GENOME ANALYSIS & BIOINFORMATICS

Week 3:

How can we align two sequences?

DNA Sequence Alignment I: Motivation

You are studying a recently discovered human non-coding RNA.

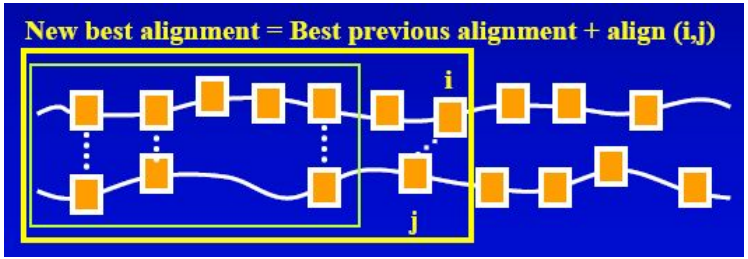
You search it against the mouse genome using BLASTN (N for nucleotide) and obtain the following alignment:

```
Q: 1    ttgacctagatgagatgtcgttcacttttactcaggtacagaaaa 45
        ||||  |||||||||||||  | ||||||||||||  ||  |||||||||
S: 403  ttgatctagatgagatgccattcacttttactgagctacagaaaa 447
```

Is this alignment significant?
Is this likely to represent a homologous RNA?

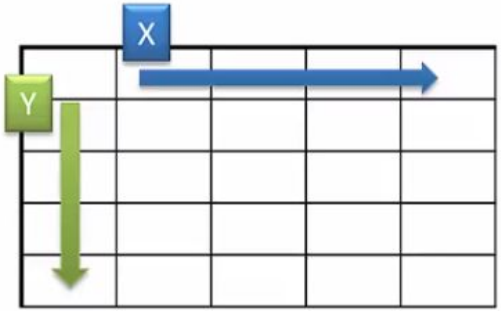
How to find alignments?

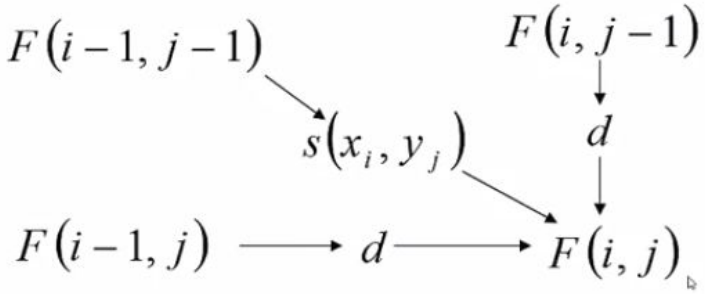
S	S	–
T	–	T

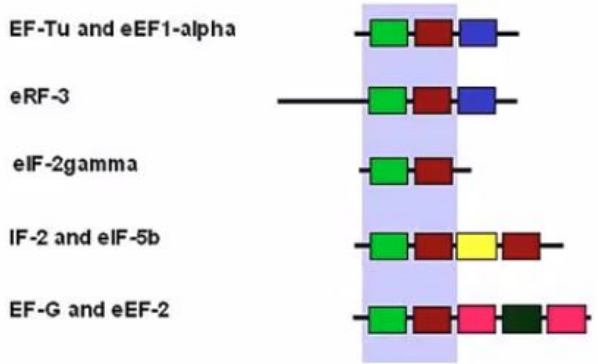


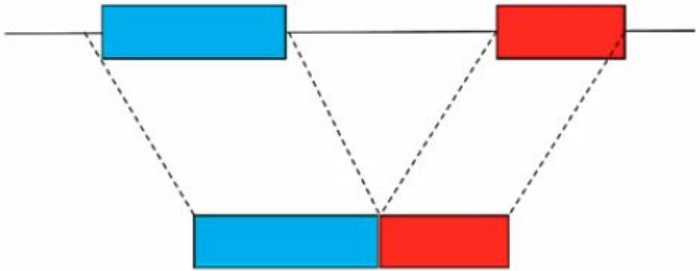
$$F(0,0) = 0$$

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) & x_i \text{ aligned to } y_j \\ F(i-1, j) + d & x_i \text{ aligned to } a\ gap \\ F(i, j-1) + d & y_j \text{ aligned to } a\ gap \end{cases}$$






EF-Tu and eEF1-alpha
eRF-3
eIF-2gamma
IF-2 and eIF-5b
EF-G and eEF-2



Local alignment: example

Scoring

Match: +2

Mismatch: -1

Indel: -2

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

```
C T - A A
C T C A A
```

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

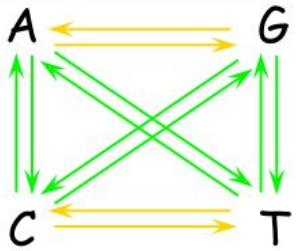
Identity

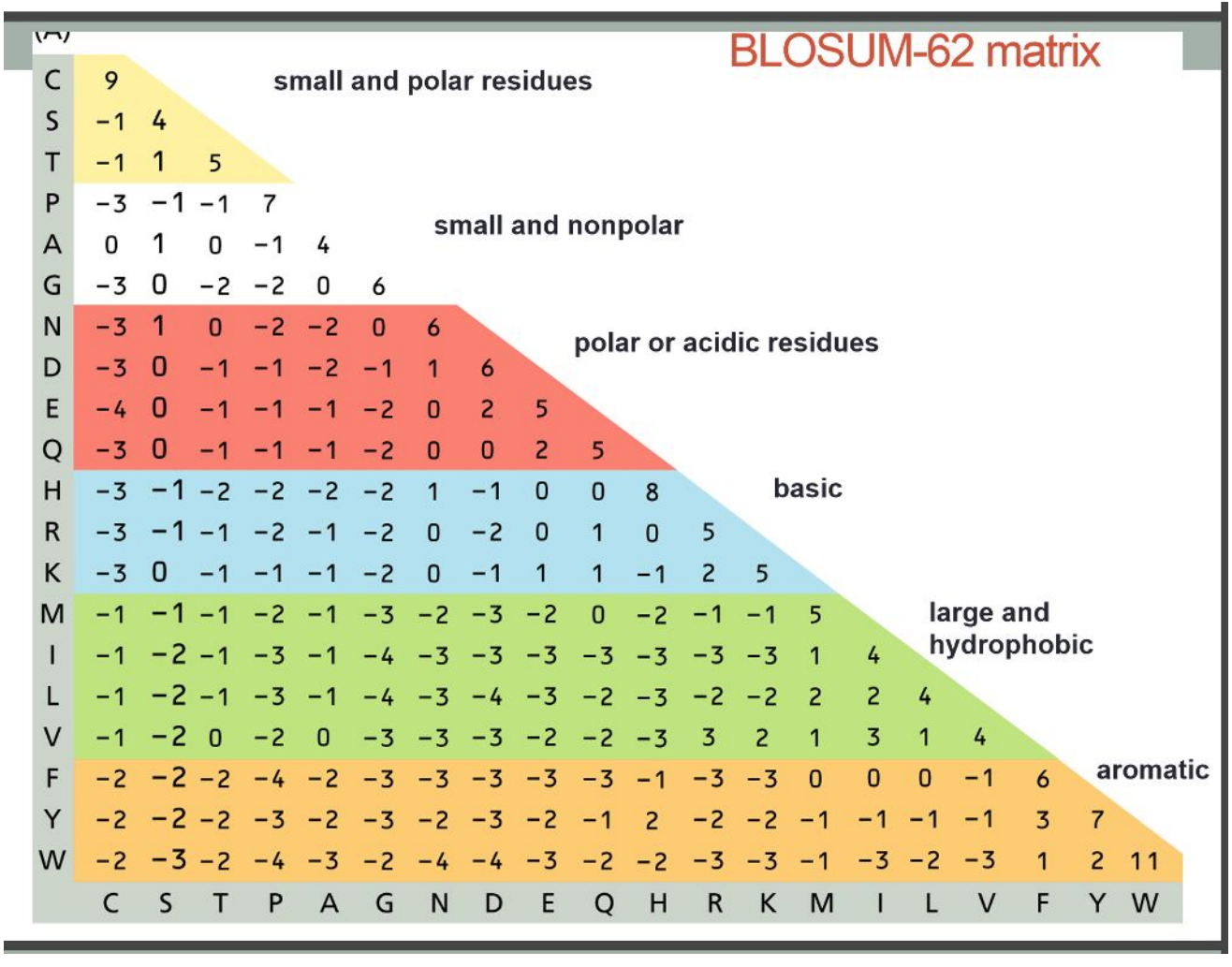
	A	T	C	G
A	5	-4	-4	-4
T	-4	5	-4	-4
C	-4	-4	5	-4
G	-4	-4	-4	5

BLAST

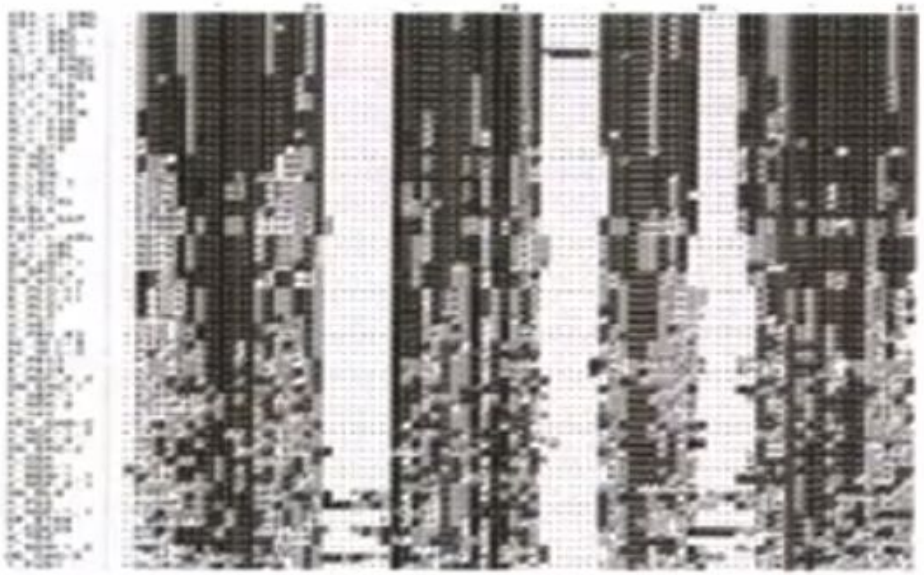
	A	T	C	G
A	0	5	5	1
T	5	0	1	5
C	5	1	0	5
G	1	5	5	0

Transition/
Transversion

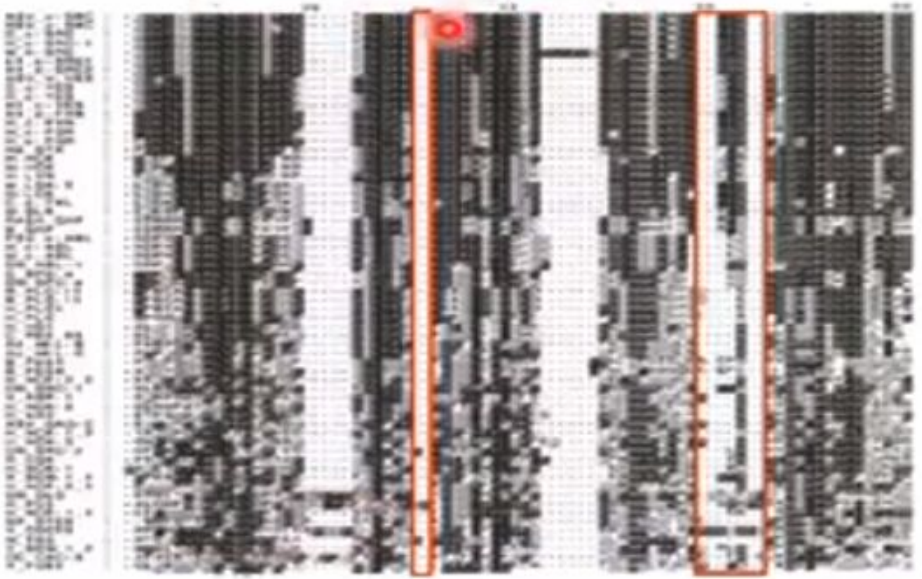


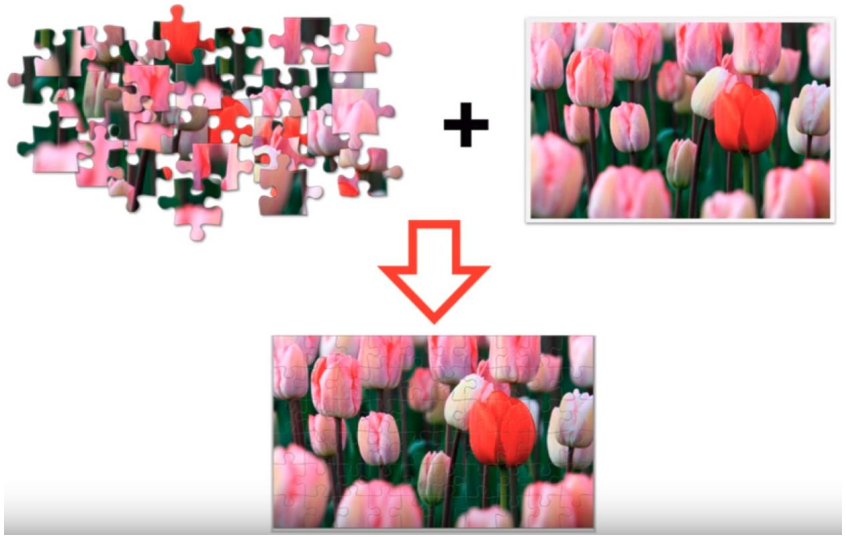

BLOSUM-62 matrix
small and polar residues
small and nonpolar
polar or acidic residues
basic
large and hydrophobic
aromatic
C 9
S -1 4
T -1 1 5
P -3 -1 -1 7
A 0 1 0 -1 4
G -3 0 -2 -2 0 6
N -3 1 0 -2 -2 0 6
D -3 0 -1 -1 -2 -1 1 6
E -4 0 -1 -1 -1 -2 0 2 5
Q -3 0 -1 -1 -1 -2 0 0 2 5
H -3 -1 -2 -2 -2 -2 1 -1 0 0 8
R -3 -1 -1 -2 -1 -2 0 -2 0 1 0 5
K -3 0 -1 -1 -1 -2 0 -1 1 1 -1 2 5
M -1 -1 -1 -2 -1 -3 -2 -3 -2 0 -2 -1 -1 5
I -1 -2 -1 -3 -1 -4 -3 -3 -3 -3 -3 -3 -3 1 4
L -1 -2 -1 -3 -1 -4 -3 -4 -3 -2 -3 -2 -2 2 2 4
V -1 -2 0 -2 0 -3 -3 -3 -2 -2 -3 3 2 1 3 1 4
F -2 -2 -2 -4 -2 -3 -3 -3 -3 -3 -1 -3 -3 0 0 0 -1 6
Y -2 -2 -2 -3 -2 -3 -2 -3 -2 -1 2 -2 -2 -1 -1 -1 -1 3 7
W -2 -3 -2 -4 -3 -2 -4 -4 -3 -2 -2 -3 -3 -1 -3 -2 -3 1 2 11
C S T P A G N D E Q H R K M I L V F Y W

best parameters



default parameters



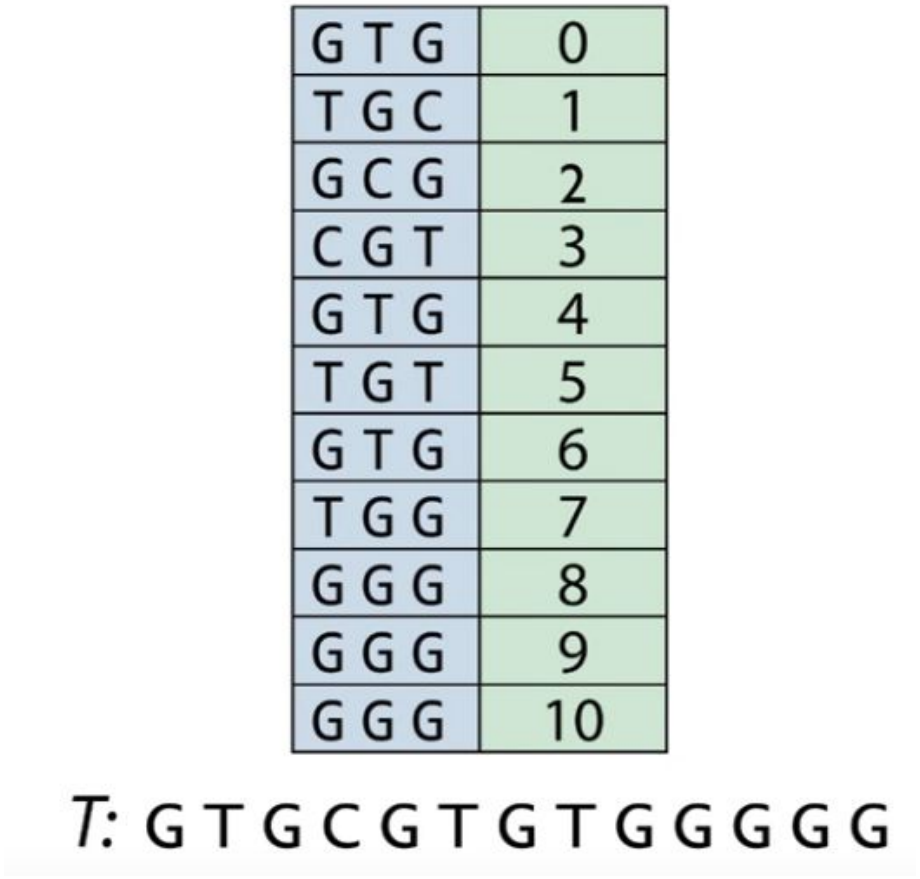


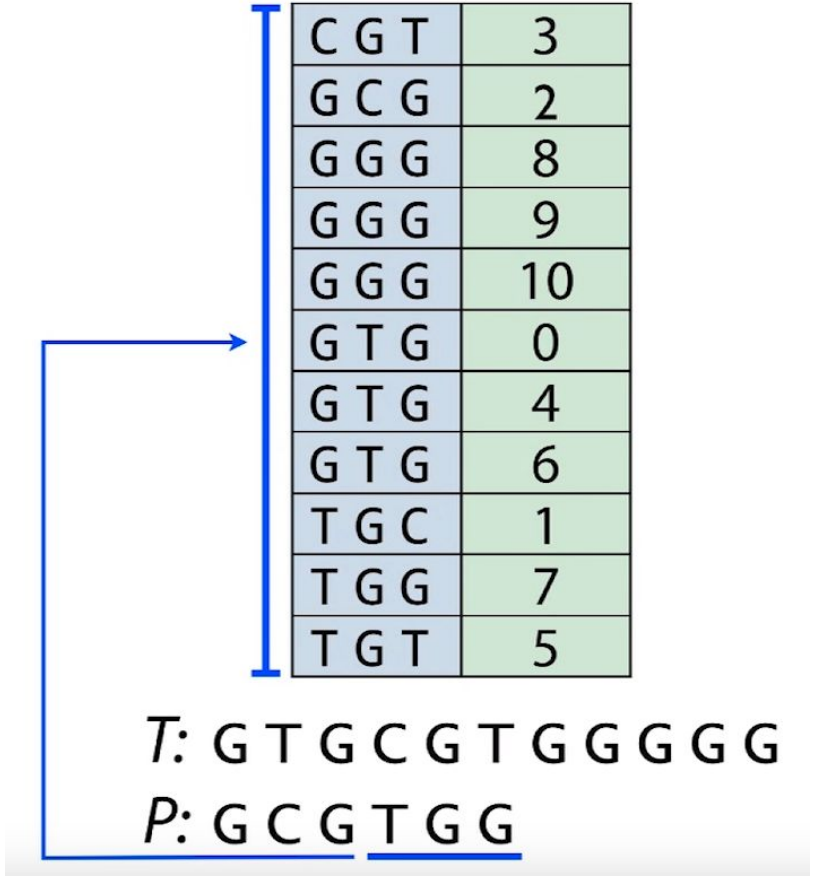
Read

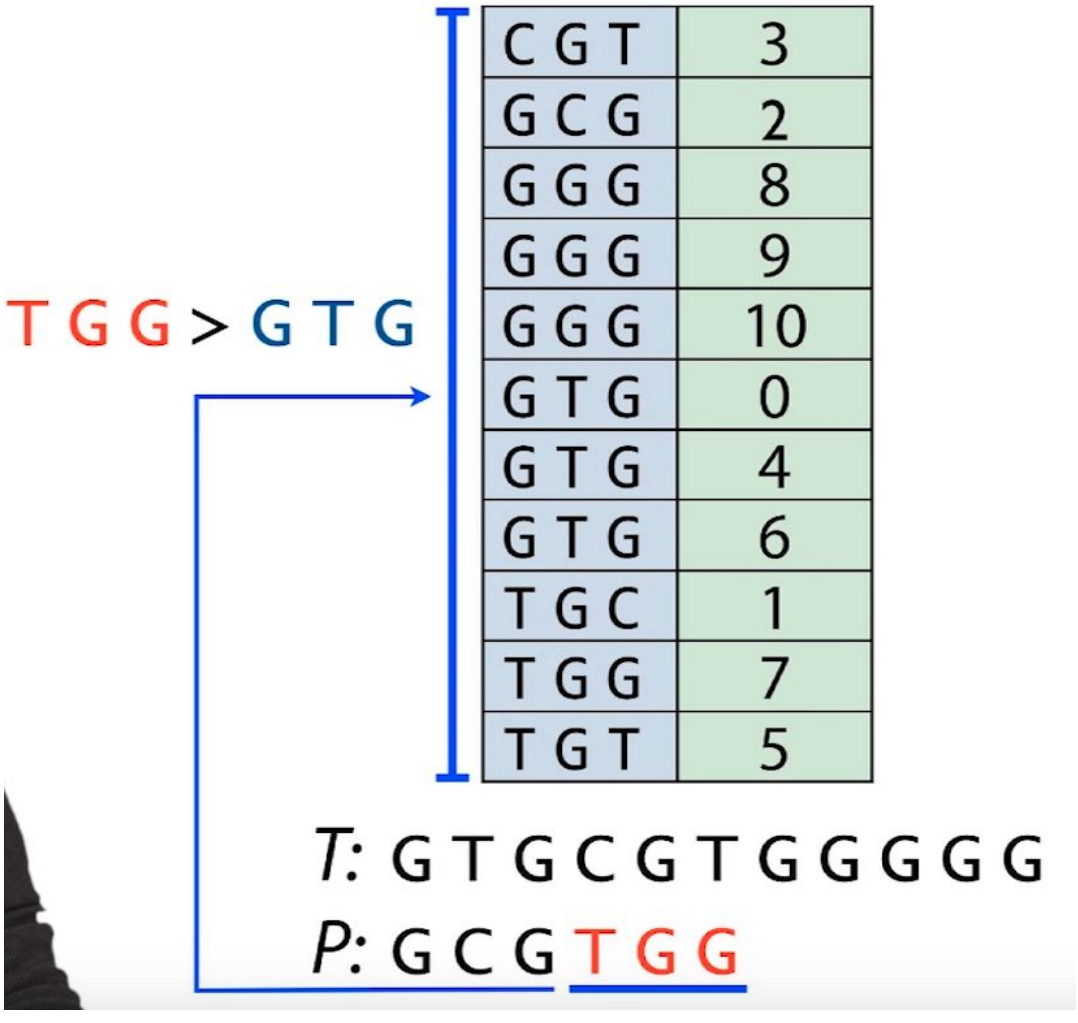
CTCAAACTCCTGACCTTTGGTGATCCACCCGCCTAGGCCTTC x million

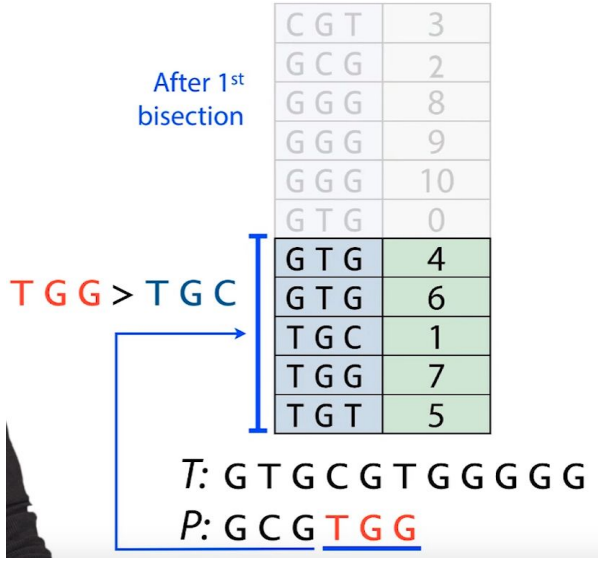
Reference

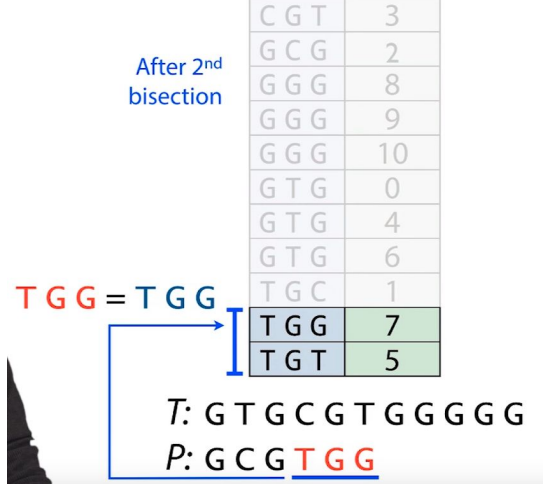
x million


G T G 0
T G C 1
G C G 2
C G T 3
G T G 4
T G T 5
G T G 6
T G G 7
G G G 8
G G G 9
G G G 10
T: G T G C G T G T G G G G G


C G T 3
G C G 2
G G G 8
G G G 9
G G G 10
G T G 0
G T G 4
G T G 6
T G C 1
T G G 7
T G T 5
T: G T G C G T G G G G G
P: G C G T G G

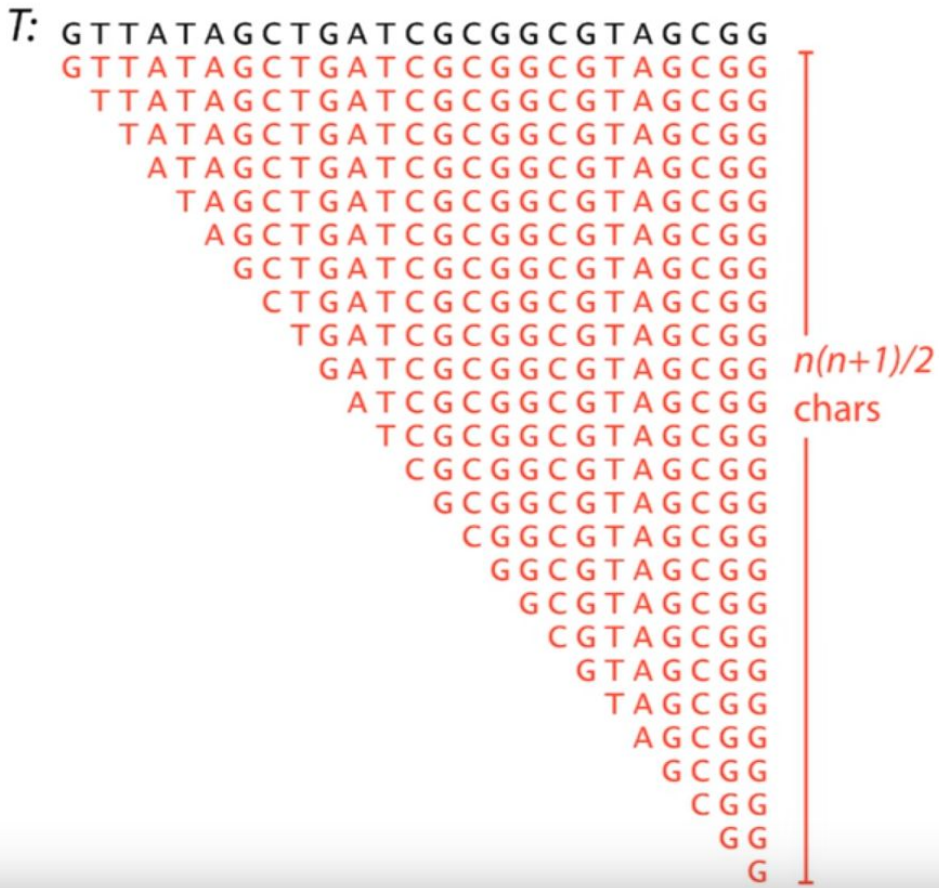

T G G > G T G
C G T 3
G C G 2
G G G 8
G G G 9
G G G 10
G T G 0
G T G 4
G T G 6
T G C 1
T G G 7
T G T 5
T: G T G C G T G G G G G
P: G C G T G G


After 1st bisection
C G T 3
G C G 2
G G G 8
G G G 9
G G G 10
G T G 0
T G G > T G C
G T G 4
G T G 6
T G C 1
T G G 7
T G T 5
T: G T G C G T G G G G G
P: G C G T G G


After 2nd bisection
C G T 3
G C G 2
G G G 8
G G G 9
G G G 10
G T G 0
G T G 4
G T G 6
T G C 1
T G G = T G G
T G G 7
T G T 5
T: G T G C G T G G G G G
P: G C G T G G

Suffix index

Suffix index



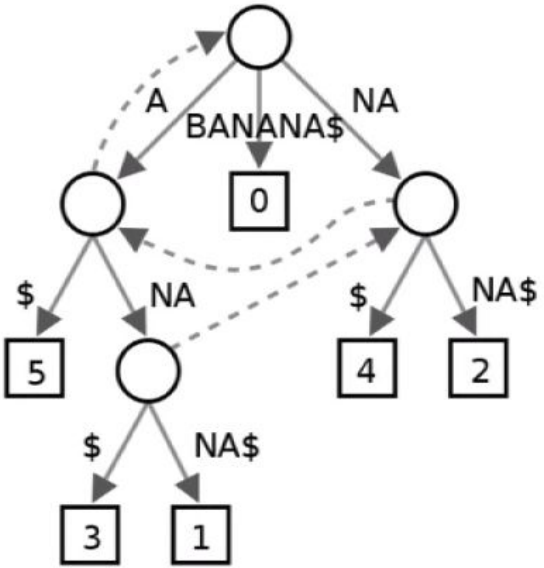
T = abaaba$
0123456

As with suffix tree,
T is part of index

SA(*T*) =

6	$
5	a $
2	a a b a $
3	a b a $
0	a b a a b a $
4	b a $
1	b a a b a $

m integers



6	$
5	A$
3	ANA$
1	ANANA$
0	BANANA$
4	NA$
2	NANA$

$ B A N A N A
A $ B A N A N
A N A $ B A N
A N A N A $ B
B A N A N A $
N A $ B A N A
N A N A $ B A

Suffix tree

≥ 45 GB

Suffix array

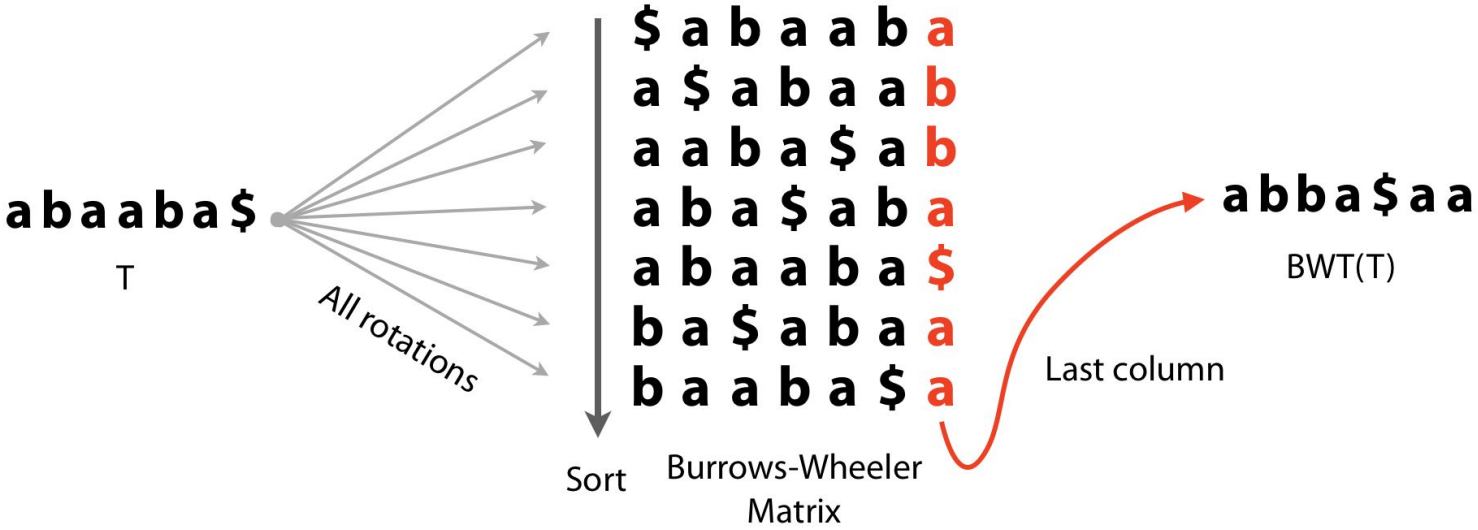
≥ 12 GB

FM Index

~ 1 GB

Burrows-Wheeler Transform

Reversible permutation of the characters of a string, used originally for compression



How is it useful for compression? How is it reversible? How is it an index?

Burrows-Wheeler Transform: T-ranking

Give each character in *T* a rank, equal to # times the character occurred previously in *T*. Call this the *T-ranking*.

$$\mathbf{a_0\ b_0\ a_1\ a_2\ b_1\ a_3\ \$}$$

Now let's re-write the BWM including ranks...

Burrows-Wheeler Transform

BWM with T-ranking:

F	*L*
$ a_0 b_0 a_1 a_2 b_1	**a_3**
a_3 $ a_0 b_0 a_1 a_2	b_1
a_1 a_2 b_1 a_3 $ a_0	b_0
a_2 b_1 a_3 $ a_0 b_0	**a_1**
a_0 b_0 a_1 a_2 b_1 a_3	$
b_1 a_3 $ a_0 b_0 a_1	**a_2**
b_0 a_1 a_2 b_1 a_3 $	**a_0**

Look at first and last columns, called *F* and *L*

And look at just the **a**s

as occur in the same order in *F* and *L*. As we look down columns, in both cases we see: **a_3**, **a_1**, **a_2**, **a_0**

Burrows-Wheeler Transform

BWM with T-ranking:

F	*L*
$ a_0 b_0 a_1 a_2 b_1 a_3	
a_3 $ a_0 b_0 a_1 a_2	**b_1**
a_1 a_2 b_1 a_3 $ a_0	**b_0**
a_2 b_1 a_3 $ a_0 b_0 a_1	
a_0 b_0 a_1 a_2 b_1 a_3 $	
b_1 a_3 $ a_0 b_0 a_1 a_2	
b_0 a_1 a_2 b_1 a_3 $ a_0	

Same with **b**s: **b_1**, **b_0**

Burrows-Wheeler Transform: LF Mapping

BWM with T-ranking:

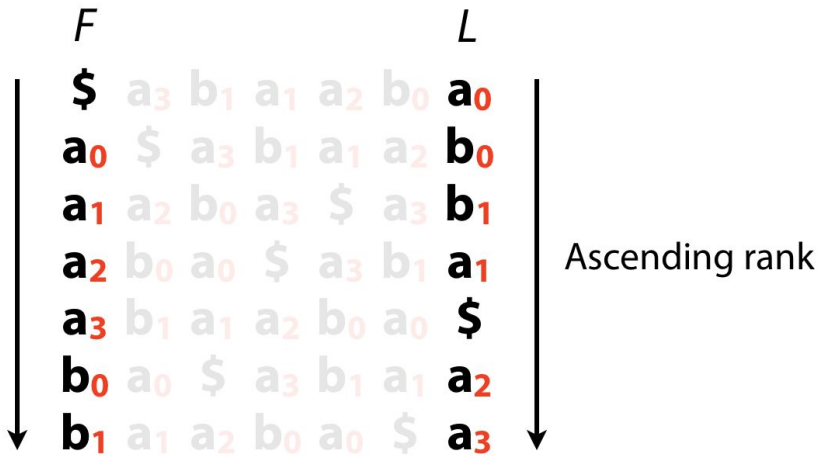
F						*L*
$	a_0	b_0	a_1	a_2	b_1	a_3
a_3	$	a_0	b_0	a_1	a_2	b_1
a_1	a_2	b_1	a_3	$	a_0	b_0
a_2	b_1	a_3	$	a_0	b_0	a_1
a_0	b_0	a_1	a_2	b_1	a_3	$
b_1	a_3	$	a_0	b_0	a_1	a_2
b_0	a_1	a_2	b_1	a_3	$	a_0

LF Mapping: The i^{th} occurrence of a character *c* in *L* and the i^{th} occurrence of *c* in *F* correspond to the *same* occurrence in *T*

However we rank occurrences of *c*, ranks appear in the same order in *F* and *L*

Burrows-Wheeler Transform: LF Mapping

BWM with B-ranking:



F now has very simple structure: a **$**, a block of **a**s *with ascending ranks*, a block of **b**s *with ascending ranks*

Burrows-Wheeler Transform: reversing

Reverse BWT(T) starting at right-hand-side of *T* and moving left

Start in first row. *F* must have **$**. *L* contains character just prior to **$**: $\mathbf{a_0}$

$\mathbf{a_0}$: LF Mapping says this is same occurrence of **a** as first **a** in *F*. Jump to row *beginning* with $\mathbf{a_0}$. *L* contains character just prior to $\mathbf{a_0}$: $\mathbf{b_0}$.

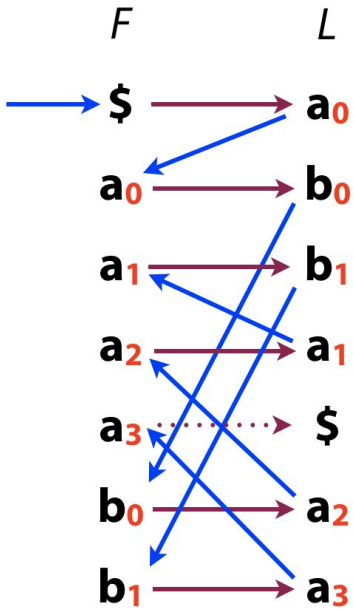
Repeat for $\mathbf{b_0}$, get $\mathbf{a_2}$

Repeat for $\mathbf{a_2}$, get $\mathbf{a_1}$

Repeat for $\mathbf{a_1}$, get $\mathbf{b_1}$

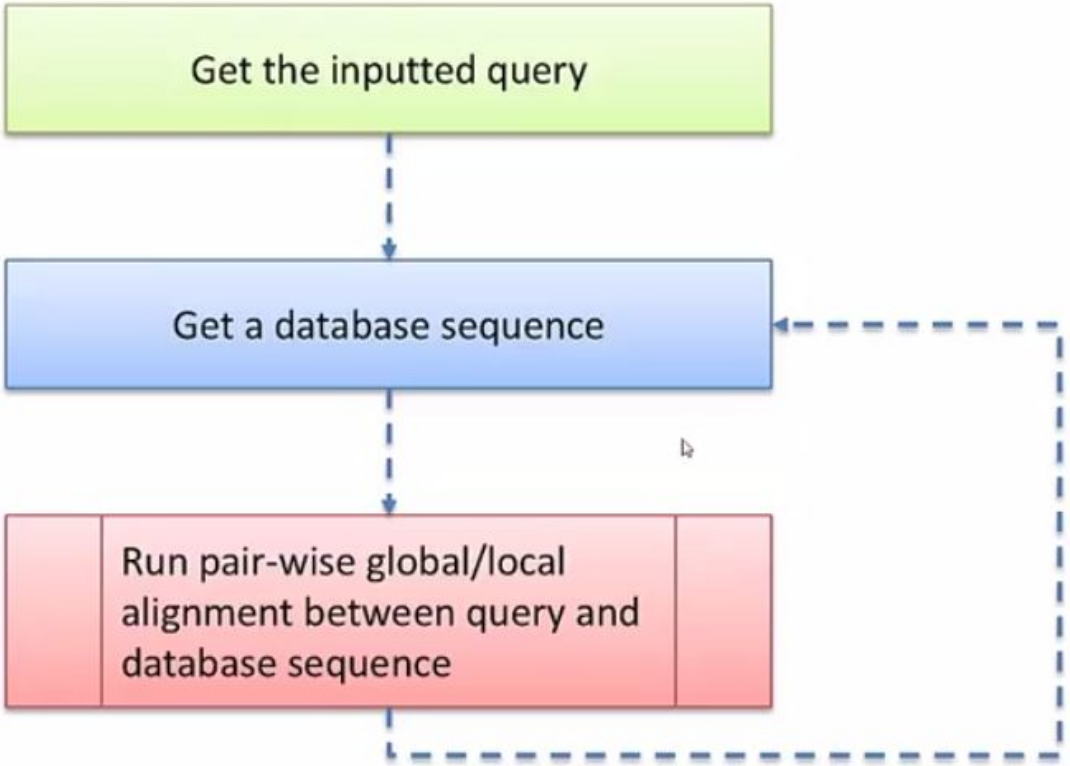
Repeat for $\mathbf{b_1}$, get $\mathbf{a_3}$

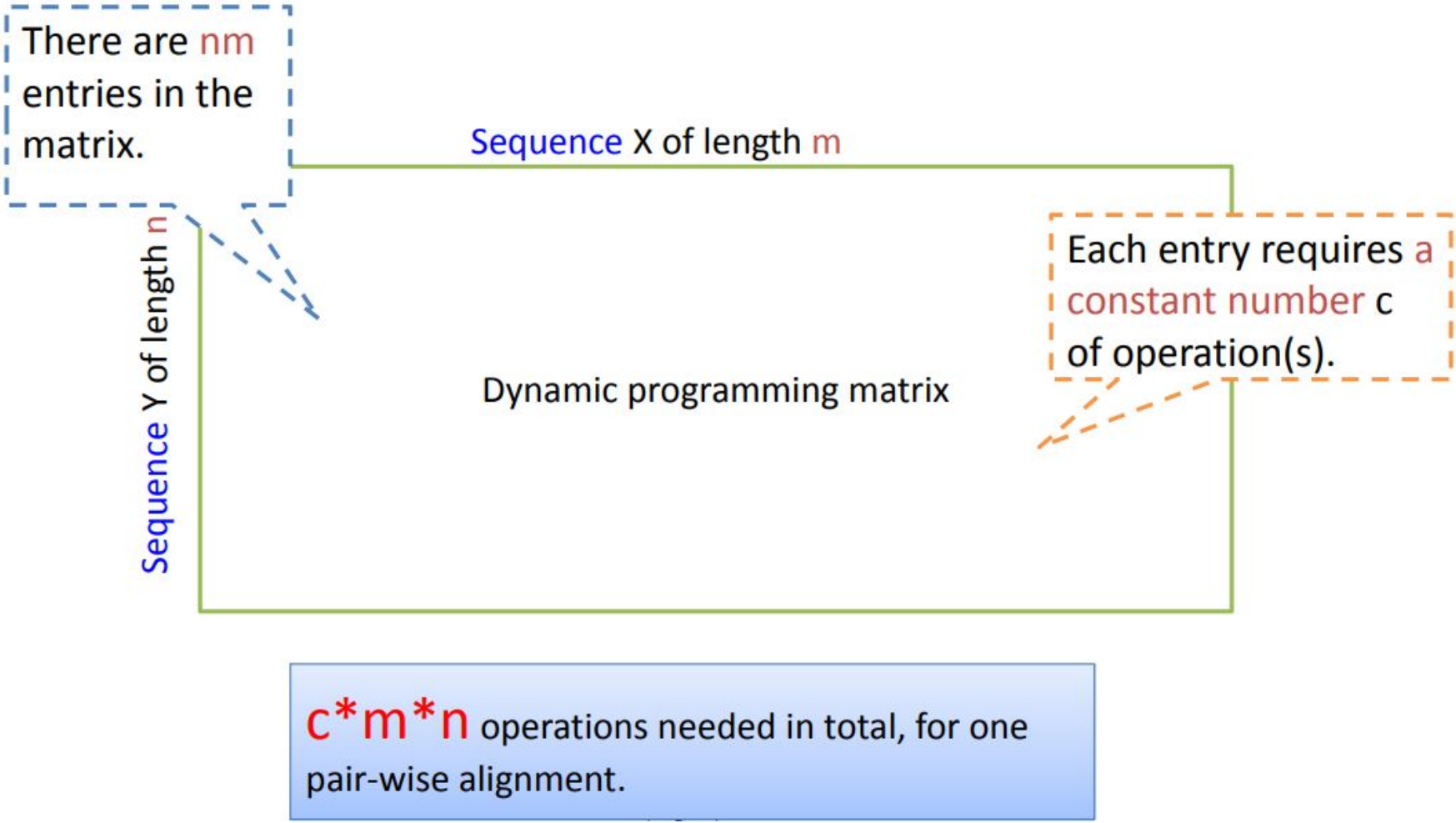
Repeat for $\mathbf{a_3}$, get **$**, done



Reverse of chars we visited = $\mathbf{a_3\ b_1\ a_1\ a_2\ b_0\ a_0\ \$} = T$

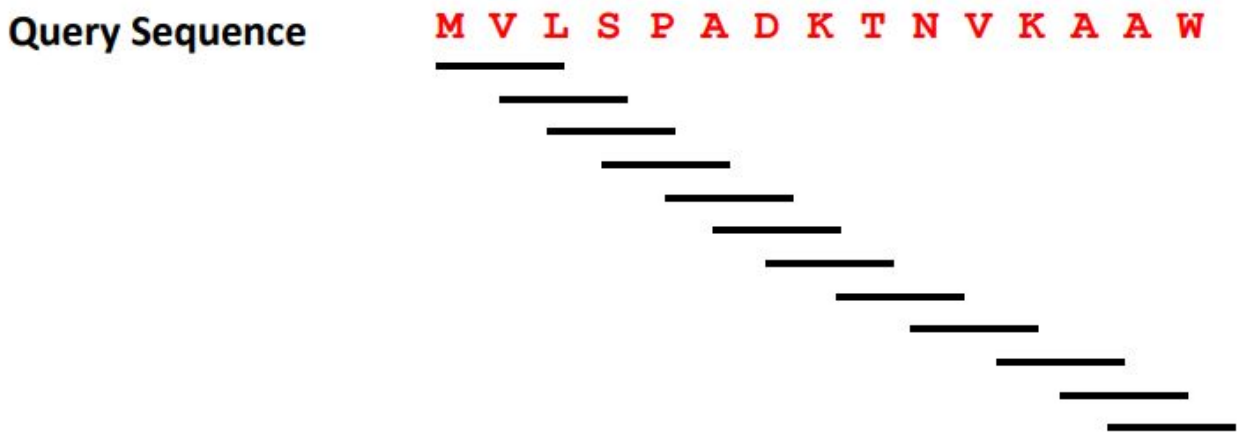
A (naïve) algorithm for database searching




There are nm entries in the matrix.
Sequence X of length m
Sequence Y of length n
Each entry requires a constant number c of operation(s).
Dynamic programming matrix
c*m*n operations needed in total, for one pair-wise alignment.

Seeding

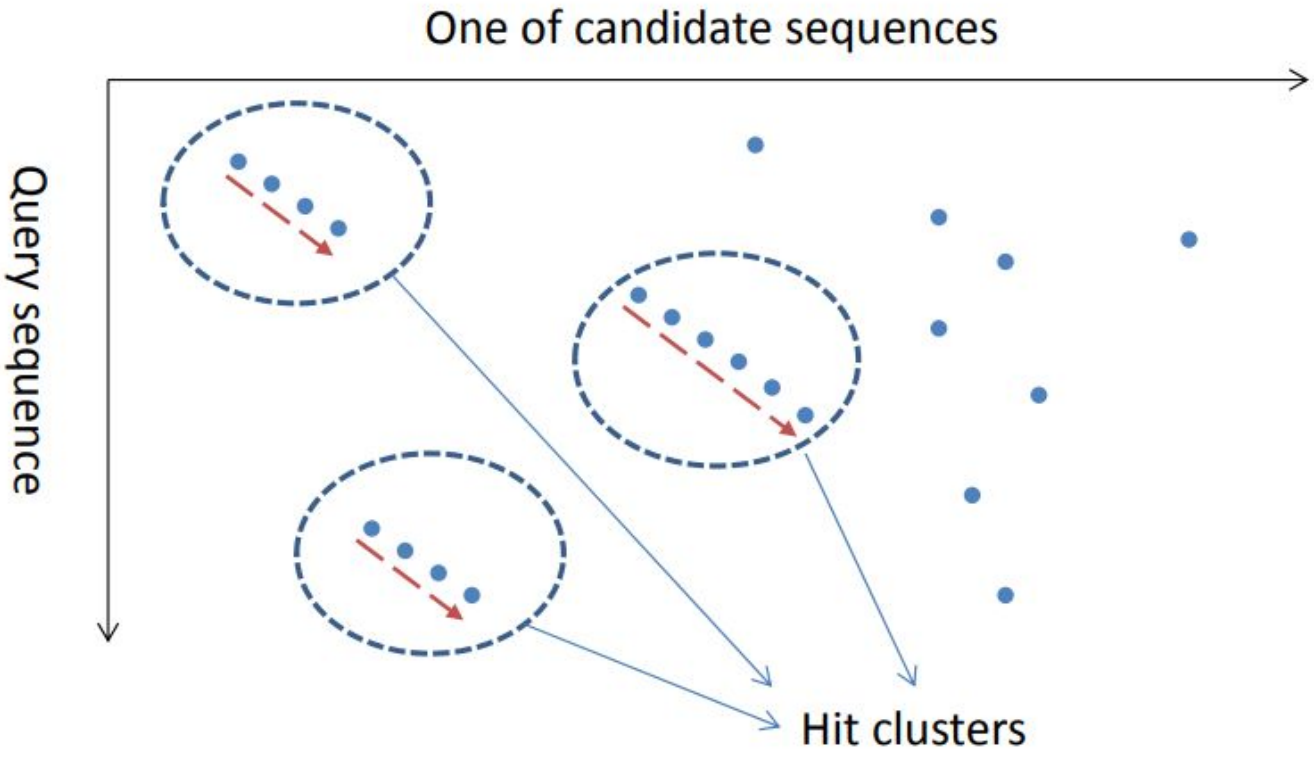
For a given word length *w* (usually 3 for proteins and 11 for nucleotides), slicing the query sequence into multiple continuous "seed words"

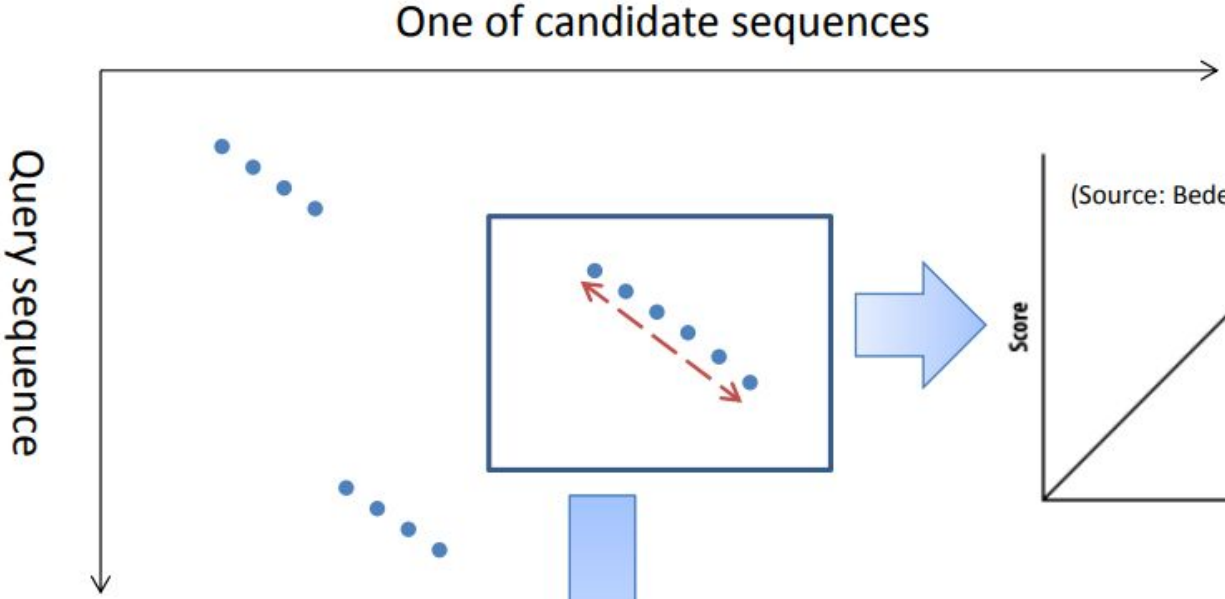


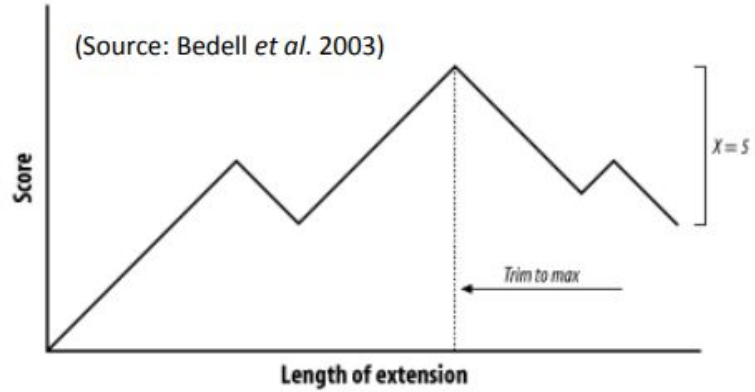
Speedup: Index database

The database was pre-indexed to quickly locate all positions in the database for a given seed.

Diagonal *and* Two-hits





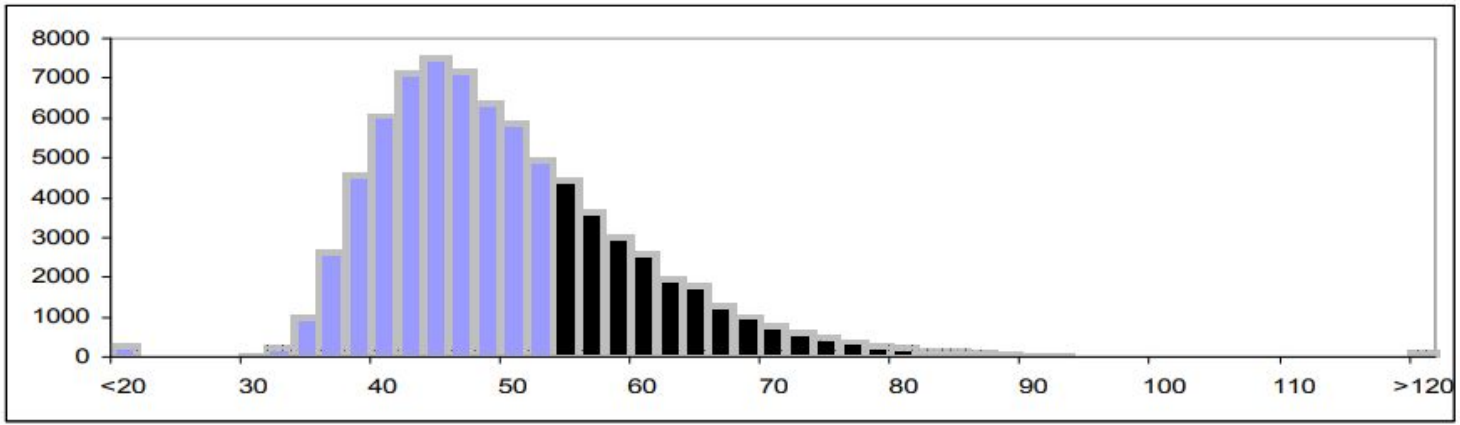


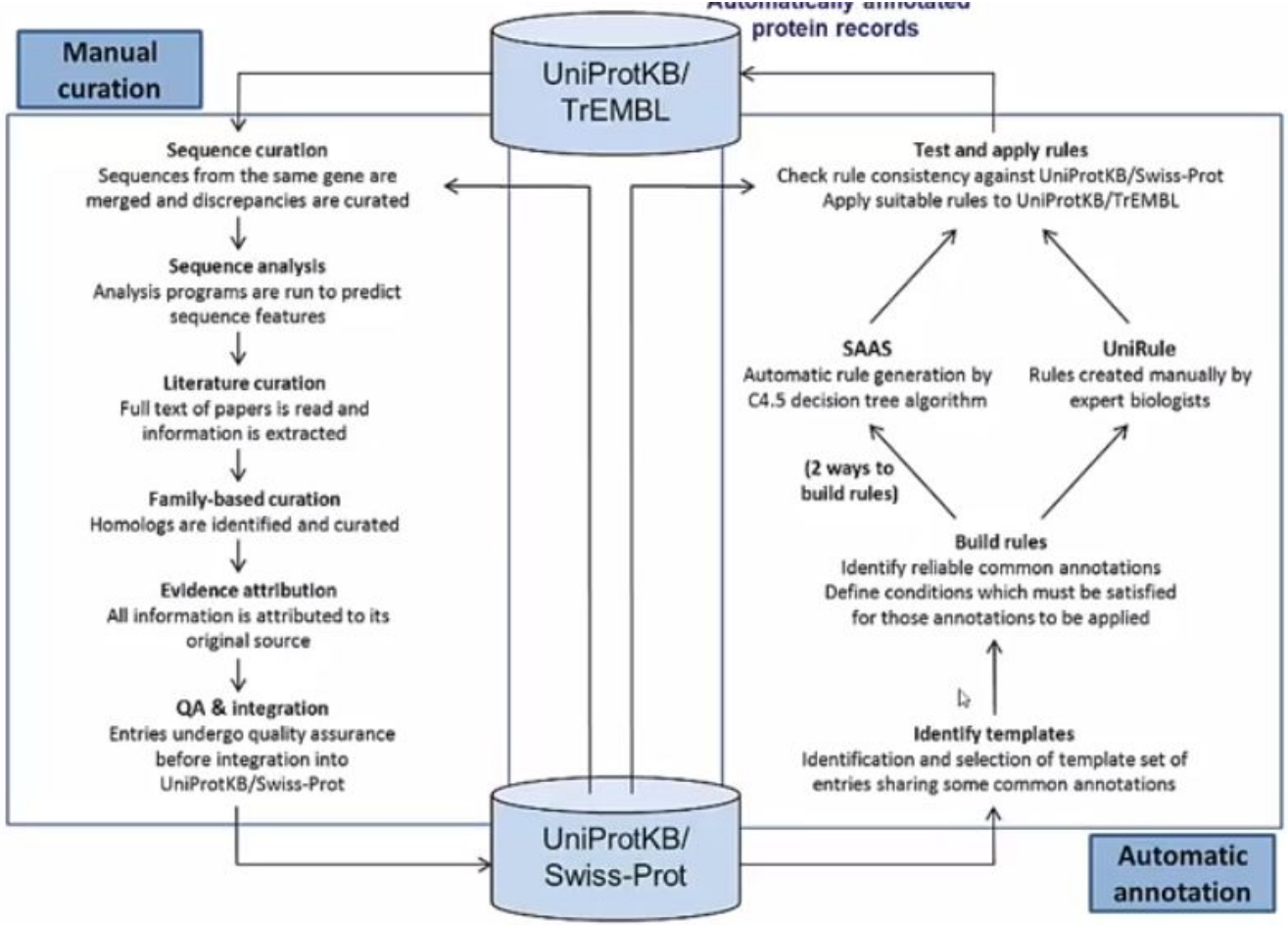
$$F(0,0) = 0$$

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j) \\ F(i-1,j) + d \\ F(i,j-1) + d \\ 0 \end{cases}$$

Quality Assessment

Given the large data volume, it's critical to provide some measures for assessing the statistical significance of a given hit.




Automatically annotated protein records
Manual curation
UniProtKB/ TrEMBL
Sequence curation
Sequences from the same gene are merged and discrepancies are curated
Sequence analysis
Analysis programs are run to predict sequence features
Literature curation
Full text of papers is read and information is extracted
Family-based curation
Homologs are identified and curated
Evidence attribution
All information is attributed to its original source
QA & integration
Entries undergo quality assurance before integration into UniProtKB/Swiss-Prot
Test and apply rules
Check rule consistency against UniProtKB/Swiss-Prot
Apply suitable rules to UniProtKB/TrEMBL
SAAS
Automatic rule generation by C4.5 decision tree algorithm
UniRule
Rules created manually by expert biologists
(2 ways to build rules)
Build rules
Identify reliable common annotations
Define conditions which must be satisfied for those annotations to be applied
Identify templates
Identification and selection of template set of entries sharing some common annotations
UniProtKB/ Swiss-Prot
Automatic annotation