitutions

Hamming Distance

For sequences of same length, count the number of mismatches at each position

Hamming distance between seq1 & seq2?

Hamming distance between seq1 & seq3?



```
Seq1: AGTAGCACTGA

Seq2: AGTAACACTGA

Seq3: AGTAGCCTGAT
```

Comparison	Distance
Seq1 vs Seq2	0
Seq1 vs Seq3	0

Comparing Sequences

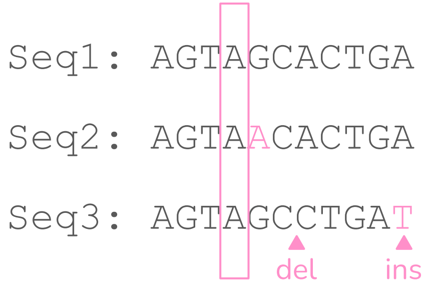
Only substitutions

Hamming Distance

For sequences of same length, count the number of mismatches at each position

Hamming distance between seq1 & seq2?

Hamming distance between seq1 & seq3?



```
Seq1: AGTAGCACTGA
Seq2: AGTAACACTGA
Seq3: AGTAGCCTGAT
```

Comparison	Distance
Seq1 vs Seq2	0
Seq1 vs Seq3	0

Comparing Sequences

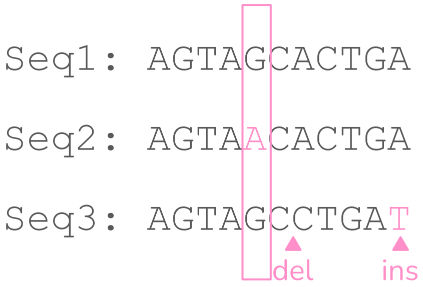
Only substitutions

Hamming Distance

For sequences of same length, count the number of mismatches at each position

Hamming distance between seq1 & seq2?

Hamming distance between seq1 & seq3?



```
Seq1: AGTAGCACTGA

Seq2: AGTAACACTGA

Seq3: AGTAGCCTGAT
```

Comparison	Distance
Seq1 vs Seq2	1
Seq1 vs Seq3	0

Comparing Sequences

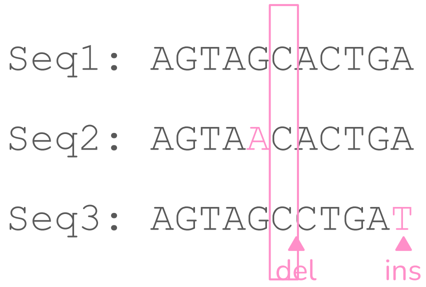
Only substitutions

Hamming Distance

For sequences of same length, count the number of mismatches at each position

Hamming distance between seq1 & seq2?

Hamming distance between seq1 & seq3?



Seq1: AGTAGCACTGA

Seq2: AGTAACACTGA

Seq3: AGTAGCCTGAT

Comparison	Distance
Seq1 vs Seq2	1
Seq1 vs Seq3	0

Comparing Sequences

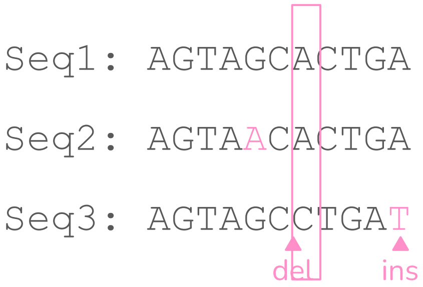
Only substitutions

Hamming Distance

For sequences of same length, count the number of mismatches at each position

Hamming distance between seq1 & seq2?

Hamming distance between seq1 & seq3?



```
Seq1: AGTAGCACTGA
Seq2: AGTAACACTGA
Seq3: AGTAGCCTGAT
```

Comparison	Distance
Seq1 vs Seq2	1
Seq1 vs Seq3	1

Comparing Sequences

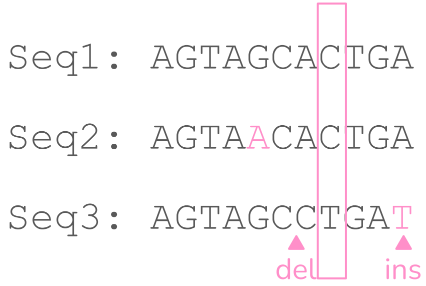
Only substitutions

Hamming Distance

For sequences of same length, count the number of mismatches at each position

Hamming distance between seq1 & seq2?

Hamming distance between seq1 & seq3?



```
Seq1: AGTAGCACTGA

Seq2: AGTAACACTGA

Seq3: AGTAGCCTGAT
```

Comparison	Distance
Seq1 vs Seq2	1
Seq1 vs Seq3	2

Comparing Sequences

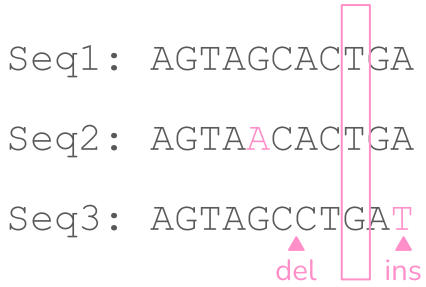
Only substitutions

Hamming Distance

For sequences of same length, count the number of mismatches at each position

Hamming distance between seq1 & seq2?

Hamming distance between seq1 & seq3?



```
Seq1: AGTAGCACTGA

Seq2: AGTAACACTGA

Seq3: AGTAGCCTGAT
```

Comparison	Distance
Seq1 vs Seq2	1
Seq1 vs Seq3	3

Comparing Sequences

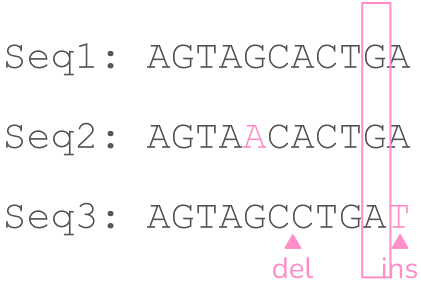
Only substitutions

Hamming Distance

For sequences of same length, count the number of mismatches at each position

Hamming distance between seq1 & seq2?

Hamming distance between seq1 & seq3?



```
Seq1: AGTAGCACTGA
Seq2: AGTAACACTGA
Seq3: AGTAGCCTGAT
```

Comparison	Distance
Seq1 vs Seq2	1
Seq1 vs Seq3	4

Comparing Sequences

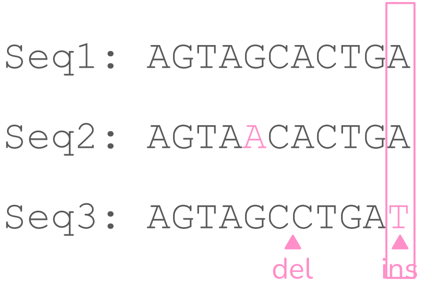
Only substitutions

Hamming Distance

For sequences of same length, count the number of mismatches at each position

Hamming distance between seq1 & seq2?

Hamming distance between seq1 & seq3?



```
Seq1: AGTAGCACTGA
Seq2: AGTAACACTGA
Seq3: AGTAGCCTGAT
```

Comparison	Distance
Seq1 vs Seq2	1
Seq1 vs Seq3	5

Comparing Sequences

Only substitutions

Hamming Distance

For sequences of same length, count the number of mismatches at each position

Hamming distance between seq1 & seq2?

Hamming distance between seq1 & seq3?

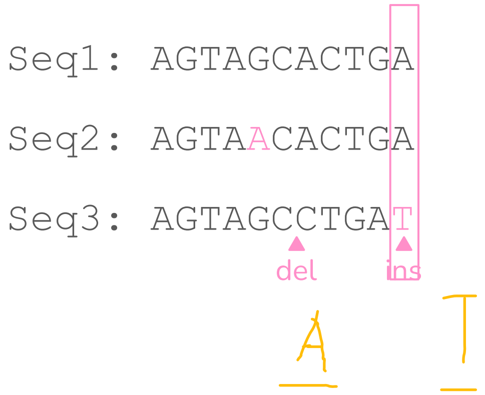
Hamming distance

Fast! Best / average / worst complexity: *O(n)*

Includes position information, but doesn't handle indels well.
(seq1 and seq3 differ by 2 mutations, not 5!)

Common form of variation!

无法处理insert和delete



```
Seq1: AGTAGCACTGA
Seq2: AGTAACACTGA
Seq3: AGTAGCCTGAT
```

Comparison	Distance
Seq1 vs Seq2	1
Seq1 vs Seq3	5

Comparing Sequences

Can we do better?

How could we compare these sequences regardless of position?

```
Seq1: AGTAGCACTGA

Seq2: AGTAACACTGA

Seq3: AGTAGCCTGAT
            ▲   ▲
           del ins
```

Comparing Sequences

Can we do better?

How could we compare these sequences regardless of position?

Seq1: AGTAGCACTGA

Seq2: AGTAACACTGA

Seq3: AGTAGCCTGAT

del ins

Sequence	Kmers
Seq1	AGT, GTA, TAG, AGC, GCA, CAC, ACT, CTG, TGA
Seq2	AGT, GTA, **TAA, AAC, ACA**, CAC, ACT, CTG, TGA
Seq3	AGT, GTA, TAG, AGC, **GCC, CCT**, CTG, TGA, **GAT**

Comparison	Distance
Seq1 vs Seq2	3
Seq1 vs Seq3	3

Comparing Sequences

Can we do better?

How could we compare these sequences regardless of position?

Kmer distance

Fast! Best / average / worst complexity: $O(n)$

Often used as preprocessing step (reduce search space)

```
Seq1: AGTAGCACTGA

Seq2: AGTAACACTGA

Seq3: AGTAGCCTGAT
            ▲    ▲
           del  ins
```

Sequence	Kmers
Seq1	AGT, GTA, TAG, AGC, GCA, CAC, ACT, CTG, TGA
Seq2	AGT, GTA, **TAA, AAC, ACA**, CAC, ACT, CTG, TGA
Seq3	AGT, GTA, TAG, AGC, **GCC, CCT**, CTG, TGA, **GAT**

Comparison	Distance
Seq1 vs Seq2	3
Seq1 vs Seq3	3

较大的k值（kmer的长度）意味着更高的特异性和较少的匹配次数。因为较长的序列在随机组合下出现的几率更小，所以找到匹配的可能性也更低。

较小的k值则意味着更低的特异性和更多的匹配次数。较短的序列在随机组合中出现的几率更大，所以会有更多的匹配。

Comparing Sequences

When working with kmers

Careful with size of ***k***

-> Larger k: more specific, less matches

-> Smaller k: less specific, more matches

size of k	combinations	examples
1	4	A, T, G or C
2	16	AA, AT, AG, AC, TA...
16*	~4 billion	

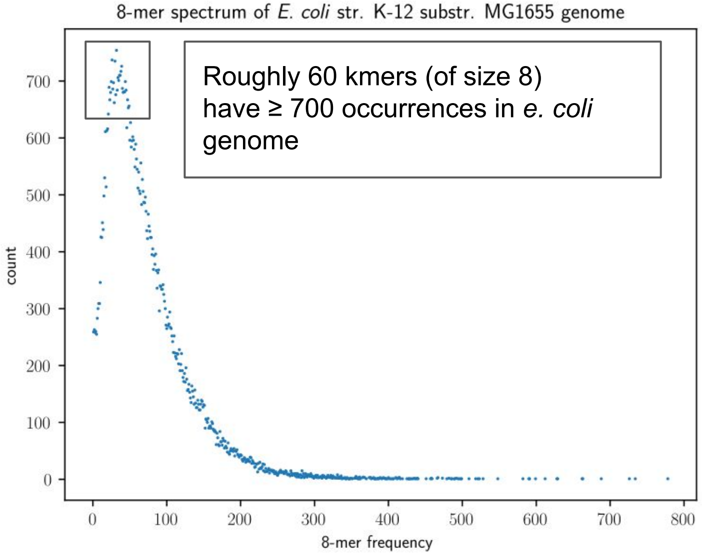
*common for read alignment

Genomes have repetitive DNA

-> For example, using k=16:

-> Expect to see some kmers many times

-> Expect to see some kmers only 1 time



Roughly 60 kmers (of size 8) have ≥ 700 occurrences in *e. coli* genome

Ytngargar via wikipedia commons

Comparing Sequences

Allowing substitutions + indels

-> There is 1 variant between seq1 & seq2

-> There are 2 variants between seq1 & seq3

Is there an algorithm which will give us a better distance metric?

```
Seq1: AGTAGCACTGA

Seq2: AGTAACACTGA

Seq3: AGTAGCCTGAT
            ▲   ▲
           del ins
```

Comparing Sequences

Allowing substitutions + indels

-> There is 1 variant between seq1 & seq2

-> There are 2 variants between seq1 & seq3

Is there an algorithm which will give us a better distance metric?

Need to consider that for a given letter:

- May have a match/mismatch
- May have inserted a letter
- May have deleted a letter

Now you’re thinking with portals.

```
Seq1: AGTAGCACTGA

Seq2: AGTAACACTGA

Seq3: AGTAGCCTGAT
            ▲   ▲
           del ins
```

Comparing Sequences

Allowing substitutions + indels

-> There is 1 variant between seq1 & seq2

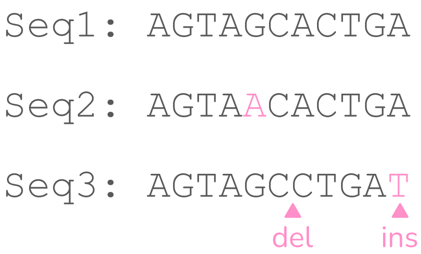
-> There are 2 variants between seq1 & seq3

Is there an algorithm which will give us a better distance metric?

Need to consider that for a given letter:

- May have a match/mismatch
- May have inserted a letter
- May have deleted a letter

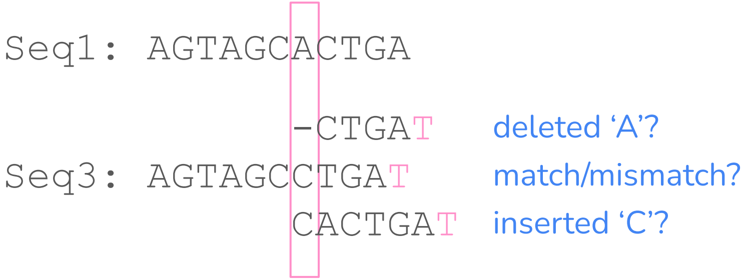
Now you're thinking with portals.



Seq1: AGTAGCACTGA

Seq2: AGTAACACTGA

Seq3: AGTAGCCTGAT



Seq1: AGTAGCACTGA

-CTGAT deleted 'A'?

Seq3: AGTAGCCTGAT match/mismatch?

CACTGAT inserted 'C'?

Comparing Sequences

Levenshtein distance (edit distance)

“Minimum number of transformations to go from one string to another”

Count mismatches, insertions and deletions as transformations

Want to find minimum number of these edits to transform str1 -> str2

Applies nicely to genomics as transformations are genomic mutation!

Allows sequences of different lengths

```
SATURDAY -> SUNDAY

SATURDAY
SUNDAY

EDITS: 0
```

Comparing Sequences

Levenshtein distance (edit distance)

“Minimum number of transformations to go from one string to another”

Count mismatches, insertions and deletions as transformations

Want to find minimum number of these edits to transform str1 -> str2

Applies nicely to genomics as transformations are genomic mutation!

Allows sequences of different lengths

```
SATURDAY -> SUNDAY

SATURDAY
SUNDAY

EDITS: 0
```

Comparing Sequences

Levenshtein distance (edit distance)

“Minimum number of transformations to go from one string to another”

Count mismatches, insertions and deletions as transformations

Want to find minimum number of these edits to transform str1 -> str2

Applies nicely to genomics as transformations are genomic mutation!

Allows sequences of different lengths

```
SATURDAY -> SUNDAY

SATURDAY
SUNDAY

EDITS: 0
```

Comparing Sequences

Levenshtein distance (edit distance)

“Minimum number of transformations to go from one string to another”

Count mismatches, insertions and deletions as transformations

Want to find minimum number of these edits to transform str1 -> str2

Applies nicely to genomics as transformations are genomic mutation!

Allows sequences of different lengths

```
SATURDAY -> SUNDAY

SATURDAY
SUNDAY

EDITS: 0
```

Comparing Sequences

Levenshtein distance (edit distance)

“Minimum number of transformations to go from one string to another”

Count mismatches, insertions and deletions as transformations

Want to find minimum number of these edits to transform str1 -> str2

Applies nicely to genomics as transformations are genomic mutation!

Allows sequences of different lengths

```
SATURDAY -> SUNDAY

SATURDAY
S--UNDAY

EDITS: 3
```

Comparing Sequences

Levenshtein distance (edit distance)

- "Minimum number of transformations to go from one string to another"
- Count mismatches, insertions and deletions as transformations
- Want to find minimum number of these edits to transform str1 -> str2
- Applies nicely to genomics as transformations are genomic mutation!
- Allows sequences of different lengths

How can we compute this?

```
SATURDAY -> SUNDAY

SATURDAY
S--UNDAY

EDITS: 3
```

```
RELEVANT -> ELEPHANT

(hamming)    (levenshtein)
RELEVANT     RELEV-ANT
ELEPHANT     -ELEPHANT

EDITS: 5     EDITS: 3
```

Comparing Sequences

Levenshtein distance

"Minimum number of transformations"

Penalize mismatches and gaps (+1)

Use 2D grid (allows insertions / deletions)

Seq A

Seq B

	start	G	C	A	C	T	G	A
start								
G								
del ▶ C								
C								
T								
G								
A								
ins ▶ T								

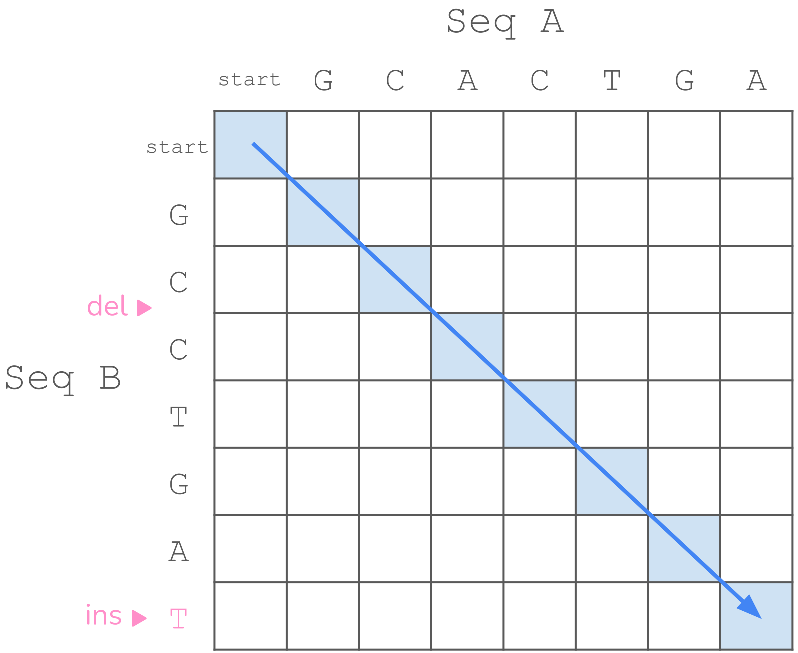
Comparing Sequences

Levenshtein distance

“Minimum number of transformations”

Penalize mismatches and gaps (+1)

Use 2D grid (allows insertions / deletions)



What we were doing before (hamming)

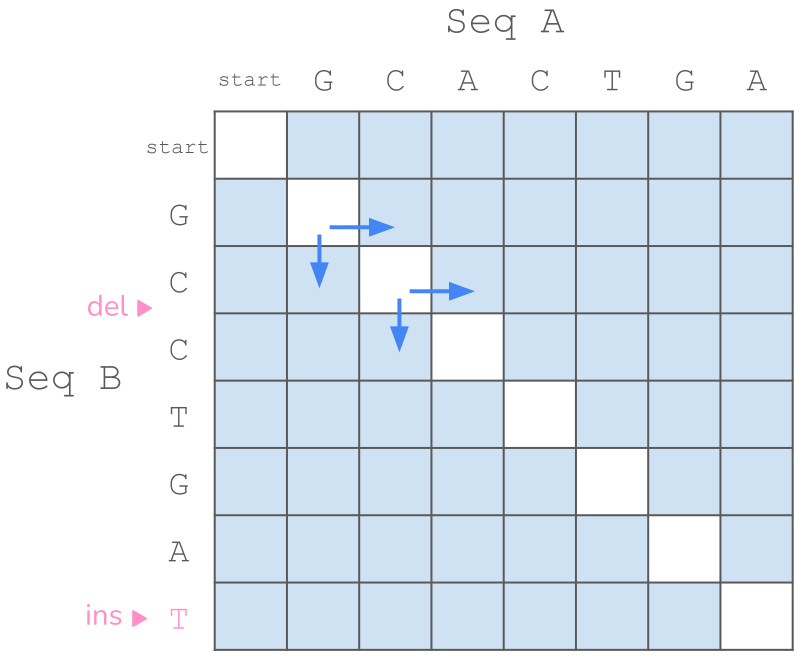
Comparing Sequences

Levenshtein distance

“Minimum number of transformations”

Penalize mismatches and gaps (+1)

Use 2D grid (allows insertions / deletions)



Now we allow shifts

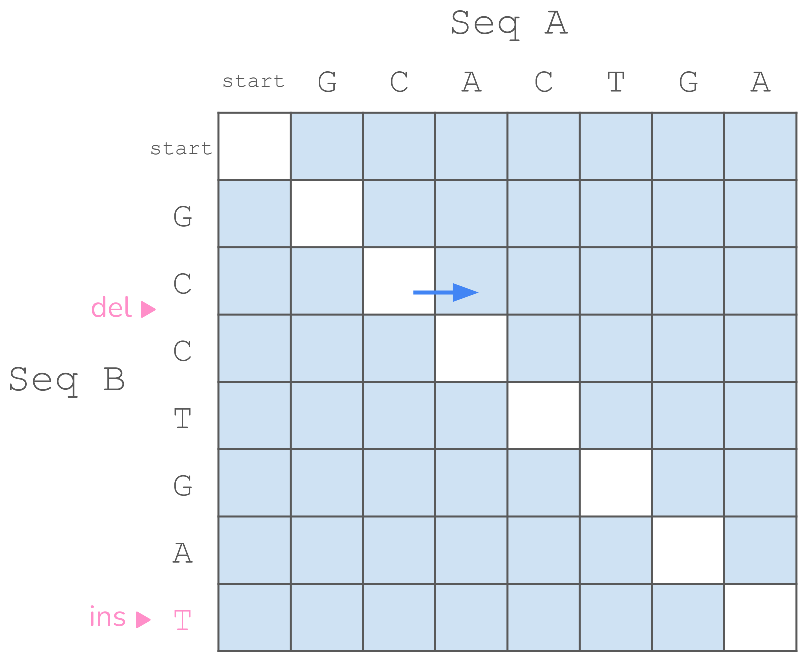
Comparing Sequences

Levenshtein distance

“Minimum number of transformations”

Penalize mismatches and gaps (+1)

Use 2D grid (allows insertions / deletions)



What is this representing?

Staying on ‘C’ for Seq B, but moving 1 position fwd in Seq A

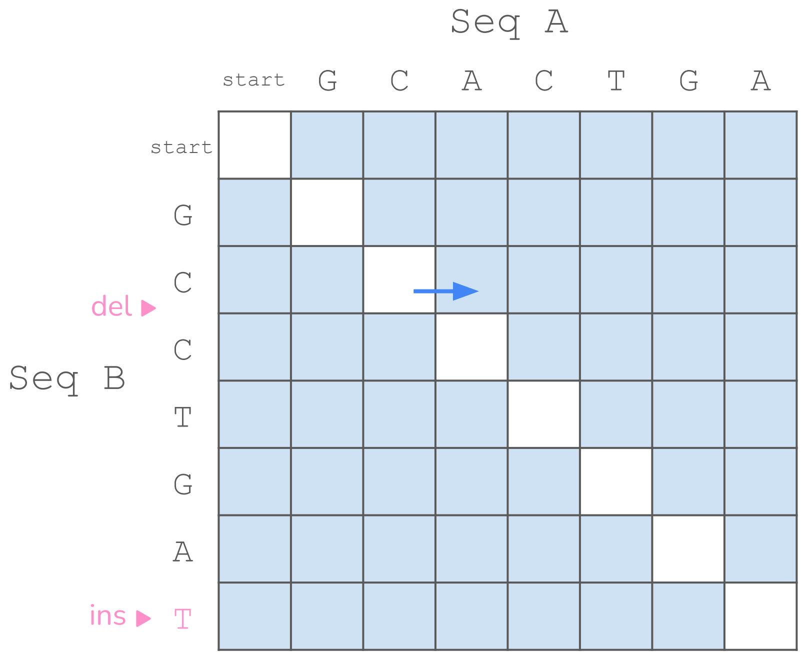
Comparing Sequences

Levenshtein distance

“Minimum number of transformations”

Penalize mismatches and gaps (+1)

Use 2D grid (allows insertions / deletions)



What is this representing?

Staying on ‘C’ for Seq B, but moving 1 position fwd in Seq A

Deletion in Seq B

Comparing Sequences

```
SeqA: GCACTGA-
SeqB: GC-CTGAT
```

Levenshtein distance

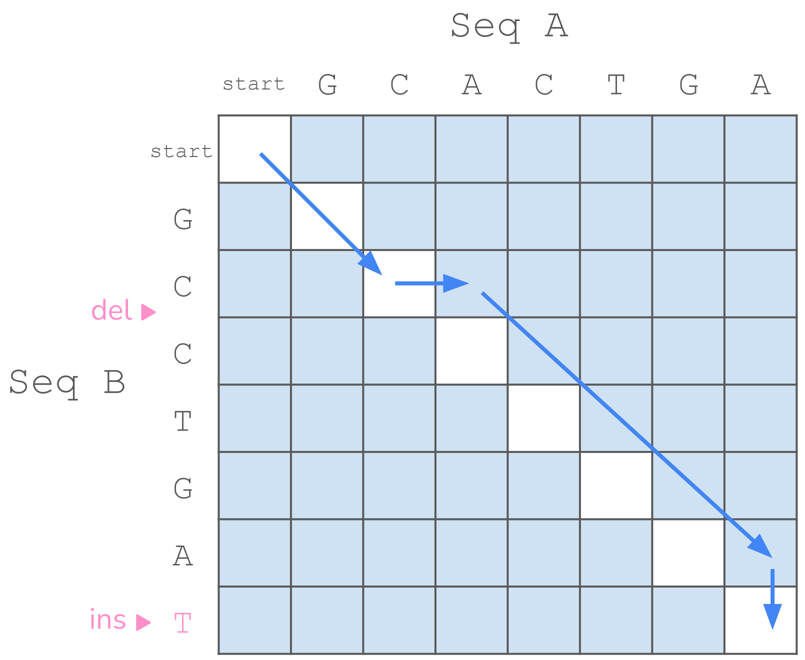
“Minimum number of transformations”

Penalize mismatches and gaps (+1)

Use 2D grid (allows insertions / deletions)

对于B来说
向A的方向shift一次 就是相对于A在B中发生了一次Del(Deletion in SeqB)

向B的方向shift一次 就是相对于A在B中发生了一次Ins(Insertion in SeqA)



This route gives us the best alignment

(Represents del at pos 3, and ins at pos 7)

Comparing Sequences

Levenshtein distance

“Minimum number of transformations”

Penalize mismatches and gaps (+1)

Use 2D grid (allows insertions / deletions)

Seq A

		start	G	C	A	C	T	G	A
	start								
	G								
del ▶	C								
Seq B	C								
	T								
	G								
	A								
ins ▶	T								

Comparing Sequences

Levenshtein distance

“Minimum number of transformations”

Penalize mismatches and gaps (+1)

Use 2D grid (allows insertions / deletions)

Preprocessing 1

Add gap costs to top row

Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G								
del ▶	C								
	C								
	T								
	G								
	A								
ins ▶	T								

Comparing Sequences

```
SeqA:  GCACTGA
SeqB:  GCCTGAT
```

Levenshtein distance

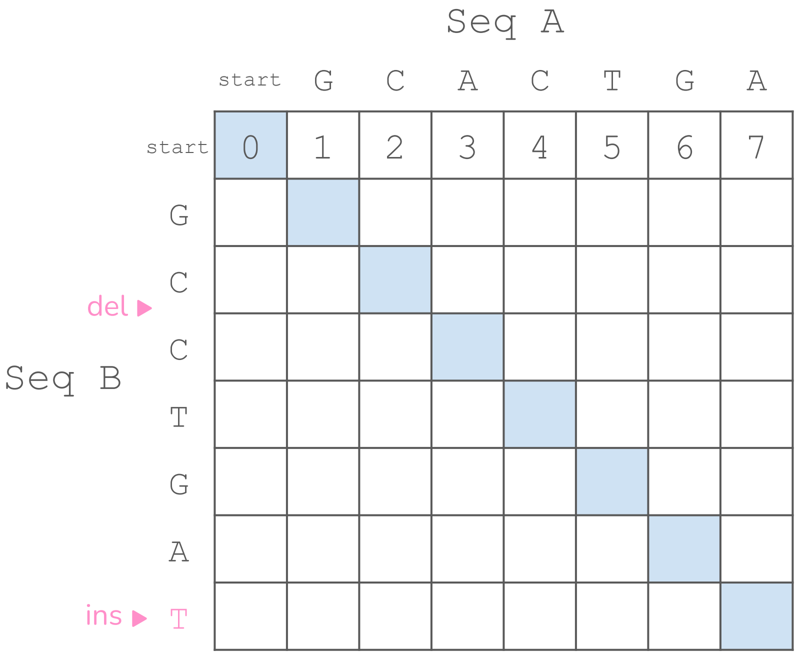
“Minimum number of transformations”

Penalize mismatches and gaps (+1)

Use 2D grid (allows insertions / deletions)

Preprocessing 1

Add gap costs to top row



	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G								
C								
C								
T								
G								
A								
T								

Comparing Sequences

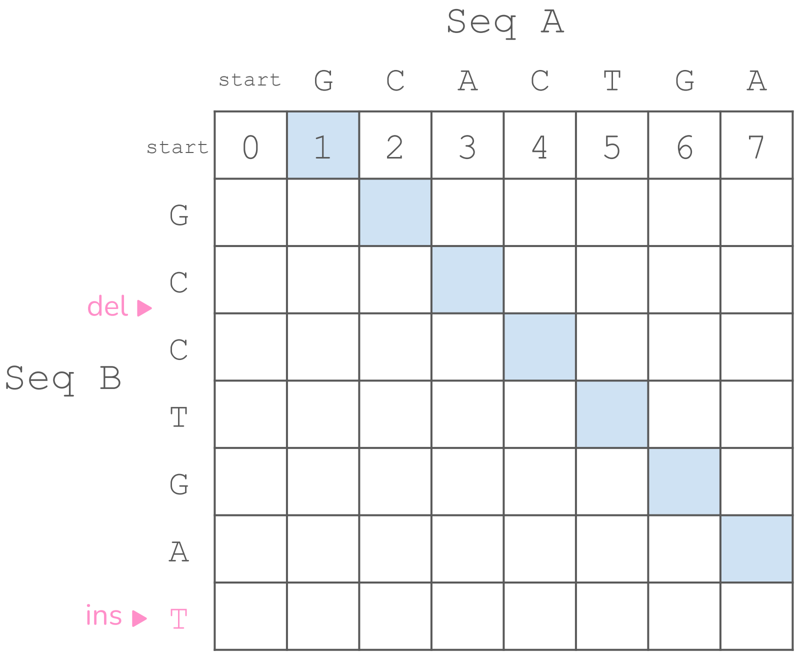
```
SeqA: GCACTGA
SeqB: -GCCTGAT
```

Levenshtein distance

- “Minimum number of transformations”
- Penalize mismatches and gaps (+1)
- Use 2D grid (allows insertions / deletions)

Preprocessing 1

- Add gap costs to top row



Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G			▨					
	C				▨				
del ▶	C					▨			
	T						▨		
	G							▨	
	A								▨
ins ▶	T								

Comparing Sequences

```
SeqA: GCACTGA
SeqB: --GCCTGAT
```

Levenshtein distance

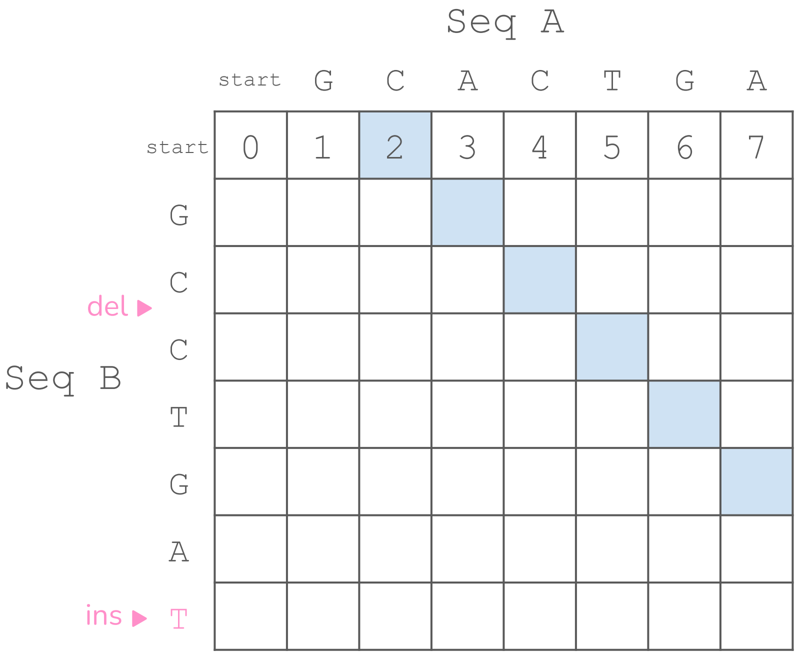
“Minimum number of transformations”

Penalize mismatches and gaps (+1)

Use 2D grid (allows insertions / deletions)

Preprocessing 1

Add gap costs to top row



	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G								
C								
C								
T								
G								
A								
T								

Comparing Sequences

Levenshtein distance

“Minimum number of transformations”

Penalize mismatches and gaps (+1)

Use 2D grid (allows insertions / deletions)

Preprocessing 1

Add gap costs to top row

Seq A

Seq B

	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G								
C								
del ▶ C								
T								
G								
A								
ins ▶ T								

Comparing Sequences

Levenshtein distance

“Minimum number of transformations”

Penalize mismatches and gaps (+1)

Use 2D grid (allows insertions / deletions)

Preprocessing 1

Add gap costs to top row

Preprocessing 2

Add gap costs to left column

Seq A

		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G	1							
	C	2							
del ▶	C	3							
Seq B	T	4							
	G	5							
	A	6							
ins ▶	T	7							

Comparing Sequences

```
SeqA: GCACTGA
SeqB: GCCTGAT
```

Levenshtein distance

“Minimum number of transformations”

Penalize mismatches and gaps (+1)

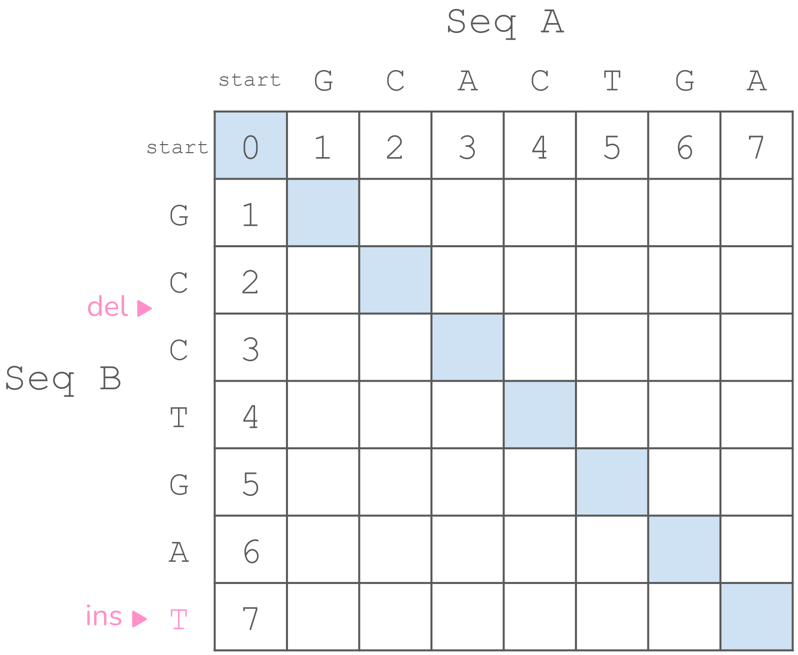
Use 2D grid (allows insertions / deletions)

Preprocessing 1

Add gap costs to top row

Preprocessing 2

Add gap costs to left column



	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G	1							
C	2							
C	3							
T	4							
G	5							
A	6							
T	7							

Comparing Sequences

```
SeqA: -GCACTGA
SeqB: GCCTGAT
```

Levenshtein distance

“Minimum number of transformations”

Penalize mismatches and gaps (+1)

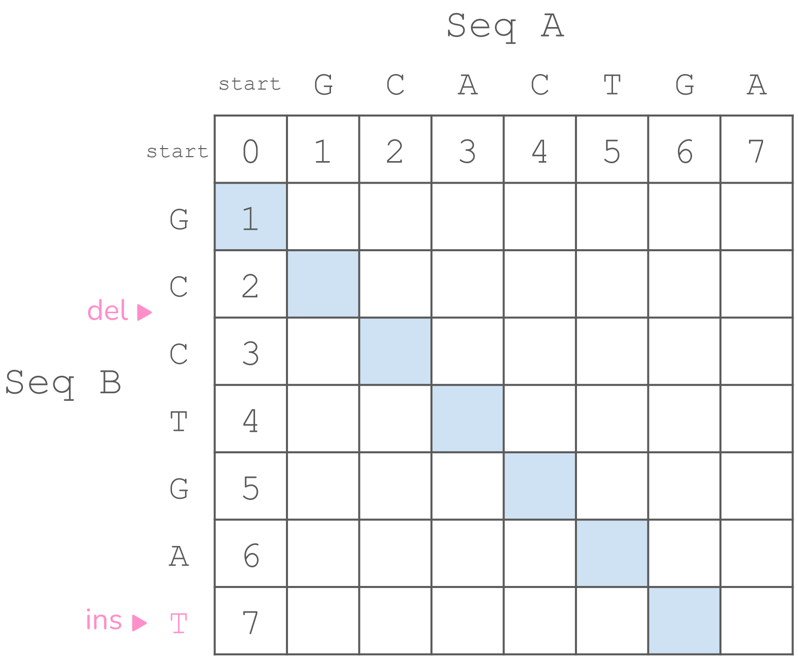
Use 2D grid (allows insertions / deletions)

Preprocessing 1

Add gap costs to top row

Preprocessing 2

Add gap costs to left column



	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G	1							
C	2							
C	3							
T	4							
G	5							
A	6							
T	7							

Comparing Sequences

Levenshtein distance

For each cell, calculate the minimum of:

Seq A

			G	C	A	C	T	G	A
		start							
	start	0	1	2	3	4	5	6	7
	G	1							
del ▶	C	2							
Seq B	C	3							
	T	4							
	G	5							
	A	6							
ins ▶	T	7							

Comparing Sequences

```
SeqA: G
SeqB: G
```

Levenshtein distance

For each cell, calculate the minimum of:

-> (i-1, j-1) + no shift (match = 0, mismatch +1)

Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G	1							
del ▶	C	2							
	C	3							
	T	4							
	G	5							
	A	6							
ins ▶	T	7							

Comparing Sequences

```
SeqA: G
SeqB: G
```

Levenshtein distance

For each cell, calculate the minimum of:

-> (i-1, j-1) + no shift (match = 0, mismatch +1)

```
Match/Mismatch: 0 + 0 = 0
```

Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G	1							
del ▶	C	2							
	C	3							
	T	4							
	G	5							
	A	6							
ins ▶	T	7							

Comparing Sequences

```
SeqA: G
SeqB: -
```

Levenshtein distance

For each cell, calculate the minimum of:

-> (i-1, j-1) + no shift (match = 0, mismatch +1)

-> (i, j-1) + insertion (gap +1)

```
Match/Mismatch: 0 + 0 = 0
Insertion:      1 + 1 = 2
```

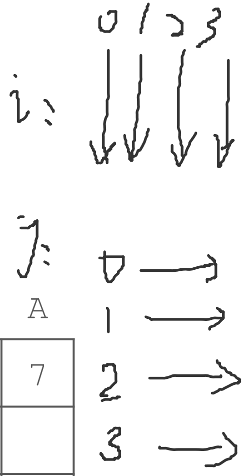
Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G	1							
del ▶	C	2							
	C	3							
	T	4							
	G	5							
	A	6							
ins ▶	T	7							

Comparing Sequences

```
SeqA: -
SeqB: G
```



Levenshtein distance

For each cell, calculate the minimum of:

-> (i-1, j-1) + no shift (match = 0, mismatch +1)

-> (i, j-1) + insertion (gap +1)

-> (i-1, j) + deletion (gap +1)

```
Match/Mismatch:  0 + 0 = 0
Insertion:       1 + 1 = 2
Deletion:        1 + 1 = 2
```

Seq A

	(i j)	start	G	C	A	C	T	G	A
	start	0	1 0 1	2 0 2	3 0 3	4	5	6	7
	G	0 1 1 →							
del ▶	C	0 2 2							
Seq B	C	0 3 3							
	T	4							
	G	5							
	A	6							
ins ▶	T	7							

Comparing Sequences

Levenshtein distance

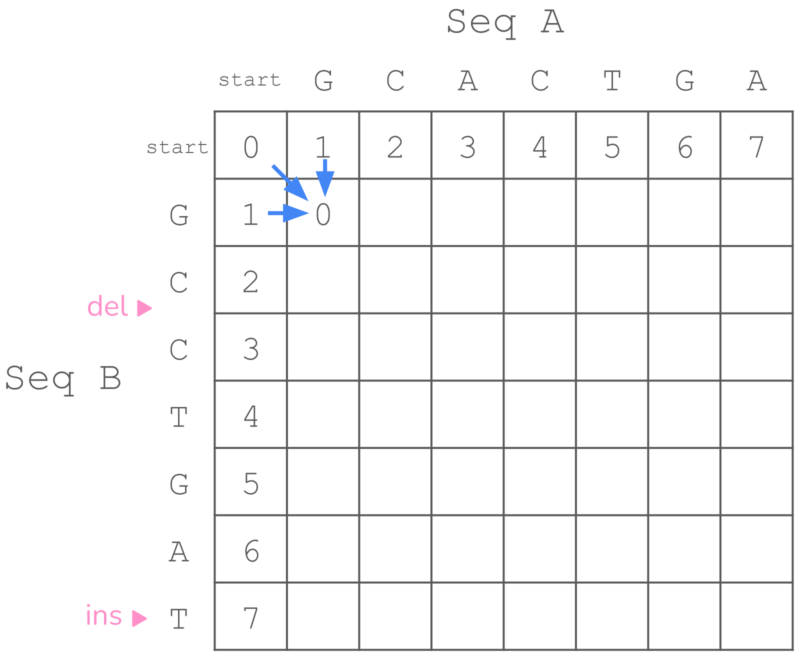
For each cell, calculate the minimum of:

-> (i-1, j-1) + no shift (match = 0, mismatch +1)

-> (i, j-1) + insertion (gap +1)

-> (i-1, j) + deletion (gap +1)

```
Match/Mismatch: 0 + 0 = 0
Insertion:      1 + 1 = 2
Deletion:       1 + 1 = 2
```



	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G	1	0						
C	2							
C	3							
T	4							
G	5							
A	6							
T	7							

Comparing Sequences

Levenshtein distance

For each cell, calculate the minimum of:

-> (i-1, j-1) + no shift (match = 0, mismatch +1)

-> (i, j-1) + insertion (gap +1)

-> (i-1, j) + deletion (gap +1)

Need to have completed

-> the cell diag up left

-> the cell above

-> the cell left

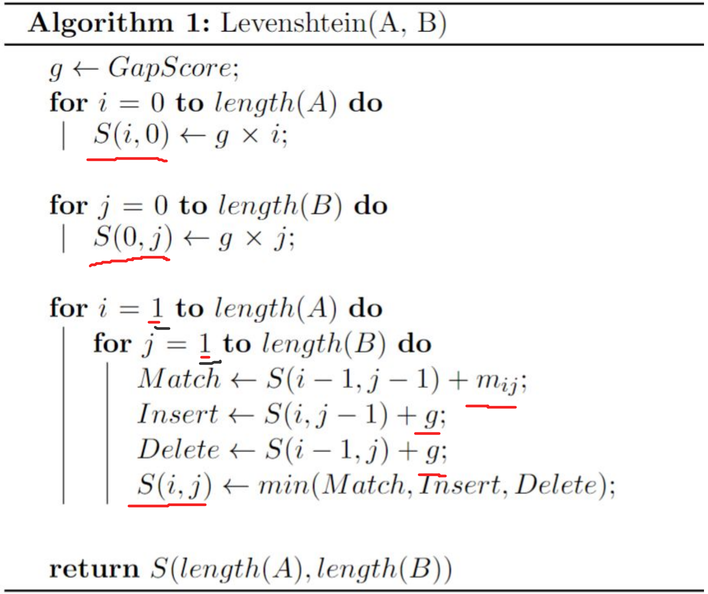
Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G	1	0						
del ▶	C	2							
	C	3							
	T	4							
	G	5							
	A	6							
ins ▶	T	7							

Comparing Sequences

Levenshtein distance



Algorithm 1: Levenshtein(A, B)

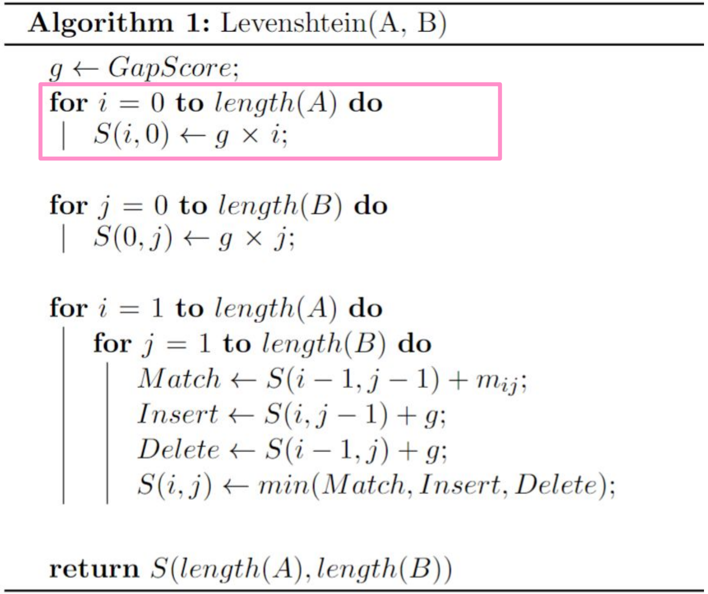
```
$g \leftarrow GapScore;$
for $i = 0$ to $length(A)$ do
    $S(i, 0) \leftarrow g \times i;$

for $j = 0$ to $length(B)$ do
    $S(0, j) \leftarrow g \times j;$

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1, j-1) + m_{ij};$
        $Insert \leftarrow S(i, j-1) + g;$
        $Delete \leftarrow S(i-1, j) + g;$
        $S(i, j) \leftarrow min(Match, Insert, Delete);$

return $S(length(A), length(B))$
```

Seq A

		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G	1	0						
del ▶	C	2							
Seq B	C	3							
	T	4							
	G	5							
	A	6							
ins ▶	T	7							

Comparing Sequences

Levenshtein distance



Algorithm 1: Levenshtein(A, B)

```
$g \leftarrow GapScore;$
for $i = 0$ to $length(A)$ do
    $S(i,0) \leftarrow g \times i;$

for $j = 0$ to $length(B)$ do
    $S(0,j) \leftarrow g \times j;$

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1,j-1) + m_{ij};$
        $Insert \leftarrow S(i,j-1) + g;$
        $Delete \leftarrow S(i-1,j) + g;$
        $S(i,j) \leftarrow min(Match, Insert, Delete);$

return $S(length(A), length(B))$
```

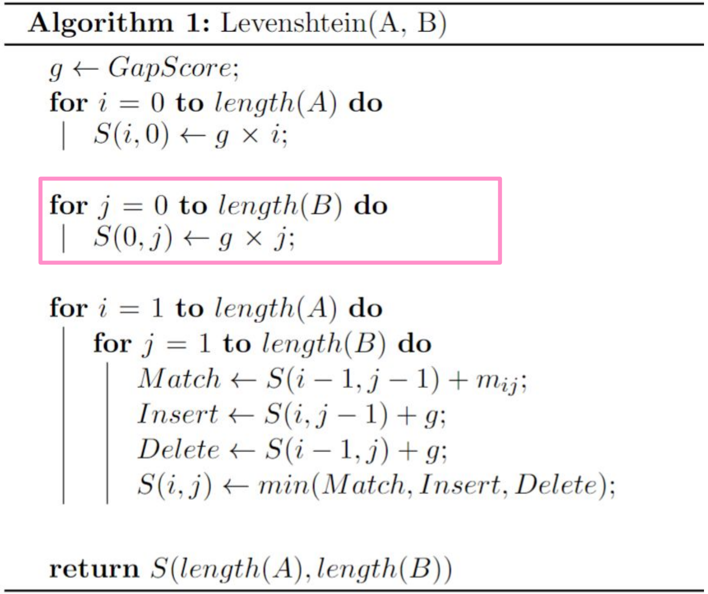
Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G	1	0						
del ▶	C	2							
	C	3							
	T	4							
	G	5							
	A	6							
ins ▶	T	7							

Comparing Sequences

Levenshtein distance



Algorithm 1: Levenshtein(A, B)

```
$g \leftarrow GapScore$;
for $i = 0$ to $length(A)$ do
    $S(i,0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0,j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1,j-1) + m_{ij}$;
        $Insert \leftarrow S(i,j-1) + g$;
        $Delete \leftarrow S(i-1,j) + g$;
        $S(i,j) \leftarrow min(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```

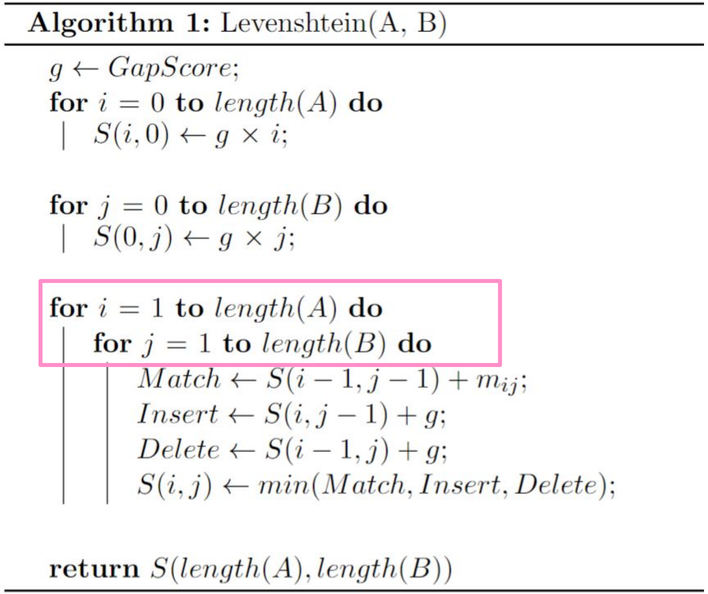
Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G	1	0						
	C	2							
del ▶	C	3							
	T	4							
	G	5							
	A	6							
ins ▶	T	7							

Comparing Sequences

Levenshtein distance



```
Algorithm 1: Levenshtein(A, B)

$g \leftarrow GapScore$;
for $i = 0$ to $length(A)$ do
    $S(i,0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0,j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1, j-1) + m_{ij}$;
        $Insert \leftarrow S(i, j-1) + g$;
        $Delete \leftarrow S(i-1, j) + g$;
        $S(i,j) \leftarrow min(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```

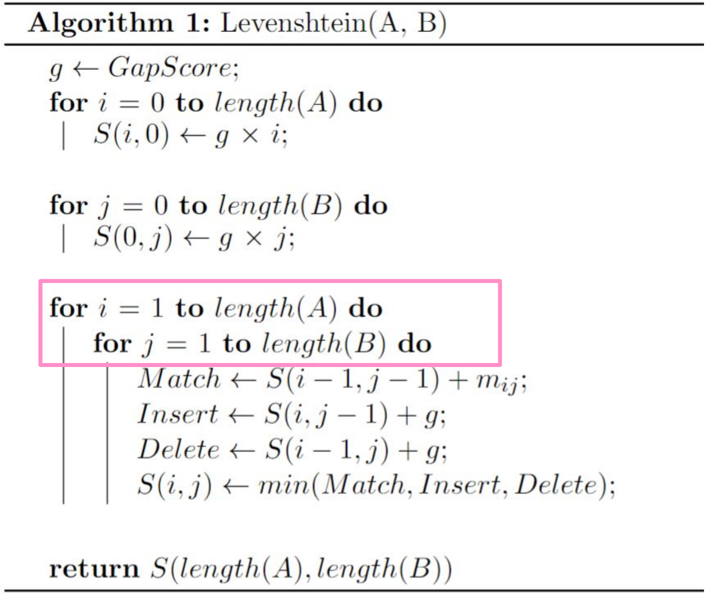
Seq A

Seq B

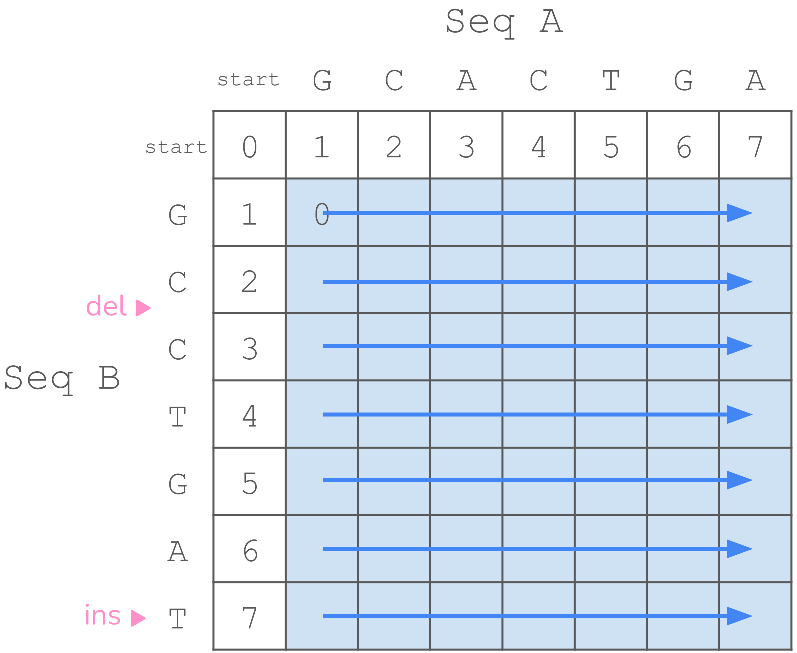
		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G	1	0						
del ▶	C	2							
	C	3							
	T	4							
	G	5							
	A	6							
ins ▶	T	7							

Comparing Sequences

Levenshtein distance



```
Algorithm 1: Levenshtein(A, B)
$g \leftarrow GapScore$;
for $i = 0$ to $length(A)$ do
    $S(i,0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0,j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1, j-1) + m_{ij}$;
        $Insert \leftarrow S(i, j-1) + g$;
        $Delete \leftarrow S(i-1, j) + g$;
        $S(i,j) \leftarrow min(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```

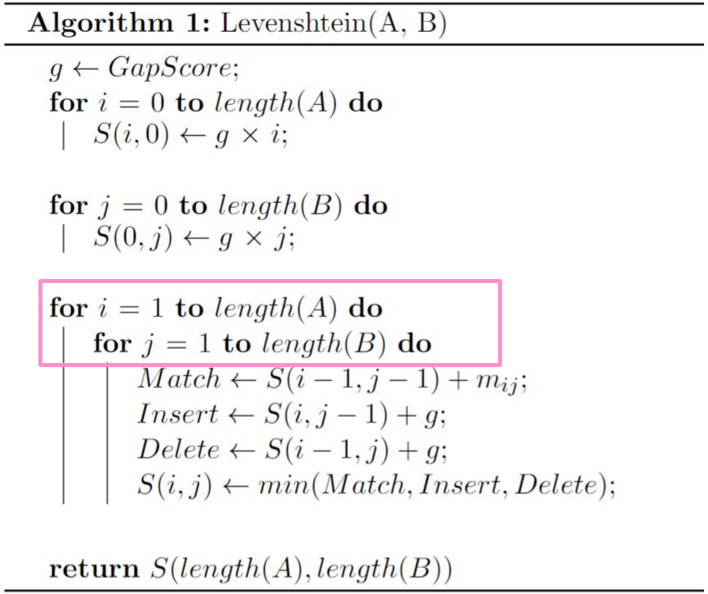


	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G	1	0						
C	2							
C	3							
T	4							
G	5							
A	6							
T	7							

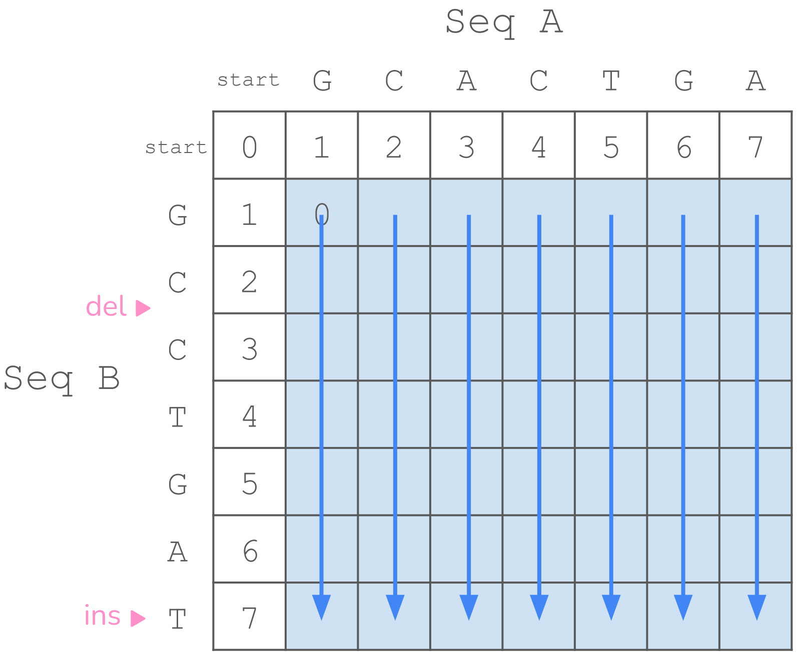
Both ok!

Comparing Sequences

Levenshtein distance



```
Algorithm 1: Levenshtein(A, B)
$g \leftarrow GapScore$;
for $i = 0$ to $length(A)$ do
    $S(i,0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0,j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1, j-1) + m_{ij}$;
        $Insert \leftarrow S(i, j-1) + g$;
        $Delete \leftarrow S(i-1, j) + g$;
        $S(i,j) \leftarrow min(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```



Seq A

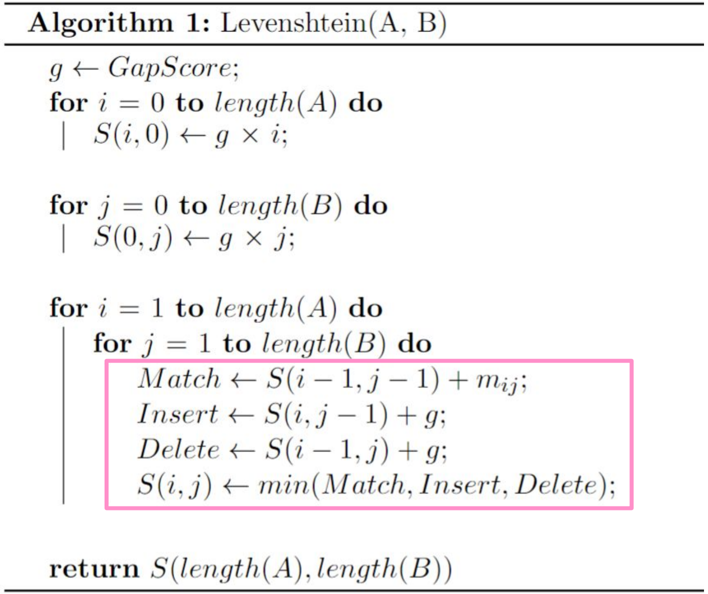
Seq B

	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G	1	0						
C	2							
C	3							
T	4							
G	5							
A	6							
T	7							

Both ok!

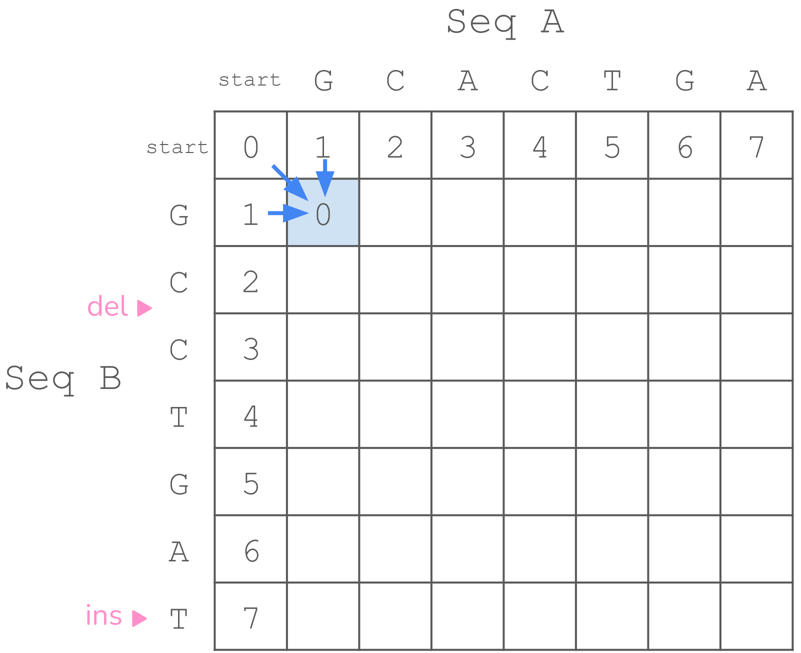
Comparing Sequences

Levenshtein distance



Algorithm 1: Levenshtein(A, B)

```
$g \leftarrow GapScore$;
for $i = 0$ to $length(A)$ do
    $S(i, 0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0, j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1, j-1) + m_{ij}$;
        $Insert \leftarrow S(i, j-1) + g$;
        $Delete \leftarrow S(i-1, j) + g$;
        $S(i, j) \leftarrow min(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```

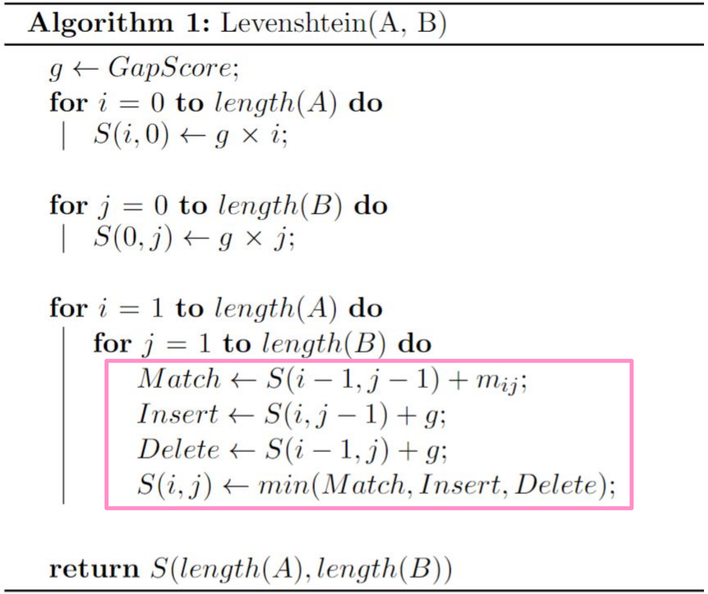


	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G	1	0						
C	2							
C	3							
T	4							
G	5							
A	6							
T	7							

```
Match/Mismatch: 0 + 0 = 0
Insertion:      1 + 1 = 2
Deletion:       1 + 1 = 2
```

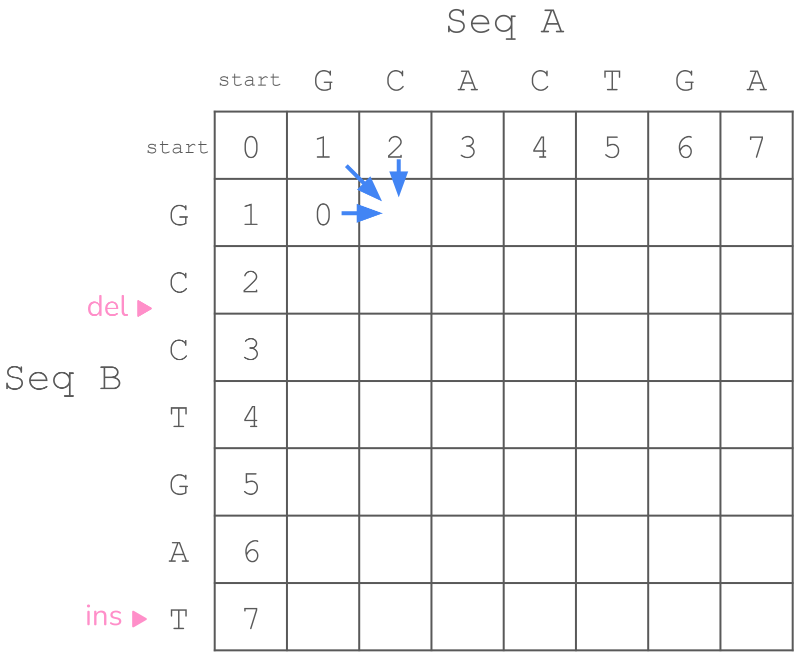
Comparing Sequences

Levenshtein distance



Algorithm 1: Levenshtein(A, B)

```
$g \leftarrow GapScore;$
for $i = 0$ to $length(A)$ do
    $S(i,0) \leftarrow g \times i;$

for $j = 0$ to $length(B)$ do
    $S(0,j) \leftarrow g \times j;$

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1, j-1) + m_{ij};$
        $Insert \leftarrow S(i, j-1) + g;$
        $Delete \leftarrow S(i-1, j) + g;$
        $S(i,j) \leftarrow min(Match, Insert, Delete);$

return $S(length(A), length(B))$
```

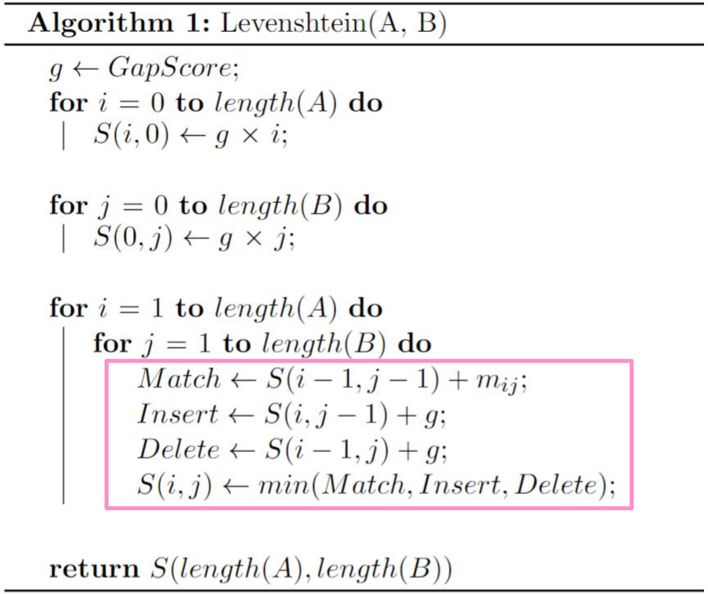


	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G	1	0						
C	2							
C	3							
T	4							
G	5							
A	6							
T	7							

```
Match/Mismatch: 1 + 1 = 2
Insertion:      2 + 1 = 3
Deletion:       0 + 1 = 1
```

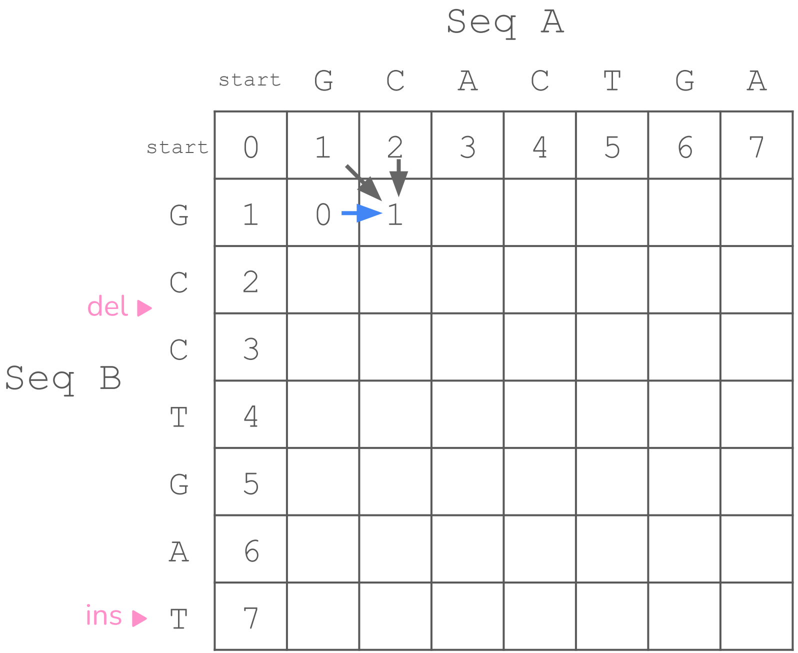
Comparing Sequences

Levenshtein distance



Algorithm 1: Levenshtein(A, B)

```
$g \leftarrow GapScore$;
for $i = 0$ to $length(A)$ do
    $S(i, 0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0, j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1, j-1) + m_{ij}$;
        $Insert \leftarrow S(i, j-1) + g$;
        $Delete \leftarrow S(i-1, j) + g$;
        $S(i, j) \leftarrow min(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```

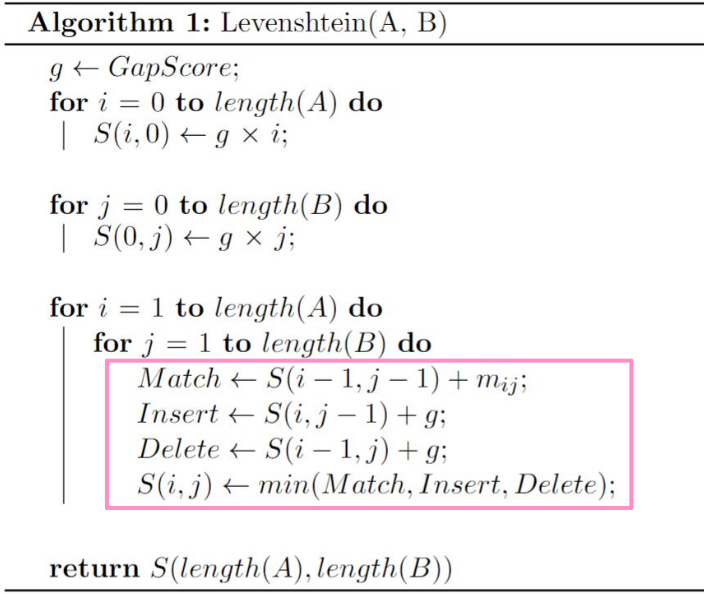


	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G	1	0	1					
C	2							
C	3							
T	4							
G	5							
A	6							
T	7							

```
Match/Mismatch: 1 + 1 = 2
Insertion:      2 + 1 = 3
Deletion:       0 + 1 = 1
```

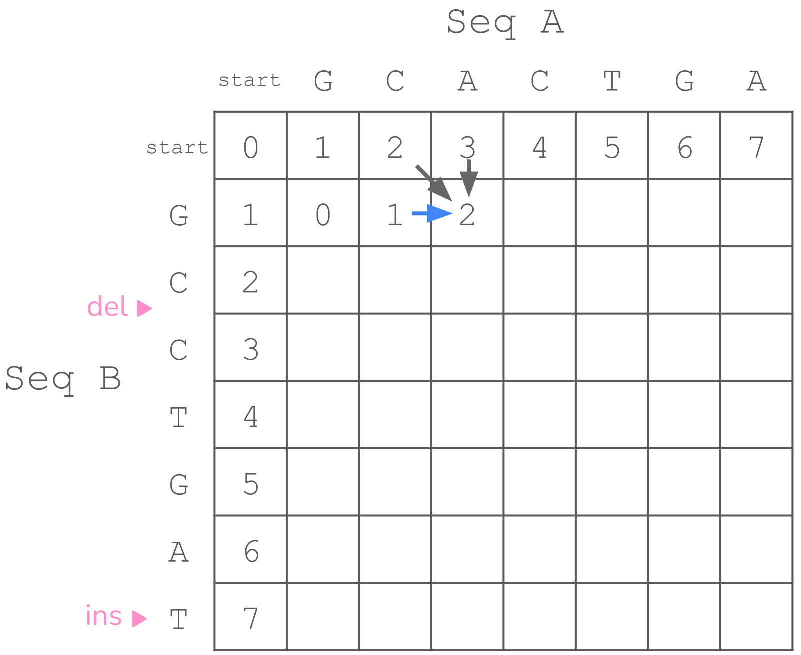
Comparing Sequences

Levenshtein distance



Algorithm 1: Levenshtein(A, B)

```
$g \leftarrow GapScore$;
for $i = 0$ to $length(A)$ do
    $S(i, 0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0, j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1, j-1) + m_{ij}$;
        $Insert \leftarrow S(i, j-1) + g$;
        $Delete \leftarrow S(i-1, j) + g$;
        $S(i, j) \leftarrow min(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```



Seq A

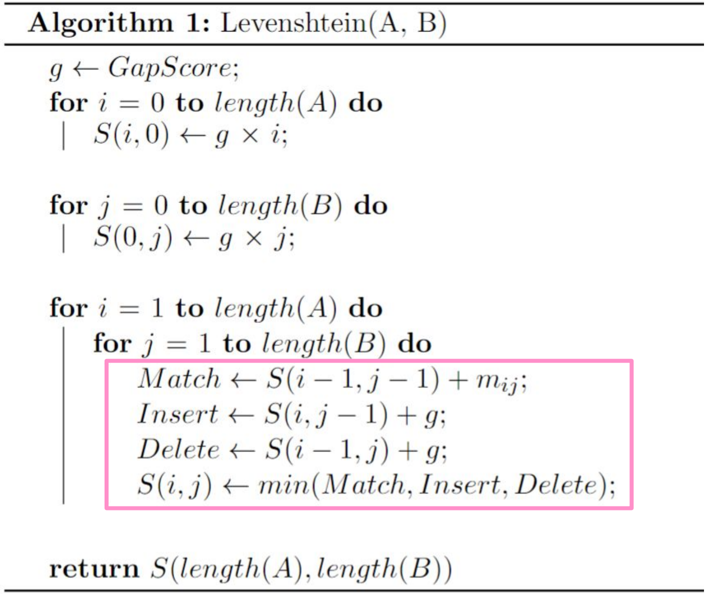
Seq B

		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G	1	0	1	2				
del ▶	C	2							
	C	3							
	T	4							
	G	5							
	A	6							
ins ▶	T	7							

```
Match/Mismatch: 2 + 1 = 3
Insertion:      3 + 1 = 4
Deletion:       1 + 1 = 2
```

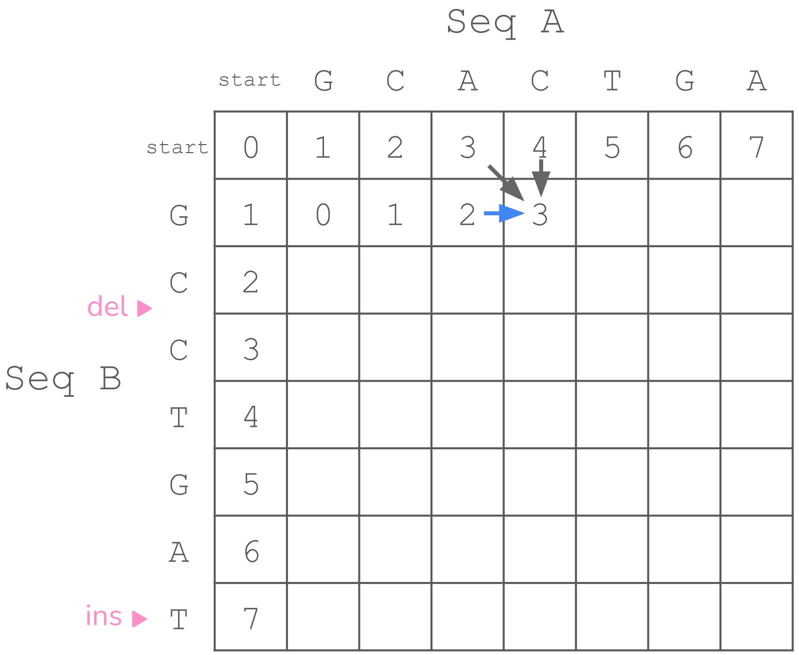
Comparing Sequences

Levenshtein distance



Algorithm 1: Levenshtein(A, B)

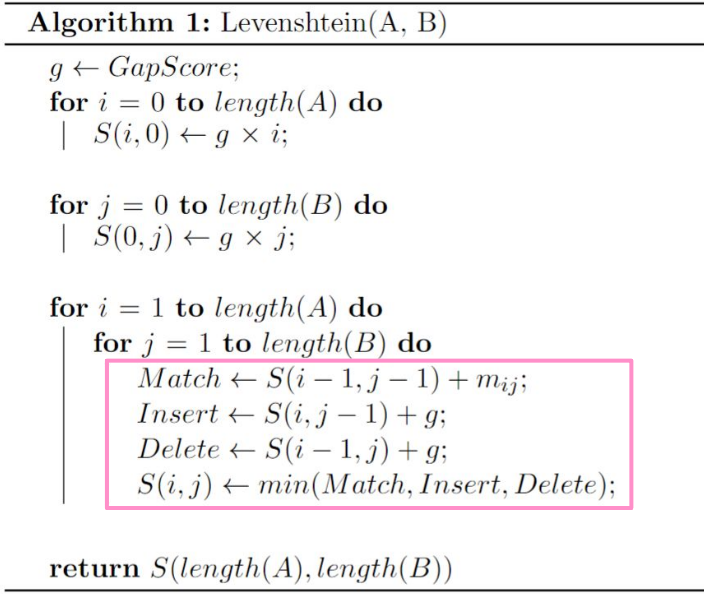
```
$g \leftarrow GapScore$;
for $i = 0$ to $length(A)$ do
    $S(i, 0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0, j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1, j-1) + m_{ij}$;
        $Insert \leftarrow S(i, j-1) + g$;
        $Delete \leftarrow S(i-1, j) + g$;
        $S(i, j) \leftarrow min(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```



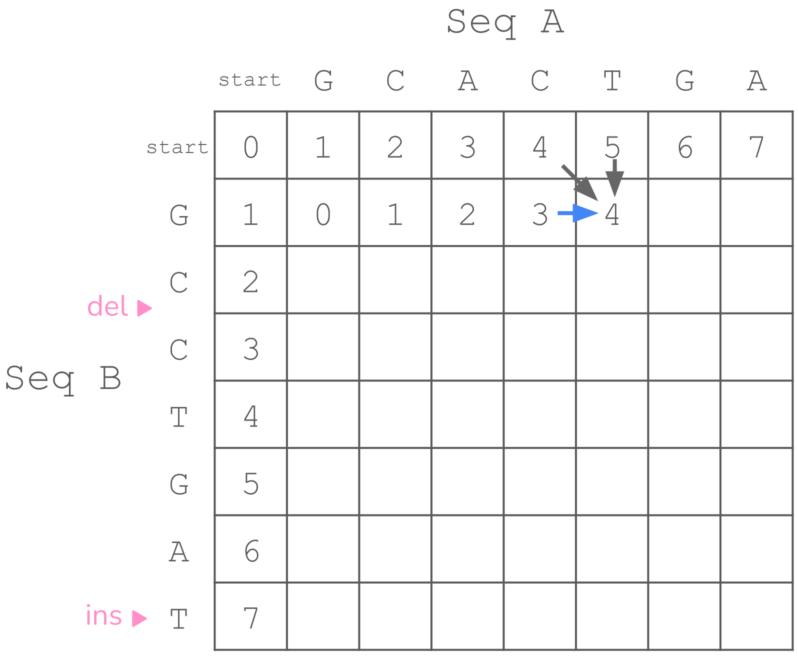
	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G	1	0	1	2	3			
C	2							
C	3							
T	4							
G	5							
A	6							
T	7							

Comparing Sequences

Levenshtein distance



```
Algorithm 1: Levenshtein(A, B)
g ← GapScore;
for i = 0 to length(A) do
  | S(i,0) ← g × i;

for j = 0 to length(B) do
  | S(0,j) ← g × j;

for i = 1 to length(A) do
  | for j = 1 to length(B) do
  |   Match ← S(i − 1, j − 1) + m_ij;
  |   Insert ← S(i, j − 1) + g;
  |   Delete ← S(i − 1, j) + g;
  |   S(i, j) ← min(Match, Insert, Delete);

return S(length(A), length(B))
```



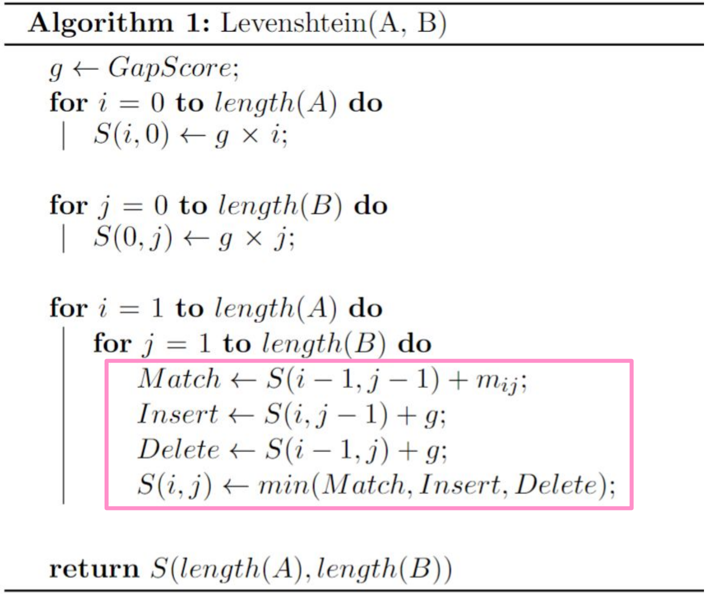
Seq A

Seq B

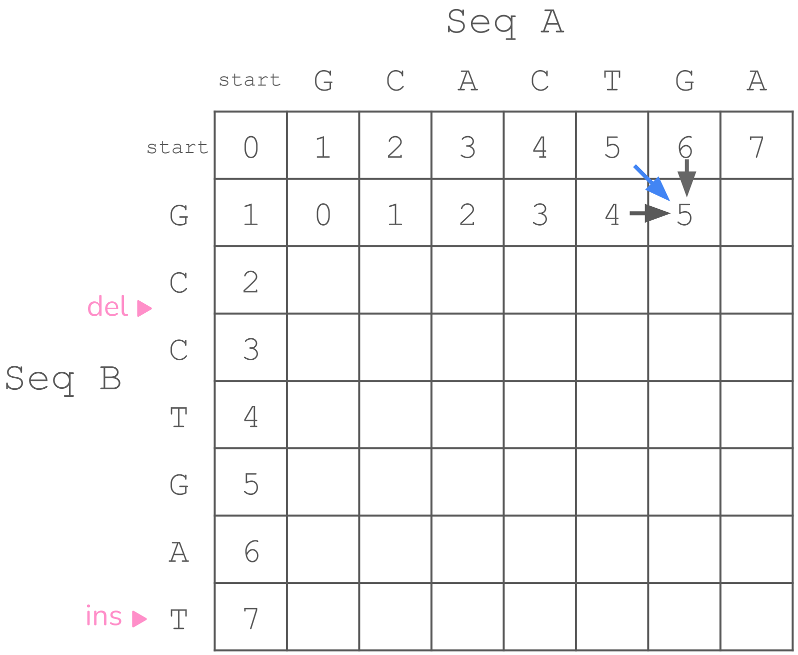
	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G	1	0	1	2	3	4		
C	2							
C	3							
T	4							
G	5							
A	6							
T	7							

Comparing Sequences

Levenshtein distance



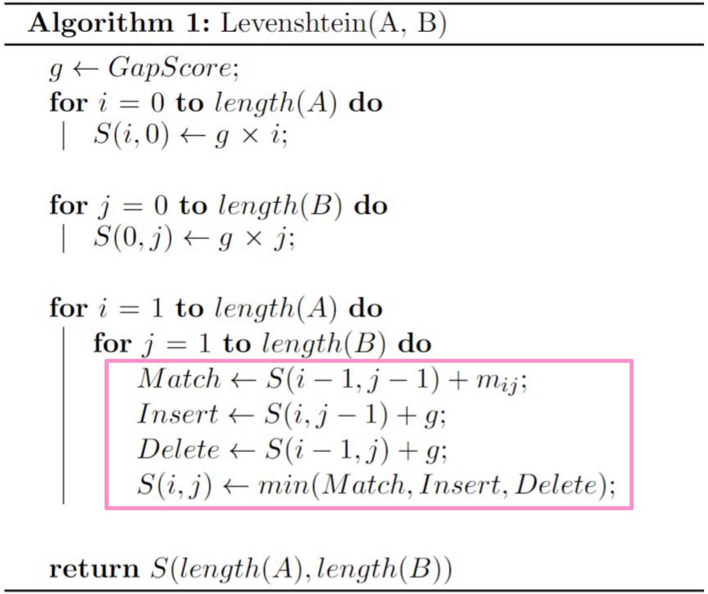
```
Algorithm 1: Levenshtein(A, B)
    $g \leftarrow GapScore$;
    for $i = 0$ to $length(A)$ do
        $S(i,0) \leftarrow g \times i$;

    for $j = 0$ to $length(B)$ do
        $S(0,j) \leftarrow g \times j$;

    for $i = 1$ to $length(A)$ do
        for $j = 1$ to $length(B)$ do
            $Match \leftarrow S(i-1, j-1) + m_{ij}$;
            $Insert \leftarrow S(i, j-1) + g$;
            $Delete \leftarrow S(i-1, j) + g$;
            $S(i,j) \leftarrow min(Match, Insert, Delete)$;

    return $S(length(A), length(B))$
```



	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G	1	0	1	2	3	4	5	
C	2							
C	3							
T	4							
G	5							
A	6							
T	7							

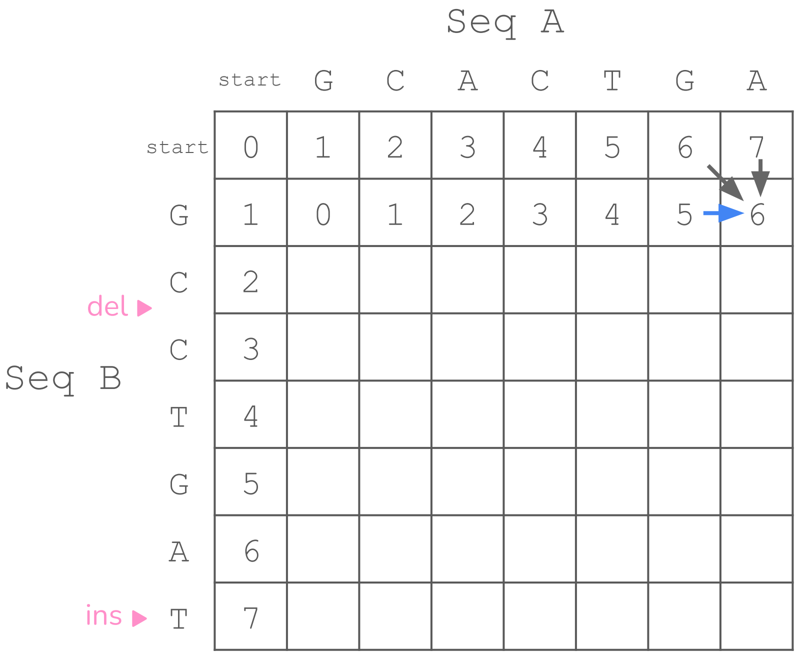
Comparing Sequences

Levenshtein distance



Algorithm 1: Levenshtein(A, B)

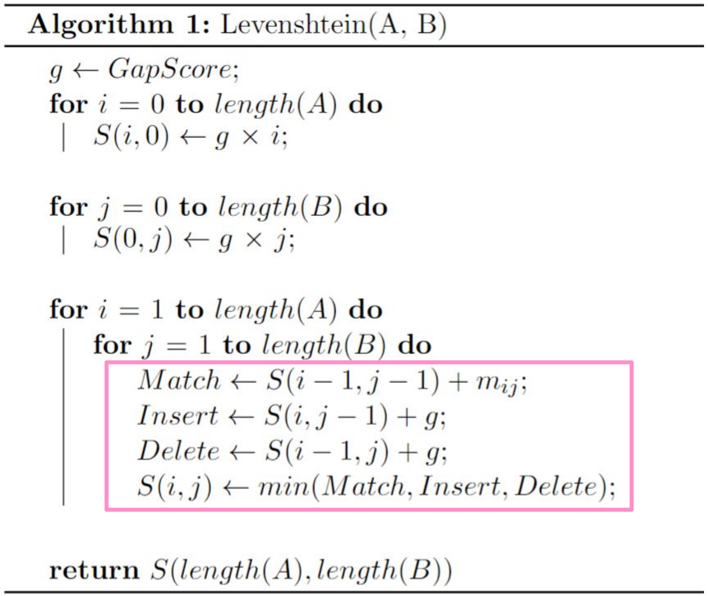
```
$g \leftarrow GapScore;$
for $i = 0$ to $length(A)$ do
    $S(i, 0) \leftarrow g \times i;$

for $j = 0$ to $length(B)$ do
    $S(0, j) \leftarrow g \times j;$

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1, j-1) + m_{ij};$
        $Insert \leftarrow S(i, j-1) + g;$
        $Delete \leftarrow S(i-1, j) + g;$
        $S(i, j) \leftarrow min(Match, Insert, Delete);$

return $S(length(A), length(B))$
```



	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G	1	0	1	2	3	4	5	6
C	2							
C	3							
T	4							
G	5							
A	6							
T	7							

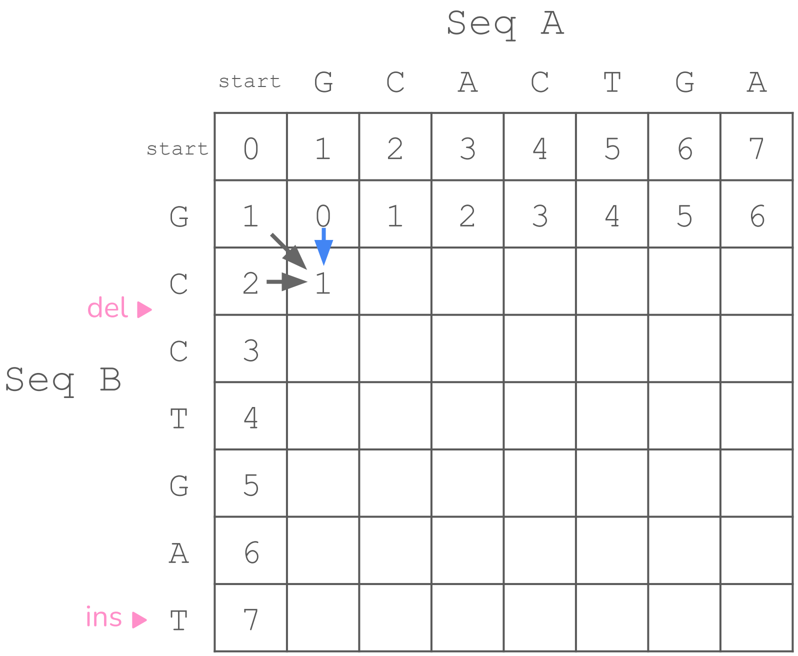
Comparing Sequences

Levenshtein distance



Algorithm 1: Levenshtein(A, B)

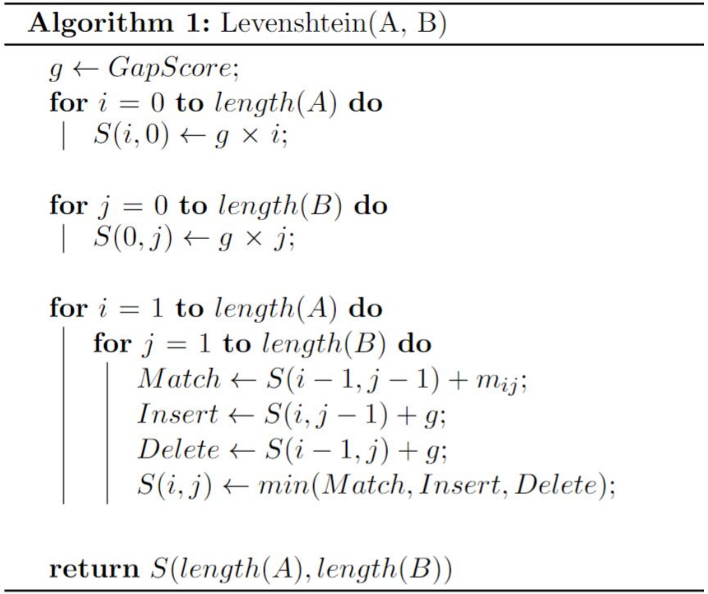
```
$g \leftarrow GapScore$;
for $i = 0$ to $length(A)$ do
    $S(i, 0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0, j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1, j-1) + m_{ij}$;
        $Insert \leftarrow S(i, j-1) + g$;
        $Delete \leftarrow S(i-1, j) + g$;
        $S(i, j) \leftarrow min(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```



	start	G	C	A	C	T	G	A
start	0	1	2	3	4	5	6	7
G	1	0	1	2	3	4	5	6
C	2	1						
C	3							
T	4							
G	5							
A	6							
T	7							

Comparing Sequences

Levenshtein distance



Algorithm 1: Levenshtein(A, B)

```
$g \leftarrow GapScore$;
for $i = 0$ to $length(A)$ do
    $S(i,0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0,j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1,j-1) + m_{ij}$;
        $Insert \leftarrow S(i,j-1) + g$;
        $Delete \leftarrow S(i-1,j) + g$;
        $S(i,j) \leftarrow min(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```

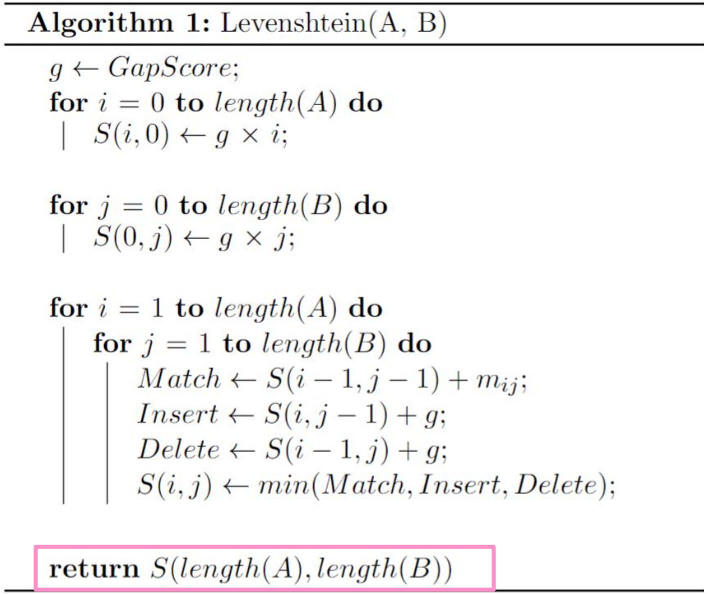
Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G	1	0	1	2	3	4	5	6
del ▶	C	2	1	0	1	2	3	4	5
	C	3	2	1	1	1	2	3	4
	T	4	3	2	2	2	1	2	3
	G	5	4	3	3	3	2	1	2
	A	6	5	4	3	4	3	2	1
ins ▶	T	7	6	5	4	4	4	3	2

Comparing Sequences

Levenshtein distance



```
Algorithm 1: Levenshtein(A, B)
  $g \leftarrow GapScore$;
  for $i = 0$ to $length(A)$ do
  |   $S(i,0) \leftarrow g \times i$;

  for $j = 0$ to $length(B)$ do
  |   $S(0,j) \leftarrow g \times j$;

  for $i = 1$ to $length(A)$ do
  |   for $j = 1$ to $length(B)$ do
  |   |   $Match \leftarrow S(i-1, j-1) + m_{ij}$;
  |   |   $Insert \leftarrow S(i, j-1) + g$;
  |   |   $Delete \leftarrow S(i-1, j) + g$;
  |   |   $S(i,j) \leftarrow min(Match, Insert, Delete)$;

  return $S(length(A), length(B))$
```

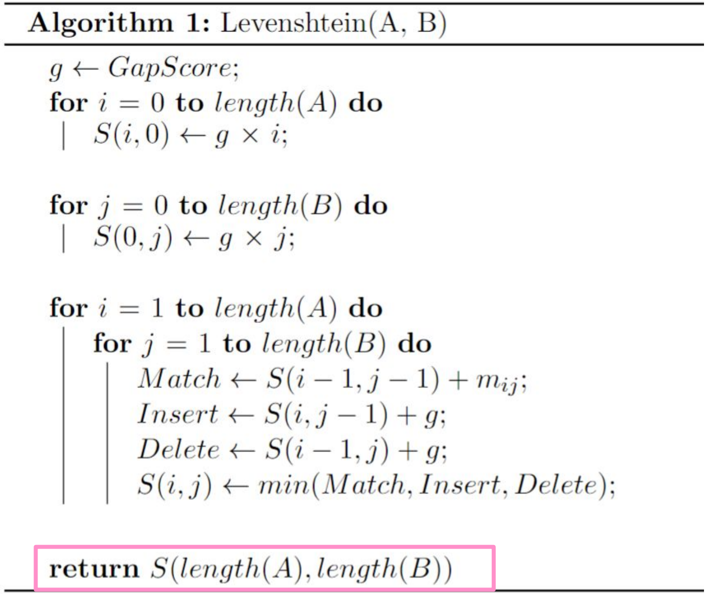
Seq A

		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G	1	0	1	2	3	4	5	6
del ▶	C	2	1	0	1	2	3	4	5
	C	3	2	1	1	1	2	3	4
Seq B	T	4	3	2	2	2	1	2	3
	G	5	4	3	3	3	2	1	2
	A	6	5	4	3	4	3	2	1
ins ▶	T	7	6	5	4	4	4	3	2

Return the bottom right cell as edit distance

Comparing Sequences

Levenshtein distance



Algorithm 1: Levenshtein(A, B)

```
$g \leftarrow GapScore$;
for $i = 0$ to $length(A)$ do
    $S(i, 0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0, j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1, j-1) + m_{ij}$;
        $Insert \leftarrow S(i, j-1) + g$;
        $Delete \leftarrow S(i-1, j) + g$;
        $S(i, j) \leftarrow min(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```

Seq A

Seq B

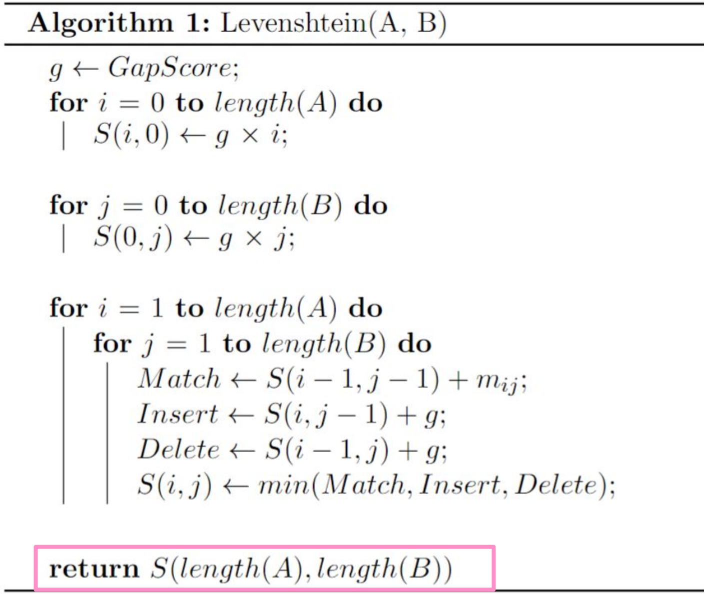
		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G	1	0	1	2	3	4	5	6
del ▶	C	2	1	0	1	2	3	4	5
	C	3	2	1	1	1	2	3	4
	T	4	3	2	2	2	1	2	3
	G	5	4	3	3	3	2	1	2
	A	6	5	4	3	4	3	2	1
ins ▶	T	7	6	5	4	4	4	3	2

Return the bottom right cell as edit distance

It’s 2! Same as what we expected!

Comparing Sequences

Levenshtein distance



```
Algorithm 1: Levenshtein(A, B)
    $g \leftarrow GapScore$;
    for $i = 0$ to $length(A)$ do
        $S(i,0) \leftarrow g \times i$;

    for $j = 0$ to $length(B)$ do
        $S(0,j) \leftarrow g \times j$;

    for $i = 1$ to $length(A)$ do
        for $j = 1$ to $length(B)$ do
            $Match \leftarrow S(i-1,j-1) + m_{ij}$;
            $Insert \leftarrow S(i,j-1) + g$;
            $Delete \leftarrow S(i-1,j) + g$;
            $S(i,j) \leftarrow min(Match, Insert, Delete)$;

    return $S(length(A), length(B))$
```

Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	1	2	3	4	5	6	7
	G	1	0	1	2	3	4	5	6
del ▶	C	2	1	0	1	2	3	4	5
	C	3	2	1	1	1	2	3	4
	T	4	3	2	2	2	1	2	3
	G	5	4	3	3	3	2	1	2
	A	6	5	4	3	4	3	2	1
ins ▶	T	7	6	5	4	4	4	3	2

Return the bottom right cell as edit distance

It's 2! Same as what we expected!

Pairwise Alignment

Pairwise Alignment

Levenshtein distance

- Forms the basis for the remainder of lecture
  - Global alignment
  - Local alignment
  - Semi-global alignment

These add a few important variations:

- Penalties rather than adding edits
- Penalties are different for mismatch vs gap
- The arrows are stored
- (directions we took to calculate each cell)

SPLATTERING -> PATTERN

Global	SPLATTERING -P-ATTERN--
Local	ATTER ATTER
Semi-global	PLATTERIN P-ATTER-N

全局比对是指在整个序列长度上比较两个序列，通常需要在序列的开始和结束位置添加间隙（gap）。
示例中，“SPLATTERING”与“PATTERN”进行全局比对，结果显示需要在“PATTERN”前后添加间隙以匹配“SPLATTERING”的长度。
局部比对寻找两个序列中最匹配的局部区域，而不是整个序列。
示例中，“ATTER”是两个序列中对应得最好的部分，这种比对不关心序列的其余部分。
半全局比对是全局比对和局部比对的结合体，它对整个序列进行比对，但是不对开头和结尾的间隙进行惩罚。
示例中，“PLATTERIN”与“PATTERN”进行比对时，在开头和结尾的间隙没有得到惩罚。

Pairwise Alignment

Penalties

Instead of adding edits, penalise certain actions

Penalties should be more severe for gaps

Mismatches more common in biology

In coding regions, gaps change the reading frame!

More unlikely to witness.

Arrows are stored

Can backtrack through the grid to return an alignment

CATS
_ATE
--AT

Penalties	
Match	0
Mismatch	-1
Gap	-2

	start	C	A	T	S
start	0	→ -2	→ -4	→ -6	→ -8
A	↓ -2	↘ -1	↘ -2	→ -4	→ -6
T	↓ -4	↘ -3	↘ -2	↘ -2	→ -4
E	↓ -6	↘ -5	↘ -4	↘ -3	↘ -3

```
ALIGNMENT: CATS
           -ATE
```

Pairwise Alignment

Penalties

Instead of adding edits, penalise certain actions

Penalties should be more severe for gaps

Mismatches more common in biology

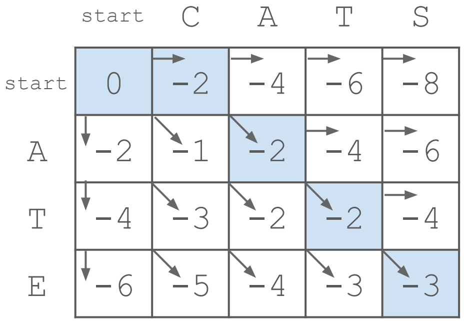
In coding regions, gaps change the reading frame!

More unlikely to witness.

Arrows are stored

Can backtrack through the grid to return an alignment

Penalties	
Match	0
Mismatch	-1
Gap	-2



	start	C	A	T	S
start	0	→ -2	→ -4	→ -6	→ -8
A	↓ -2	↘ -1	↘ -2	→ -4	→ -6
T	↓ -4	↘ -3	↘ -2	↘ -2	→ -4
E	↓ -6	↘ -5	↘ -4	↘ -3	↘ -3

```
ALIGNMENT: CATS
           -ATE
```

Global Alignment

End-to-end alignment of two sequences

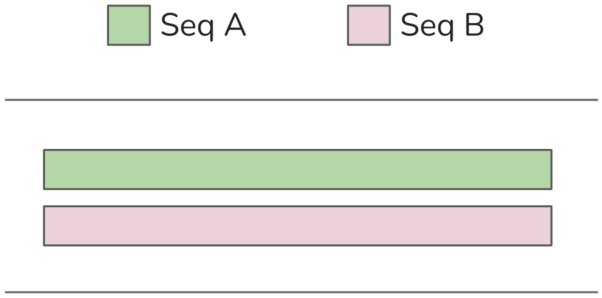
All of Seq A
All of Seq B

Best when sequences are similar in length & expected to be similar throughout

eg. Read alignment to segment of genome

Returns the full alignment

Algorithm: Needleman Wunsch



Global Alignment

End-to-end alignment of two sequences

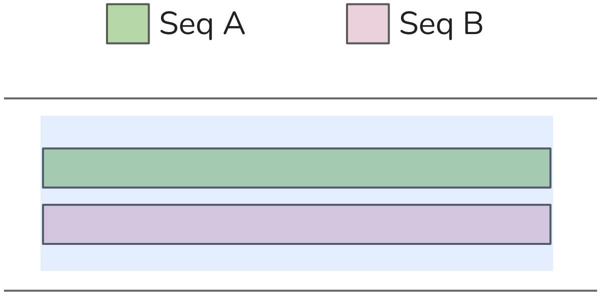
All of Seq A
All of Seq B

Best when sequences are similar in length & expected to be similar throughout

eg. Read alignment to segment of genome

Returns the full alignment

Algorithm: Needleman Wunsch



Global Alignment

Algorithm: Needleman Wunsch

Same as Levenshtein, except use scoring system & arrow directions are stored

Penalties generally greater for gaps than mismatches

Scoring (**Gaps:** -2)

	C	**T**	**A**	**G**
C	+1	-1	-1	-1
T	-1	+1	-1	-1
A	-1	-1	+1	-1
G	-1	-1	-1	+1

Seq A

Seq B

	start	G	C	A	C	T	G	A
start								
G								
del ▶ C								
C								
T								
G								
A								
ins ▶ T								

Global Alignment

Algorithm: Needleman Wunsch

Same as Levenshtein, except use scoring system & arrow directions are stored

Penalties generally greater for gaps than mismatches

For this example, scoring system + gap penalties seen top right.

Real scoring schemes more complex
(explored next lecture)

Scoring (**Gaps:** -2)

	C	**T**	**A**	**G**
C	+1	-1	-1	-1
T	-1	+1	-1	-1
A	-1	-1	+1	-1
G	-1	-1	-1	+1

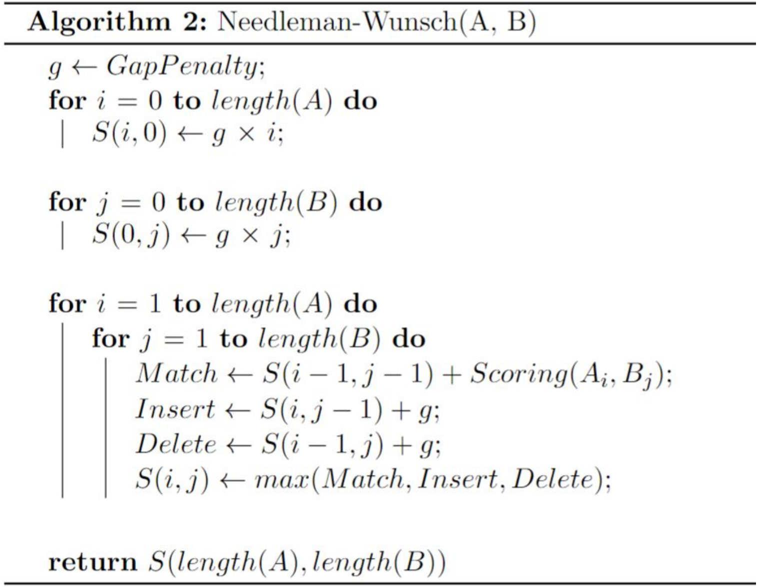
Seq A

Seq B

		start	G	C	A	C	T	G	A
	start								
	G								
del ▶	C								
	C								
	T								
	G								
	A								
ins ▶	T								

Global Alignment

Algorithm: Needleman Wunsch



Algorithm 2: Needleman-Wunsch(A, B)

```
$g \leftarrow GapPenalty$;
for $i = 0$ to $length(A)$ do
    $S(i,0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0,j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1,j-1) + Scoring(A_i, B_j)$;
        $Insert \leftarrow S(i,j-1) + g$;
        $Delete \leftarrow S(i-1,j) + g$;
        $S(i,j) \leftarrow max(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```

Scoring (**Gaps:** -2)

	C	**T**	**A**	**G**
C	+1	-1	-1	-1
T	-1	+1	-1	-1
A	-1	-1	+1	-1
G	-1	-1	-1	+1

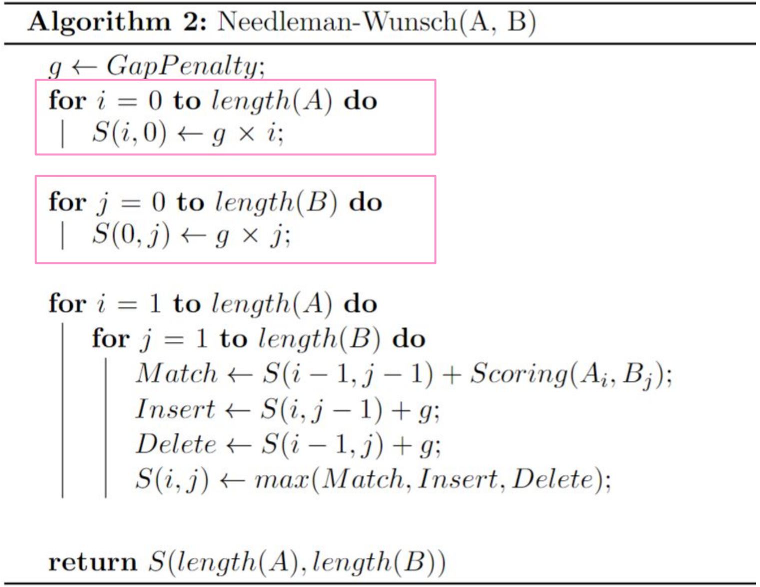
Seq A

Seq B

		start	G	C	A	C	T	G	A
	start								
	G								
del ▶	C								
	C								
	T								
	G								
	A								
ins ▶	T								

Global Alignment

Algorithm: Needleman Wunsch



```
Algorithm 2: Needleman-Wunsch(A, B)
g ← GapPenalty;
for $i = 0$ to $length(A)$ do
    $S(i,0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0,j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1,j-1) + Scoring(A_i, B_j)$;
        $Insert \leftarrow S(i,j-1) + g$;
        $Delete \leftarrow S(i-1,j) + g$;
        $S(i,j) \leftarrow max(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```

Scoring (**Gaps:** -2)

	C	**T**	**A**	**G**
C	+1	-1	-1	-1
T	-1	+1	-1	-1
A	-1	-1	+1	-1
G	-1	-1	-1	+1

Seq A

Seq B

	start	G	C	A	C	T	G	A
start	0	-2	-4	-6	-8	-10	-12	-14
G	-2							
del ▶ C	-4							
C	-6							
T	-8							
G	-10							
A	-12							
ins ▶ T	-14							

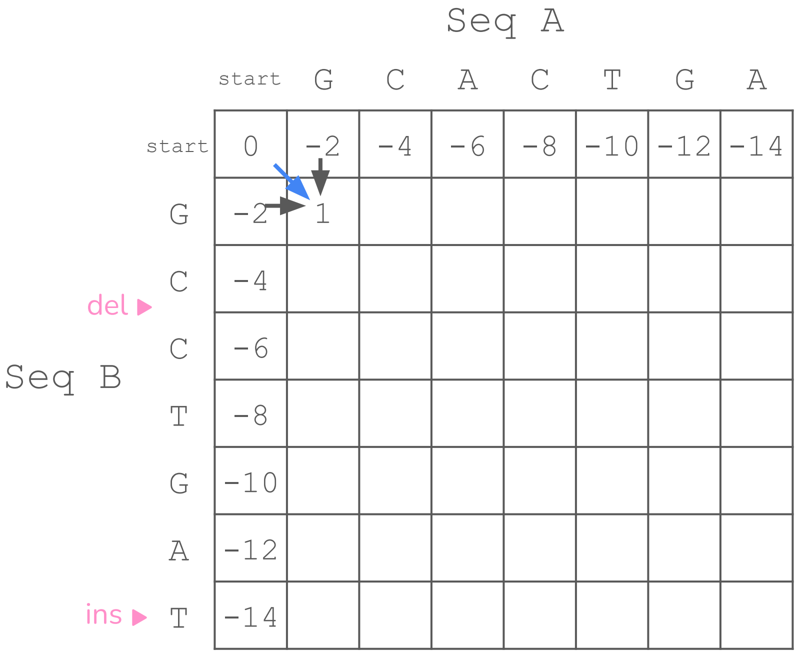
Global Alignment

Algorithm: Needleman Wunsch

```
Algorithm 2: Needleman-Wunsch(A, B)

$g \leftarrow GapPenalty$;
for $i = 0$ to $length(A)$ do
    $S(i, 0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0, j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1, j-1) + Scoring(A_i, B_j)$;
        $Insert \leftarrow S(i, j-1) + g$;
        $Delete \leftarrow S(i-1, j) + g$;
        $S(i, j) \leftarrow \underline{max}(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```

Scoring (**Gaps:** -2)

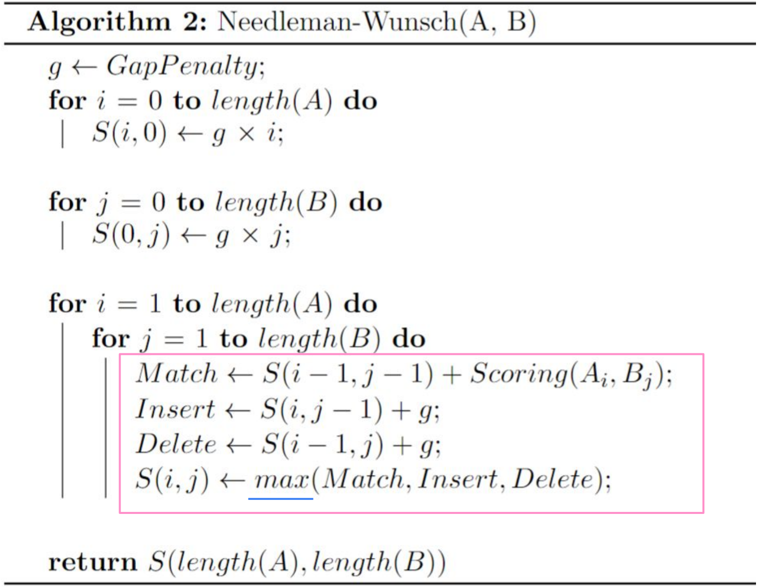
	C	**T**	**A**	**G**
C	+1	-1	-1	-1
T	-1	+1	-1	-1
A	-1	-1	+1	-1
G	-1	-1	-1	+1



	start	G	C	A	C	T	G	A
start	0	-2	-4	-6	-8	-10	-12	-14
G	-2	1						
C	-4							
C	-6							
T	-8							
G	-10							
A	-12							
T	-14							

Global Alignment

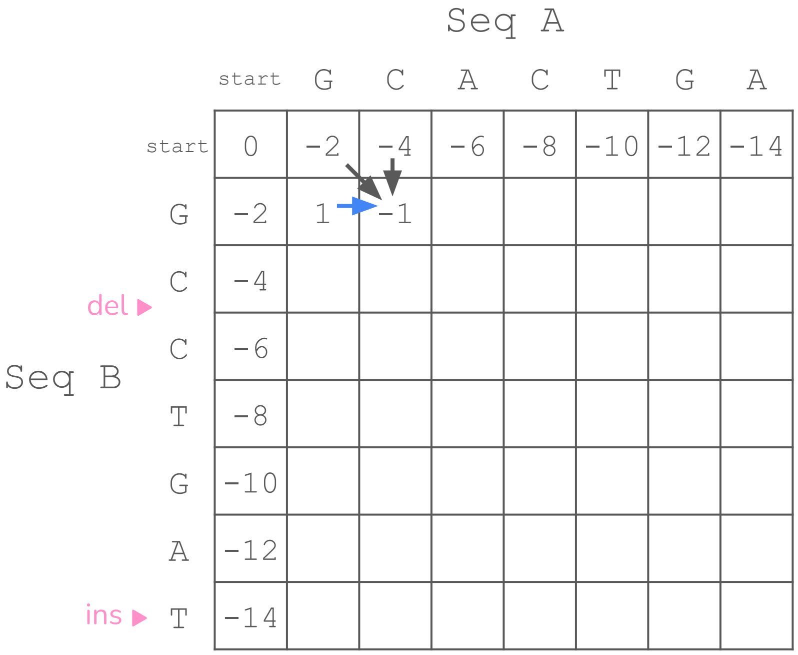
Algorithm: Needleman Wunsch



```
Algorithm 2: Needleman-Wunsch(A, B)

$g \leftarrow GapPenalty$;
for $i = 0$ to $length(A)$ do
    $S(i,0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0,j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1, j-1) + Scoring(A_i, B_j)$;
        $Insert \leftarrow S(i, j-1) + g$;
        $Delete \leftarrow S(i-1, j) + g$;
        $S(i,j) \leftarrow \underline{max}(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```

Scoring (**Gaps:** -2)

	C	**T**	**A**	**G**
C	+1	-1	-1	-1
T	-1	+1	-1	-1
A	-1	-1	+1	-1
G	-1	-1	-1	+1



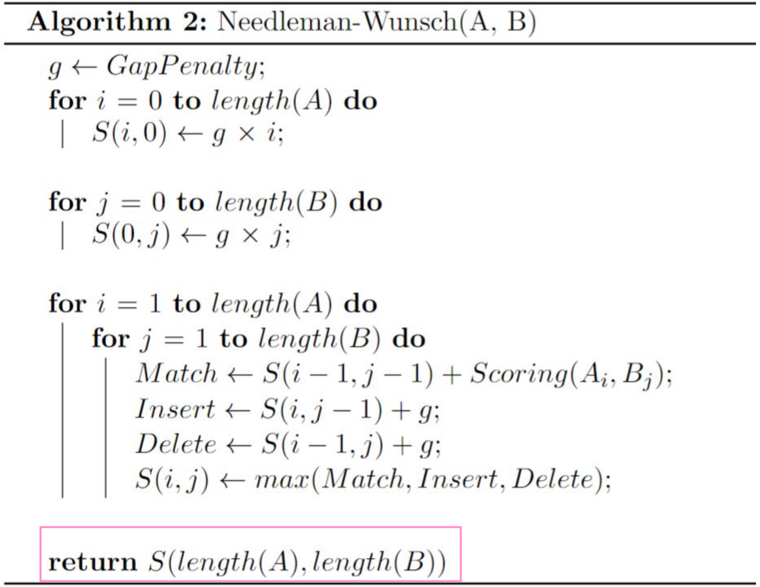
Seq A

Seq B

	start	G	C	A	C	T	G	A
start	0	-2	-4	-6	-8	-10	-12	-14
G	-2	1	-1					
C	-4							
C	-6							
T	-8							
G	-10							
A	-12							
T	-14							

Global Alignment

Algorithm: Needleman Wunsch



Algorithm 2: Needleman-Wunsch(A, B)

```
$g \leftarrow GapPenalty$;
for $i = 0$ to $length(A)$ do
    $S(i, 0) \leftarrow g \times i$;

for $j = 0$ to $length(B)$ do
    $S(0, j) \leftarrow g \times j$;

for $i = 1$ to $length(A)$ do
    for $j = 1$ to $length(B)$ do
        $Match \leftarrow S(i-1, j-1) + Scoring(A_i, B_j)$;
        $Insert \leftarrow S(i, j-1) + g$;
        $Delete \leftarrow S(i-1, j) + g$;
        $S(i, j) \leftarrow max(Match, Insert, Delete)$;

return $S(length(A), length(B))$
```

Scoring (**Gaps:** -2)

	C	**T**	**A**	**G**
C	+1	-1	-1	-1
T	-1	+1	-1	-1
A	-1	-1	+1	-1
G	-1	-1	-1	+1

Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	-2	-4	-6	-8	-10	-12	-14
	G	-2	1	-1	-3	-5	-7	-9	-11
	C	-4	-1	2	0	-2	-4	-6	-8
del ▶	C	-6	-3	0	1	1	-1	-3	-5
	T	-8	-5	-2	-1	0	2	0	-2
	G	-10	-7	-4	-3	-2	0	3	1
	A	-12	-9	-6	-3	-4	-2	1	4
ins ▶	T	-14	-11	-8	-5	-4	-3	-1	2

Global Alignment

Algorithm: Needleman Wunsch

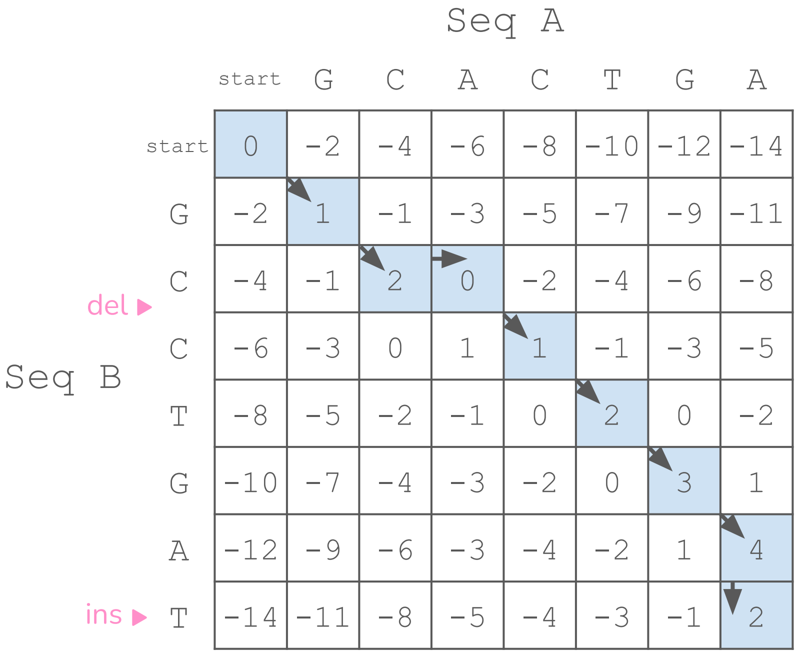
Scoring (**Gaps:** -2)

	C	**T**	**A**	**G**
C	+1	-1	-1	-1
T	-1	+1	-1	-1
A	-1	-1	+1	-1
G	-1	-1	-1	+1

Backtracking

- Starts bottom right cell
- Ends top left cell
- Reveals the alignment

```
GCACTGA-
GC-CTGAT
```



Seq A

Seq B

del ▶

ins ▶

	start	G	C	A	C	T	G	A
start	0	-2	-4	-6	-8	-10	-12	-14
G	-2	1	-1	-3	-5	-7	-9	-11
C	-4	-1	2	0	-2	-4	-6	-8
C	-6	-3	0	1	1	-1	-3	-5
T	-8	-5	-2	-1	0	2	0	-2
G	-10	-7	-4	-3	-2	0	3	1
A	-12	-9	-6	-3	-4	-2	1	4
T	-14	-11	-8	-5	-4	-3	-1	2

Local Alignment

Region of best local similarity

Some of Seq A
Some of Seq B

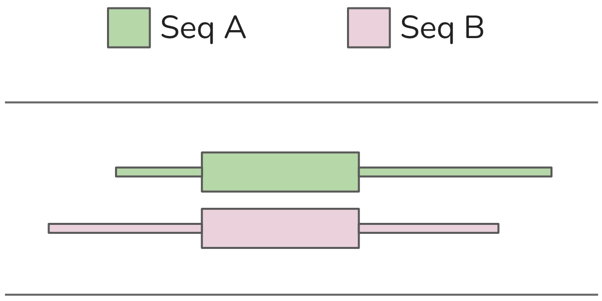
Best when sequences are dissimilar, but contain regions of similarity

eg. BLAST: gene homology

Best region dictated by penalty scores

Returns the alignment in highest scoring region

Algorithm: Smith Waterman



Gene homology between rat and human

Parts of the gene will be similar
(due to evolutionary viability)
(active sites, specific domains)

Parts of the gene will be dissimilar
(those which are more permissive to mutation)

Local Alignment

Region of best local similarity

Some of Seq A
Some of Seq B

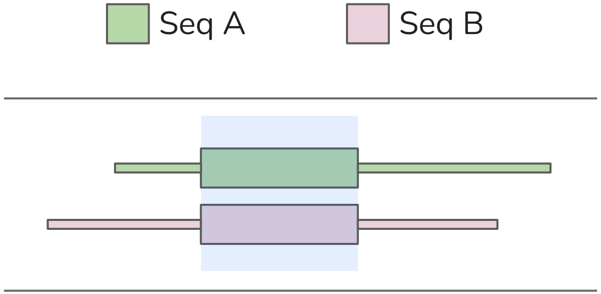
Best when sequences are dissimilar, but contain regions of similarity

eg. BLAST: gene homology

Best region dictated by penalty scores

Returns the alignment in highest scoring region

Algorithm: Smith Waterman



Gene homology between rat and human

Parts of the gene will be similar
(due to evolutionary viability)
(active sites, specific domains)

Parts of the gene will be dissimilar
(those which are more permissive to mutation)

Local Alignment

Algorithm: Smith Waterman

Same as Needleman-Wunsch except:

-> First row and first column set to 0

-> Negative score set to 0

-> the return score: *max(S)*

-> the backtracking:

Starts at *max(S)*

Ends when hit score of zero

Scoring (**Gaps:** -2)

	C	**T**	**A**	**G**
C	+2	-1	-1	-1
T	-1	+2	-1	-1
A	-1	-1	+2	-1
G	-1	-1	-1	+2

Seq A

Seq B

	start	G	C	A	C	T	G	A
start	0	0	0	0	0	0	0	0
G	0	2	0	0	0	0	2	0
del ▶ C	0	0	4	2	2	0	0	0
C	0	0	2	3	4	2	0	0
T	0	0	0	1	2	6	4	2
G	0	2	0	0	0	4	8	6
A	0	0	1	0	0	2	6	10
ins ▶ T	0	0	0	0	0	2	4	8

Local Alignment

Algorithm: Smith Waterman

Same as Needleman-Wunsch except:

-> First row and first column set to 0

-> Negative score set to 0

-> the return score: *max(S)*

-> the backtracking:

Starts at *max(S)*

Ends when hit score of zero

Scoring (**Gaps:** -2)

	C	**T**	**A**	**G**
C	+2	-1	-1	-1
T	-1	+2	-1	-1
A	-1	-1	+2	-1
G	-1	-1	-1	+2

Seq A

Seq B

	start	G	C	A	C	T	G	A
start	0	0	0	0	0	0	0	0
G	0	2	0	0	0	0	2	0
C	0	0	4	2	2	0	0	0
C	0	0	2	3	4	2	0	0
T	0	0	0	1	2	6	4	2
G	0	2	0	0	0	4	8	6
A	0	0	1	0	0	2	6	10
T	0	0	0	0	0	2	4	8

del ▶

ins ▶

Local Alignment

Algorithm: Smith Waterman

Same as Needleman-Wunsch except:

-> First row and first column set to 0

-> Negative score set to 0

-> the return score: *max(S)*

-> the backtracking:

Starts at *max(S)*

Ends when hit score of zero

第一行和第一列被初始化为0，这意味着比对可以在序列的任何位置开始，不必从头开始。任何得分为负的单元格都设置为0，这防止了得分的负累积，允许算法仅关注积极的匹配。返回的得分是矩阵中最大的得分（max(S)），这表示了最佳局部匹配的得分。回溯（backtracking）从得分矩阵中的最大值开始，当得分达到0时停止。这就找到了最佳的局部比对。

Scoring (**Gaps**: -2)

	C	**T**	**A**	**G**
C	+2	-1	-1	-1
T	-1	+2	-1	-1
A	-1	-1	+2	-1
G	-1	-1	-1	+2

Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	0	0	0	0	0	0	0
	G	0	2	0	0	0	0	2	0
del ▶	C	0	0	4	2	2	0	0	0
	C	0	0	2	3	4	2	0	0
	T	0	0	0	1	2	6	4	2
	G	0	2	0	0	0	4	8	6
	A	0	0	1	0	0	2	6	10
ins ▶	T	0	0	0	0	0	2	4	8

it s going to find region og simiilarity without much care for the alignment score was up to that point

Local Alignment

Algorithm: Smith Waterman

Same as Needleman-Wunsch except:

-> First row and first column set to 0

-> Negative score set to 0

-> the return score: *max(S)*

-> the backtracking:

Starts at *max(S)*

Ends when hit score of zero

Scoring (**Gaps:** -2)

	C	**T**	**A**	**G**
C	+2	-1	-1	-1
T	-1	+2	-1	-1
A	-1	-1	+2	-1
G	-1	-1	-1	+2

Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	0	0	0	0	0	0	0
	G	0	2	0	0	0	0	2	0
del ▶	C	0	0	4	2	2	0	0	0
	C	0	0	2	3	4	2	0	0
	T	0	0	0	1	2	6	4	2
	G	0	2	0	0	0	4	8	6
	A	0	0	1	0	0	2	6	10
ins ▶	T	0	0	0	0	0	2	4	8

Local Alignment

如果执行回溯，我们会跟随得分的来源直到得分为零，这将确定局部比对中涉及的确切序列片段

Algorithm: Smith Waterman

Same as Needleman-Wunsch except:

-> First row and first column set to 0

-> Negative score set to 0

-> the return score: *max(S)*

-> the backtracking:

Starts at *max(S)*

Ends when hit score of zero

```
GCACTGA
GC-CTGA
```

Scoring (**Gaps:** -2)

	C	**T**	**A**	**G**
C	+2	-1	-1	-1
T	-1	+2	-1	-1
A	-1	-1	+2	-1
G	-1	-1	-1	+2

Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	0	0	0	0	0	0	0
	G	0	2	0	0	0	0	2	0
del ▶	C	0	0	4	2	2	0	0	0
	C	0	0	2	3	4	2	0	0
	T	0	0	0	1	2	6	4	2
	G	0	2	0	0	0	4	8	6
	A	0	0	1	0	0	2	6	10
ins ▶	T	0	0	0	0	0	2	4	8

Local Alignment

Algorithm: Smith Waterman

Same as Needleman-Wunsch except:

-> First row and first column set to 0

-> Negative score set to 0

-> the return score: *max(S)*

-> the backtracking:

Starts at *max(S)*

Ends when hit score of zero

```
ACTGA
CCTGA
```

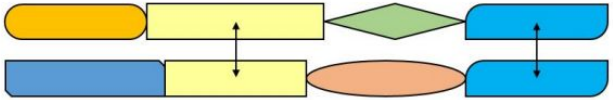
Scoring (**Gaps:** -2)

	C	**T**	**A**	**G**
C	+2	-1	-1	-1
T	-1	+2	-1	-1
A	-1	-1	+2	-1
G	-1	-1	-1	+2

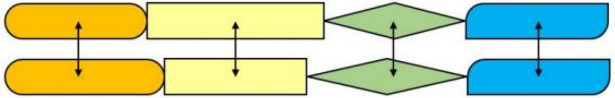
Seq A

		start	G	C	A	C	T	G	A
	start	0	0	0	0	0	0	0	0
	G	0	2	0	0	0	0	2	0
	C	0	0	4	0	2	0	0	0
del ▶	C	0	0	2	3	4	2	0	0
Seq B	T	0	0	0	1	2	6	4	2
	G	0	2	0	0	0	4	8	6
	A	0	0	1	0	0	2	6	10
ins ▶	T	0	0	0	0	0	2	4	8

Local *vs.* Global



Local Alignment



Global Alignment

	Smith–Waterman algorithm	Needleman-Wunsch algorithm
Initialization	First row and first column are set to 0	First row and first column are subject to gap penalty
Scoring	Negative score is set to 0	Score can be negative
Traceback	Begin with the highest score, end when 0 is encountered	Begin with the cell at the lower right of the matrix, end at top left cell
Complexity	$O(m \times n)$	$O(m \times n)$

Semi-global Alignment

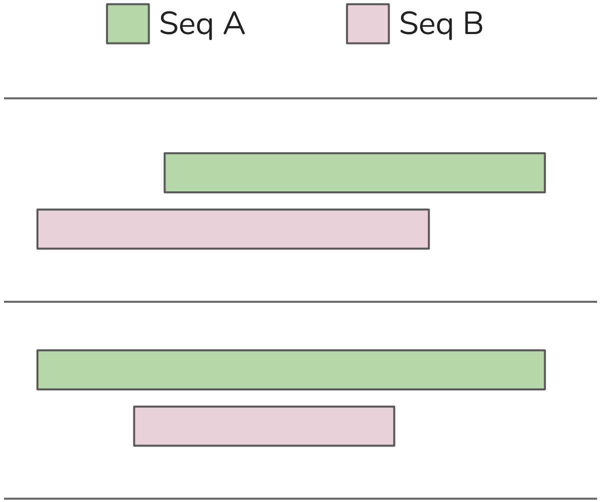
Alignment of complete sequences, where offset is not penalised

All of Seq A
All of Seq B

Best when sequences are expected to have similar overlapping region

eg. Read overlaps in OLC assembly

Returns the full alignment, clipped by best offset



Semi-global Alignment

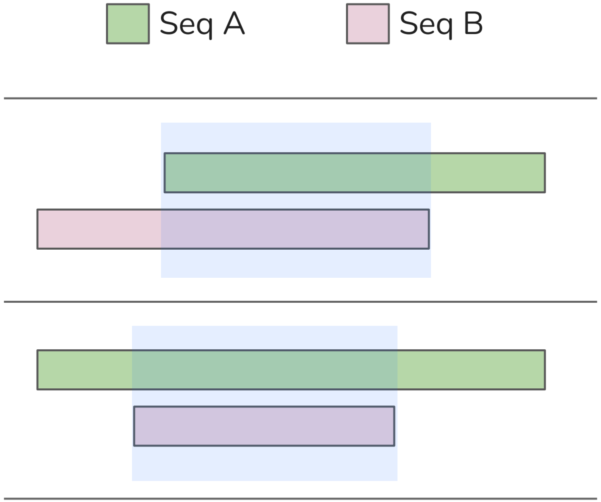
Alignment of complete sequences, where offset is not penalised

All of Seq A
All of Seq B

Best when sequences are expected to have similar overlapping region

eg. Read overlaps in OLC assembly

Returns the full alignment, clipped by best offset



Semi-global Alignment

Algorithm: Needleman Wunsch (variant)

Same as Needleman-Wunsch except:

-> no offset penalties for top row, left column

-> the return score:

Max(bottom row), or *Max(right column)*

-> the backtracking:

Starts at max cell

Ends when hit top row, or left column

Scoring (**Gaps:** -2)

	C	**T**	**A**	**G**
C	+1	-1	-1	-1
T	-1	+1	-1	-1
A	-1	-1	+1	-1
G	-1	-1	-1	+1

Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	0	0	0	0	0	0	0
	G	0	1	-1	-1	-1	-1	1	-1
del ▶	C	0	-1	2	0	0	-1	-1	0
	C	0	-1	0	1	1	-1	-2	-2
	T	0	-1	-2	-1	0	2	0	-2
	G	0	1	-1	-3	-1	0	3	1
	A	0	-1	0	0	-2	-2	1	4
ins ▶	T	0	-1	-2	-1	-1	-1	-1	2

Semi-global Alignment

Algorithm: Needleman Wunsch (variant)

Same as Needleman-Wunsch except:

-> no offset penalties for top row, left column

-> the return score:

Max(bottom row), or *Max(right column)*

-> the backtracking:

Starts at max cell

Ends when hit top row, or left column

Scoring (**Gaps:** -2)

	C	**T**	**A**	**G**
C	+1	-1	-1	-1
T	-1	+1	-1	-1
A	-1	-1	+1	-1
G	-1	-1	-1	+1

Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	0	0	0	0	0	0	0
	G	0	1	-1	-1	-1	-1	1	-1
del ▶	C	0	-1	2	0	0	-1	-1	0
	C	0	-1	0	1	1	-1	-2	-2
	T	0	-1	-2	-1	0	2	0	-2
	G	0	1	-1	-3	-1	0	3	1
	A	0	-1	0	0	-2	-2	1	4
ins ▶	T	0	-1	-2	-1	-1	-1	-1	2

Semi-global Alignment

Algorithm: Needleman Wunsch (variant)

Same as Needleman-Wunsch except:

-> no offset penalties for top row, left column

-> the return score:

Max(bottom row), or *Max(right column)*

-> the backtracking:

Starts at max cell

Ends when hit top row, or left column

Scoring (**Gaps:** -2)

	C	**T**	**A**	**G**
C	+1	-1	-1	-1
T	-1	+1	-1	-1
A	-1	-1	+1	-1
G	-1	-1	-1	+1

Seq A

Seq B

		start	G	C	A	C	T	G	A
	start	0	0	0	0	0	0	0	0
	G	0	1	-1	-1	-1	-1	1	-1
del ▶	C	0	-1	2	0	0	-1	-1	0
	C	0	-1	0	1	1	-1	-2	-2
	T	0	-1	-2	-1	0	2	0	-2
	G	0	1	-1	-3	-1	0	3	1
	A	0	-1	0	0	-2	-2	1	4
ins ▶	T	0	-1	-2	-1	-1	-1	-1	2

Semi-global Alignment

Algorithm: Needleman Wunsch (variant)

Same as Needleman-Wunsch except:

-> no offset penalties for top row, left column

-> the return score:

Max(bottom row), or *Max(right column)*

-> the backtracking:

Starts at max cell

Ends when hit top row, or left column

```
GCACTGA
GC-CTGA
```

Scoring (**Gaps:** -2)

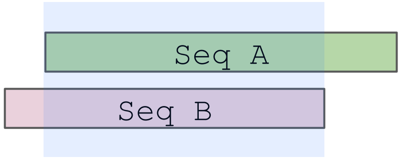
	C	**T**	**A**	**G**
C	+1	-1	-1	-1
T	-1	+1	-1	-1
A	-1	-1	+1	-1
G	-1	-1	-1	+1

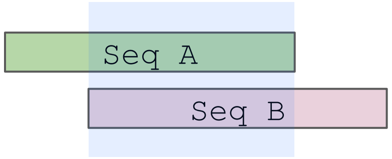
Seq A

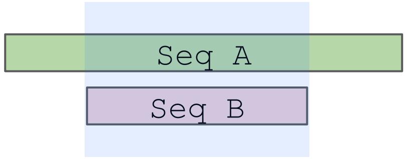
Seq B

	start	G	C	A	C	T	G	A
start	0	0	0	0	0	0	0	0
G	0	1	-1	-1	-1	-1	1	-1
del ▶ C	0	-1	2	0	0	-1	-1	0
C	0	-1	0	1	1	-1	-2	-2
T	0	-1	-2	-1	0	2	0	-2
G	0	1	-1	-3	-1	0	3	1
A	0	-1	0	0	-2	-2	1	4
ins ▶ T	0	-1	-2	-1	-1	-1	-1	2

Semi-global Alignment







Seq A

Seq B

	start	G	C	A	C	T	G	A
start	0	0	0	0	0	0	0	0
G	0							
C	0							
C	0							
T	0							
G	0							
A	0							
T	0					max		

Seq A

	start	G	C	A	C	T	G	A
start	0	0	0	0	0	0	0	0
G	0							
C	0							
C	0							max
T	0							
G	0							
A	0							
T	0							

Seq A

	start	G	C	A	C	T	G	A	T	A
start	0	0	0	0	0	0	0	0	0	0
G	0									
C	0									
C	0									
T	0							max		

Pairwise Alignment

Local Alignment

Finds region of best local similarity

O(m x n)

Global Alignment

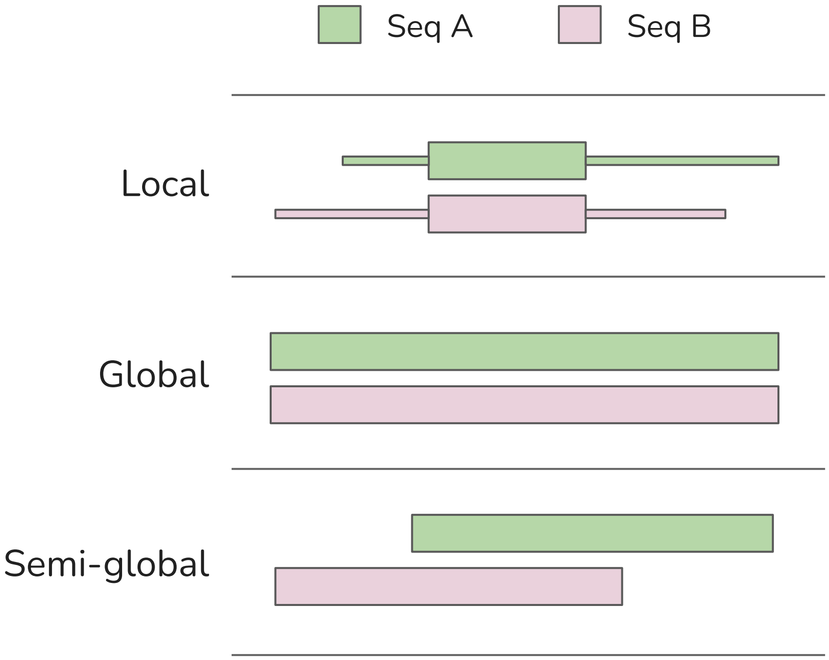
End-to-end alignment of two sequences

O(m x n)

Semi-Global Alignment

Alignment of complete sequences, where offset is not penalised

O(m x n)



Does this scale to large datasets?

Pairwise Alignment

These first two weeks underpin:

- Phylogenetics
- Protein function
- Conservation
- Sequence database searching
- De novo genome assembly
- Genetic variant detection

Huge proportion of bioinformatics

Week 1B

Indexing & kmers

Week 2A

Sequence Alignment

Week 2B

Putting it all together - efficient sequence mapping

Thank you!

Don’t forget your signed academic integrity statement

Today: Sequence Alignment I

Next time: Sequencing Alignment II