CS CM122 Discussion

Apr 14 2023

Overview for today

Part 1. Textbook chapter 9

Part 2. Project 1 overview

Chapter 9

Trie of a set of patterns

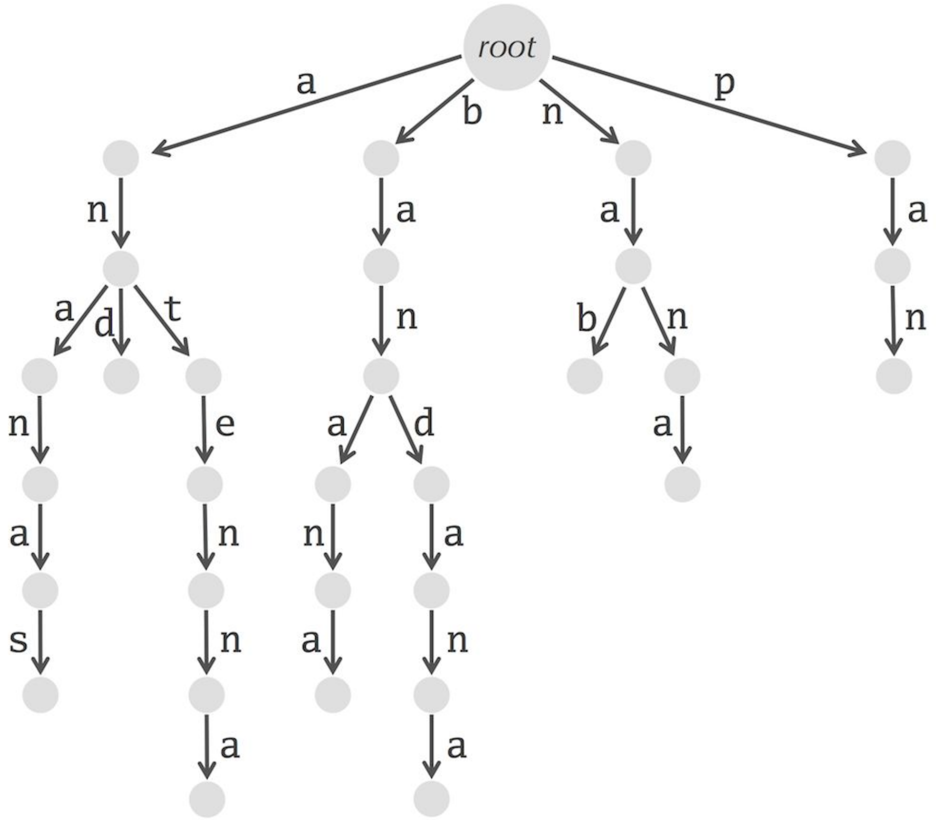
- Each edge corresponds to a character

- Each path to a leaf (end node) is a string in the set of patterns

- Each node has unique outgoing edges corresponding to different characters

- To search for a string in the trie, start from the root and traverse the edges corresponding to the characters in the string

- Question: what is the time complexity for searching a string of length K in a trie?



Trie of patterns: "ananas", "and", "antenna", "banana", "bandana", "nab", "nana", "pan" (textbook 9.3)

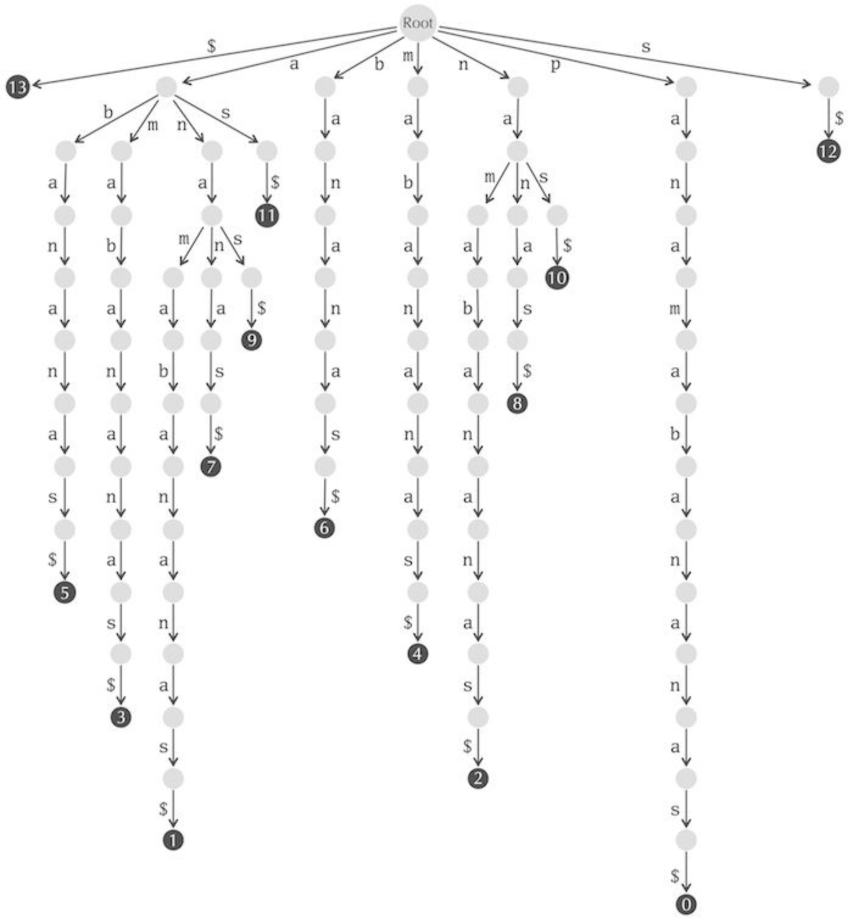
Suffix trie

The suffix trie of a string Text is the trie formed from all its suffixes.

For example, the suffix trie of “ATAC$” is the trie with set of patterns: “$”, “C$”, “AC$”, “TAC$”, “ATAC$”.

Each path to a leaf is a unique suffix

The leaves are labelled with the starting positions of the suffix



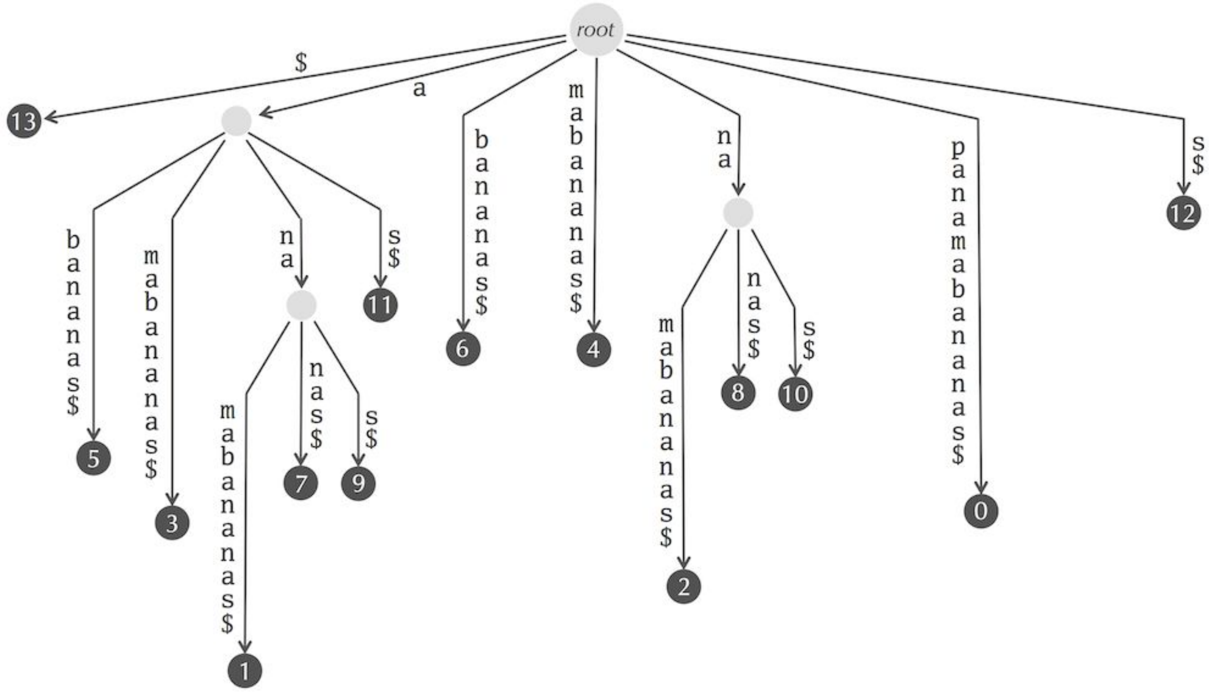
Suffix trie of text panamabananas$ (textbook 9.4)

In-class whiteboard demonstration: why the “$” sign is mandatory

- Suffix trie of string “PAPA”

- Suffix trie of string “PAPA$”

Suffix tree



Suffix tree of text panamabananas$ (textbook 9.5)

Merging single-branch paths in a suffix trie results in a suffix tree.

Each path to leaf still corresponds to a unique suffix. The number of leaves is therefore equal to the number of suffixes.

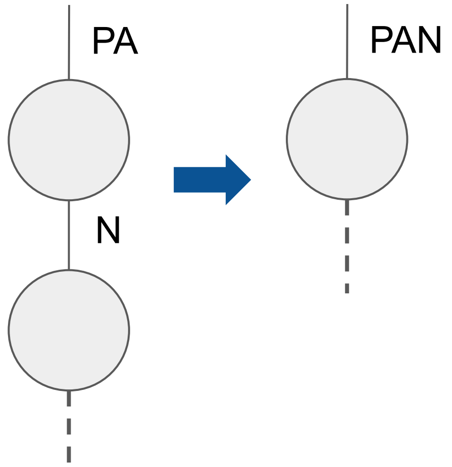
Converting a suffix trie to a suffix tree

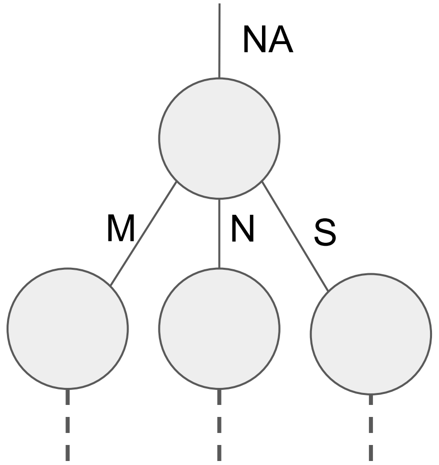
- Traverse every node in the suffix trie while keeping track of its previous edge

- If you encounter a node with only 1 child, merge the two nodes and update the previous edge, then traverse the child(ren).

- If you encounter a node with multiple children, traverse every child instead

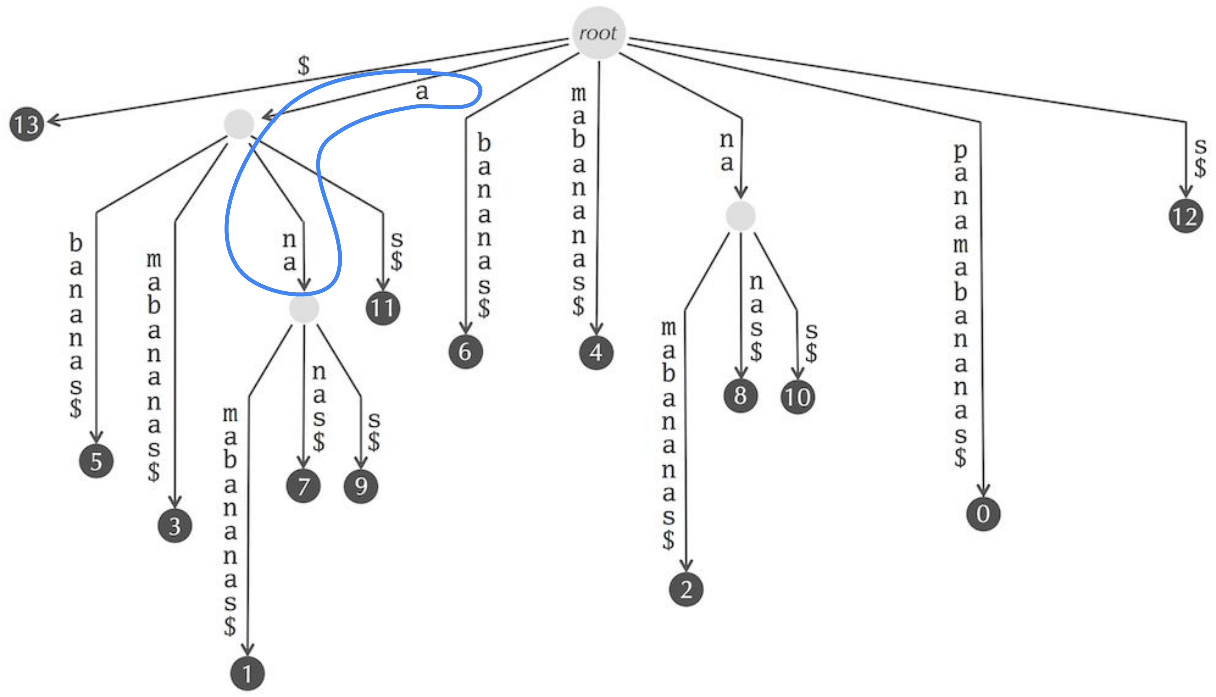
- Stop when you encounter a leaf





Longest repeat in a string

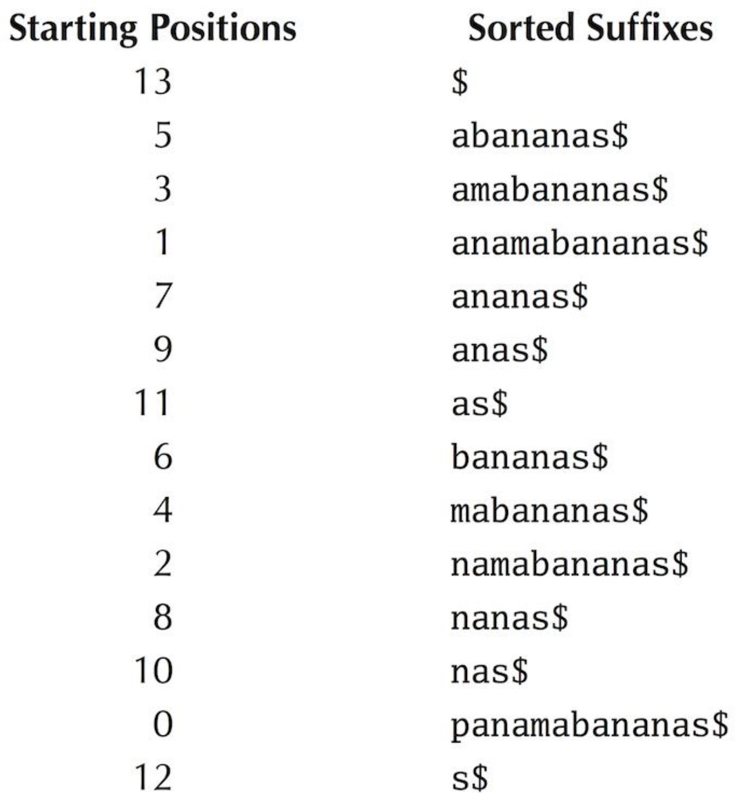
The longest repeat in a string is its longest suffix tree path that ends in a branch



Suffix array

All suffixes of a string are sorted. The resulting list of starting positions is the suffix array (no need to store the actual suffixes in memory)

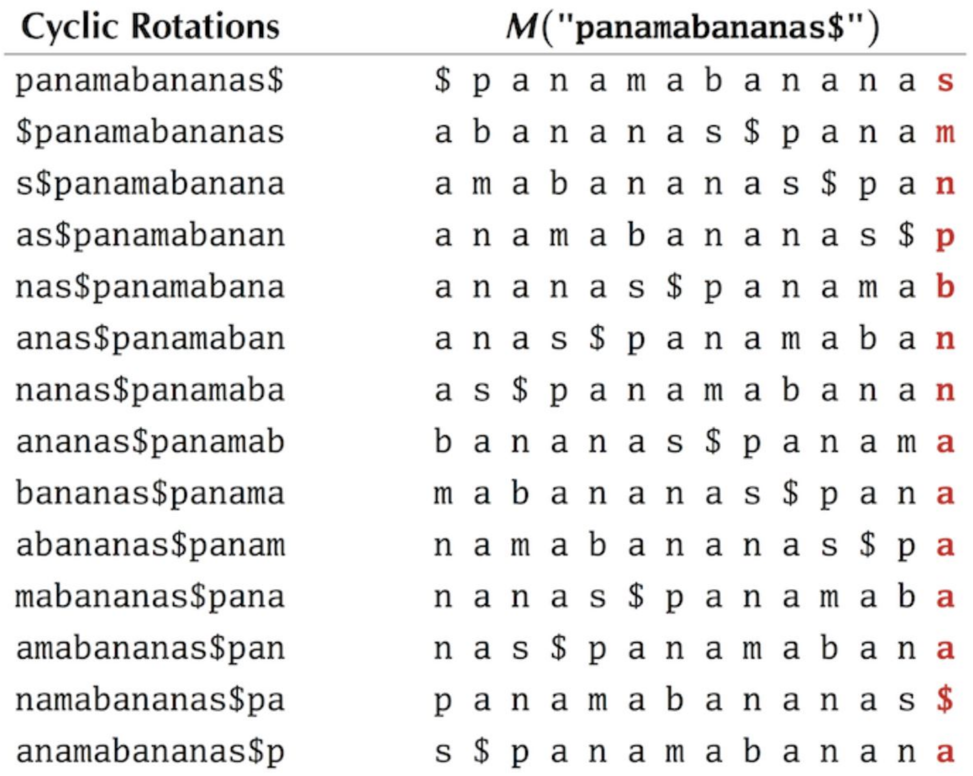
Question: given a suffix array of length N, what is an efficient algorithm to search for a substring and what is its time complexity?



Starting Positions	Sorted Suffixes
13	$
5	abananas$
3	amabananas$
1	anamabananas$
7	ananas$
9	anas$
11	as$
6	bananas$
4	mabananas$
2	namabananas$
8	nanas$
10	nas$
0	panamabananas$
12	s$

Suffix array of text panamabananas$ (textbook 9.6)

Burrows-Wheeler transform of “panamabananas$”



Cyclic Rotations	*M*("panamabananas$")
panamabananas$	$ p a n a m a b a n a n a s
$panamabananas	a b a n a n a s $ p a n a m
s$panamabanana	a m a b a n a n a s $ p a n
as$panamabanan	a n a m a b a n a n a s $ p
nas$panamabana	a n a n a s $ p a n a m a b
anas$panamaban	a n a s $ p a n a m a b a n
nanas$panamaba	a s $ p a n a m a b a n a n
ananas$panamab	b a n a n a s $ p a n a m a
bananas$panama	m a b a n a n a s $ p a n a
abananas$panam	n a m a b a n a n a s $ p a
mabananas$pana	n a n a s $ p a n a m a b a
amabananas$pan	n a s $ p a n a m a b a n a
namabananas$pa	p a n a m a b a n a n a s $
anamabananas$p	s $ p a n a m a b a n a n a

Textbook 9.7

In the original text, the first letter in a row follows the last letter in the same row.

Question: to generate the BWT do you need to store the entire matrix of cyclic rotations in the memory?

First-last property of a Burrows-Wheeler transform

$pa_3na_2ma_1ba_4na_5na_6s\$$

The ith occurrence of a letter in the last column is exactly the same instance in the original text as the the ith occurrence of the letter in the first column.

$\$$	p	a	n	a	m	a	b	a	n	a	n	a	s
a_1	b	a	n	a	n	a	s	$\$$	p	a	n	a	m
a_2	m	a	b	a	n	a	n	a	s	$\$$	p	a	n
a_3	n	a	m	a	b	a	n	a	n	a	s	$\$$	p
a_4	n	a	n	a	s	$\$$	p	a	n	a	m	a	b
a_5	n	a	s	$\$$	p	a	n	a	m	a	b	a	n
a_6	s	$\$$	p	a	n	a	m	a	b	a	n	a	n
b	a	n	a	n	a	s	$\$$	p	a	n	a	m	a_1
m	a	b	a	n	a	n	a	s	$\$$	p	a	n	a_2
n	a	m	a	b	a	n	a	n	a	s	$\$$	p	a_3
n	a	n	a	s	$\$$	p	a	n	a	m	a	b	a_4
n	a	s	$\$$	p	a	n	a	m	a	b	a	n	a_5
p	a	n	a	m	a	b	a	n	a	n	a	s	$\$$
s	$\$$	p	a	n	a	m	a	b	a	n	a	n	a_6

Textbook 9.9

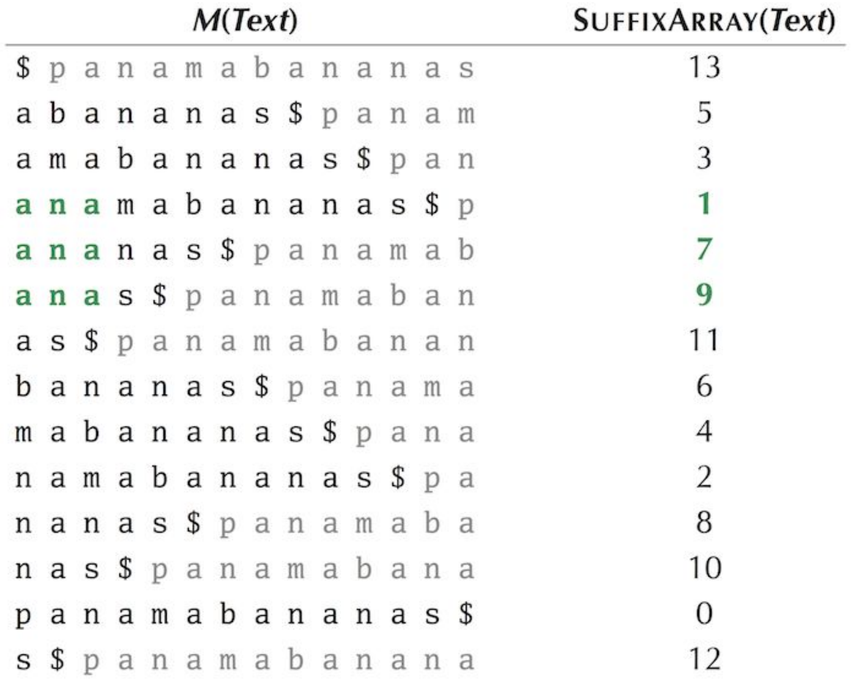
In-class whiteboard demonstrations: BWT procedures

- Inverting a transformed text

- Finding occurrences of a substring

How to find the original positions of matched substrings?

Have an array containing original positions of each suffix

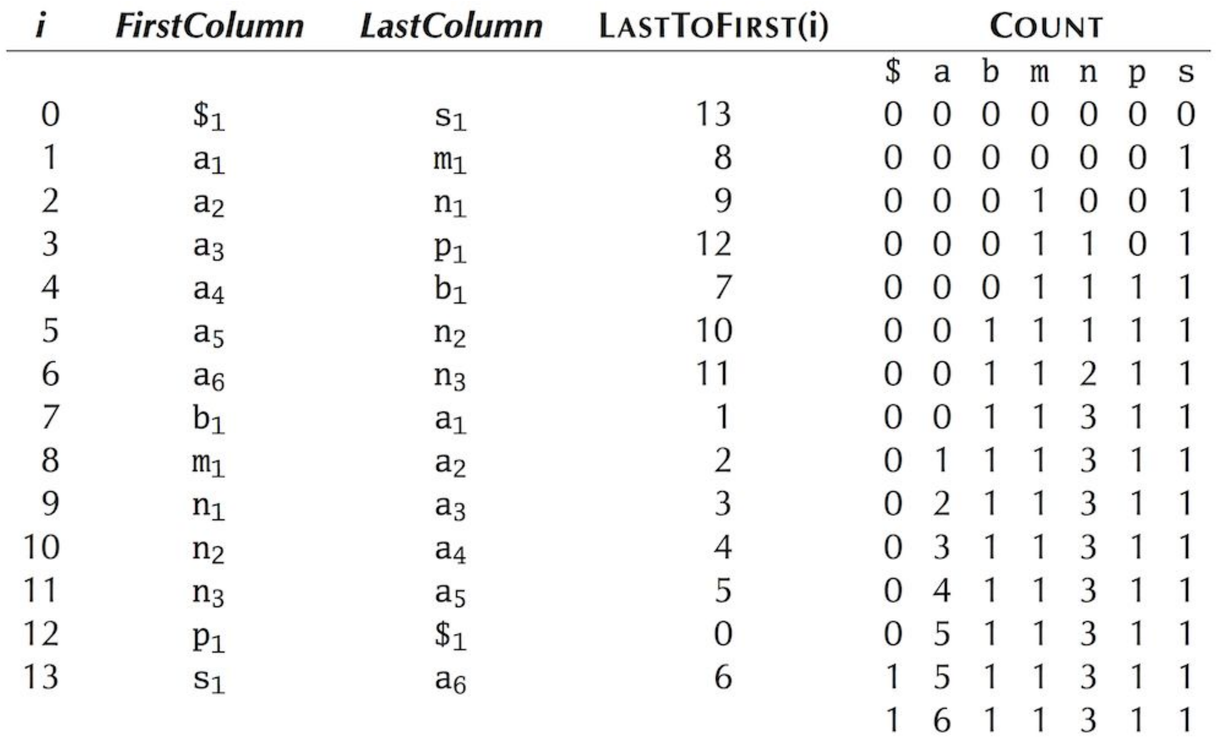


M(*Text*)	SUFFIXARRAY(*Text*)
$ p a n a m a b a n a n a s	13
a b a n a n a s $ p a n a m	5
a m a b a n a n a s $ p a n	3
a n a m a b a n a n a s $ p	**1**
a n a n a s $ p a n a m a b	**7**
a n a s $ p a n a m a b a n	**9**
a s $ p a n a m a b a n a n	11
b a n a n a s $ p a n a m a	6
m a b a n a n a s $ p a n a	4
n a m a b a n a n a s $ p a	2
n a n a s $ p a n a m a b a	8
n a s $ p a n a m a b a n a	10
p a n a m a b a n a n a s $	0
s $ p a n a m a b a n a n a	12

Textbook 9.12

How do you know how many times a letter occurs in the last column?

Have an array counting the occurrence of each letter in the last column up to each row

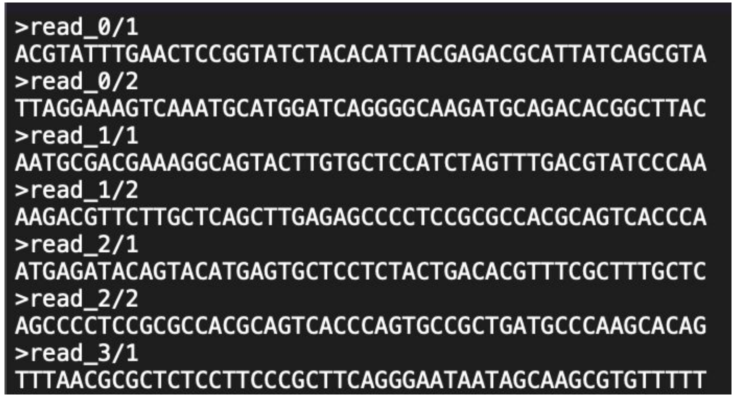


i	*FirstColumn*	*LastColumn*	LastToFirst(i)	Count						
				\$	a	b	m	n	p	s
0	$\$_1$	s_1	13	0	0	0	0	0	0	0
1	a_1	m_1	8	0	0	0	0	0	0	1
2	a_2	n_1	9	0	0	0	1	0	0	1
3	a_3	p_1	12	0	0	0	1	1	0	1
4	a_4	b_1	7	0	0	0	1	1	1	1
5	a_5	n_2	10	0	0	1	1	1	1	1
6	a_6	n_3	11	0	0	1	1	2	1	1
7	b_1	a_1	1	0	0	1	1	3	1	1
8	m_1	a_2	2	0	1	1	1	3	1	1
9	n_1	a_3	3	0	2	1	1	3	1	1
10	n_2	a_4	4	0	3	1	1	3	1	1
11	n_3	a_5	5	0	4	1	1	3	1	1
12	p_1	$\$_1$	0	0	5	1	1	3	1	1
13	s_1	a_6	6	1	5	1	1	3	1	1
				1	6	1	1	3	1	1

Textbook 9.11

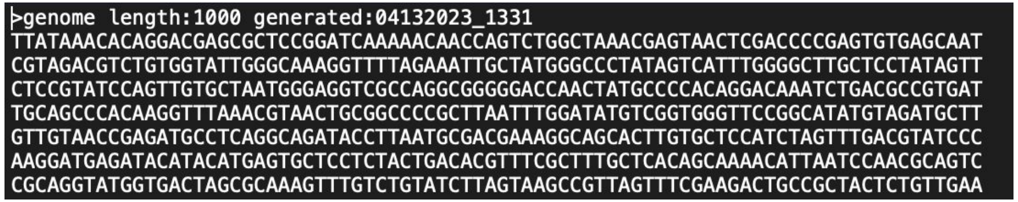
Project 1

Format of data files



```
>read_0/1
ACGTATTTGAACTCCGGTATCTACACATTACGAGACGCATTATCAGCGTA
>read_0/2
TTAGGAAAGTCAAATGCATGGATCAGGGGCAAGATGCAGACACGGCTTAC
>read_1/1
AATGCGACGAAAGGCAGTACTTGTGCTCCATCTAGTTTGACGTATCCCAA
>read_1/2
AAGACGTTCTTGCTCAGCTTGAGAGCCCCTCCGCGCCACGCAGTCACCCA
>read_2/1
ATGAGATACAGTACATGAGTGCTCCTCTACTGACACGTTTCGCTTTGCTC
>read_2/2
AGCCCCTCCGCGCCACGCAGTCACCCAGTGCCGCTGATGCCCAAGCACAG
>read_3/1
TTTAACGCGCTCTCCTTCCCGCTTCAGGGAATAATAGCAAGCGTGTTTTT
```

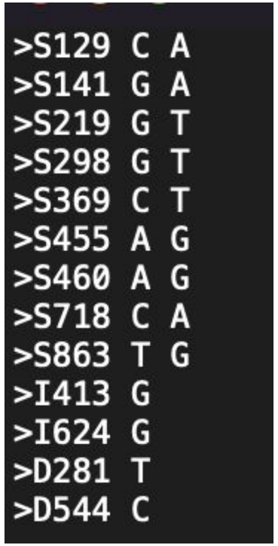
Reads file: each line is a read, each read may contain errors



```
>genome length:1000 generated:04132023_1331
TTATAAACACAGGACGAGCGCTCCGGATCAAAAACAACCAGTCTGGCTAAACGAGTAACTCGACCCCGAGTGTGAGCAAT
CGTAGACGTCTGTGGTATTGGGCAAAGGTTTTAGAAATTGCTATGGGCCCTATAGTCATTTGGGGCTTGCTCCTATAGTT
CTCCGTATCCAGTTGTGCTAATGGGAGGTCGCCAGGCGGGGGACCAACTATGCCCCACAGGACAAATCTGACGCCGTGAT
TGCAGCCCACAAGGTTTAAACGTAACTGCGGCCCCGCTTAATTTGGATATGTCGGTGGGTTCCGGCATATGTAGATGCTT
GTTGTAACCGAGATGCCTCAGGCAGATACCTTAATGCGACGAAAGGCAGCACTTGTGCTCCATCTAGTTTGACGTATCCC
AAGGATGAGATACATACATGAGTGCTCCTCTACTGACACGTTTCGCTTTGCTCACAGCAAAACATTAATCCAACGCAGTC
CGCAGGTATGGTGACTAGCGCAAAGTTTGTCTGTATCTTAGTAAGCCGTTAGTTTCGAAGACTGCCGCTACTCTGTTGAA
```

Genome file: a genome split into several lines

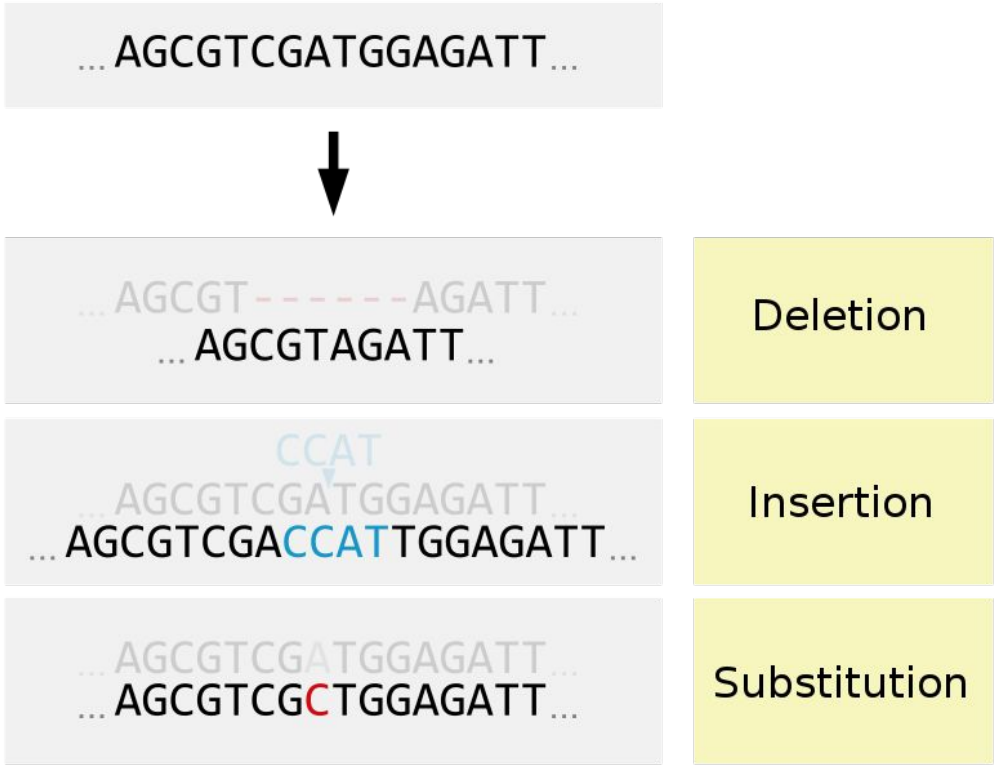
Your output should look like this. S for substitutions, I for insertions and D for deletions



```
>S129 C A
>S141 G A
>S219 G T
>S298 G T
>S369 C T
>S455 A G
>S460 A G
>S718 C A
>S863 T G
>I413 G
>I624 G
>D281 T
>D544 C
```

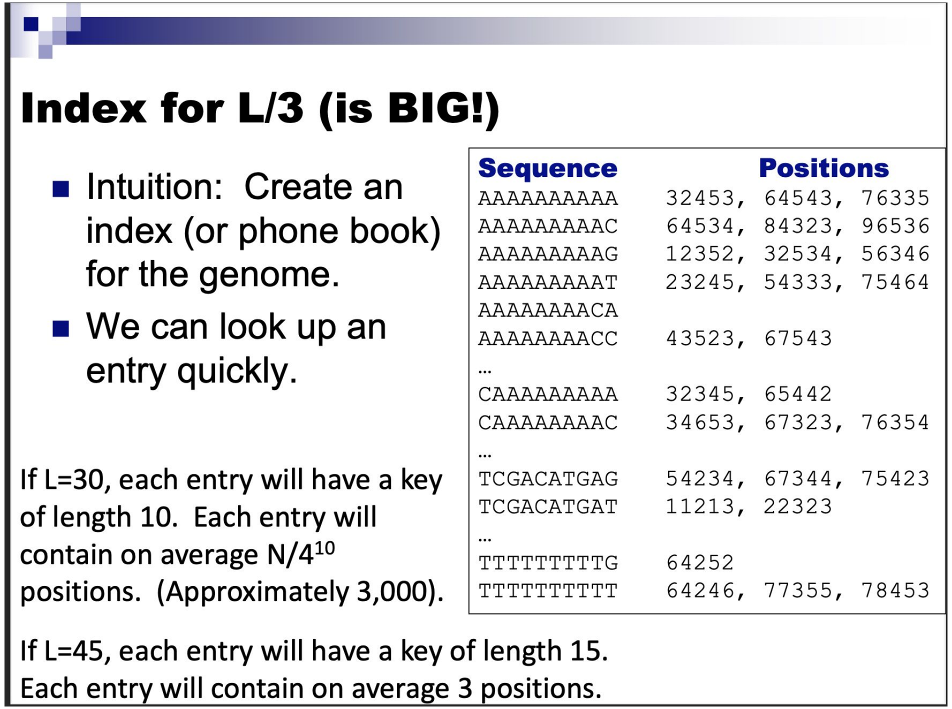
Types of genetic mutations used for grading

https://commons.wikimedia.org/wiki/File:Deletion_Insertion_Substitution-en.svg



What data structure should you use to represent the genome?

Lecture 3



Index for L/3 (is BIG!)

- Intuition: Create an index (or phone book) for the genome.
- We can look up an entry quickly.

Sequence	Positions
AAAAAAAAAA	32453, 64543, 76335
AAAAAAAAAC	64534, 84323, 96536
AAAAAAAAAG	12352, 32534, 56346
AAAAAAAAAT	23245, 54333, 75464
AAAAAAAACA	
AAAAAAAACC	43523, 67543
…	
CAAAAAAAAA	32345, 65442
CAAAAAAAAC	34653, 67323, 76354
…	
TCGACATGAG	54234, 67344, 75423
TCGACATGAT	11213, 22323
…	
TTTTTTTTTG	64252
TTTTTTTTTT	64246, 77355, 78453

If L=30, each entry will have a key of length 10. Each entry will contain on average N/4^{10} positions. (Approximately 3,000).

If L=45, each entry will have a key of length 15. Each entry will contain on average 3 positions.

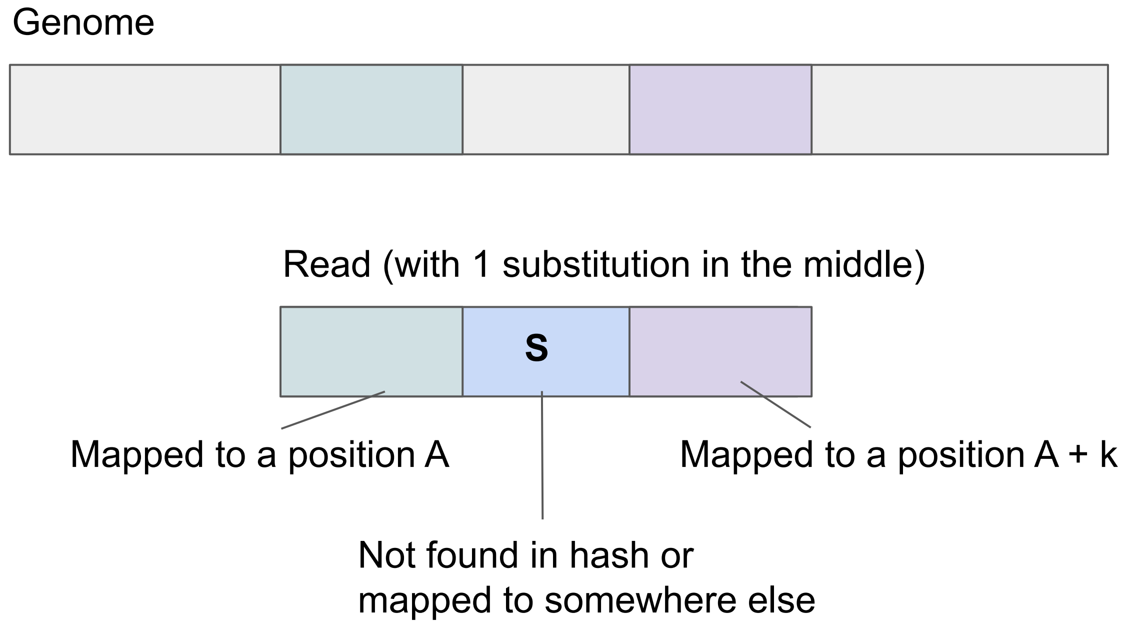
You can create a hash table containing positions of all unique k-mers in the genome.

You can also use tries or BWT.

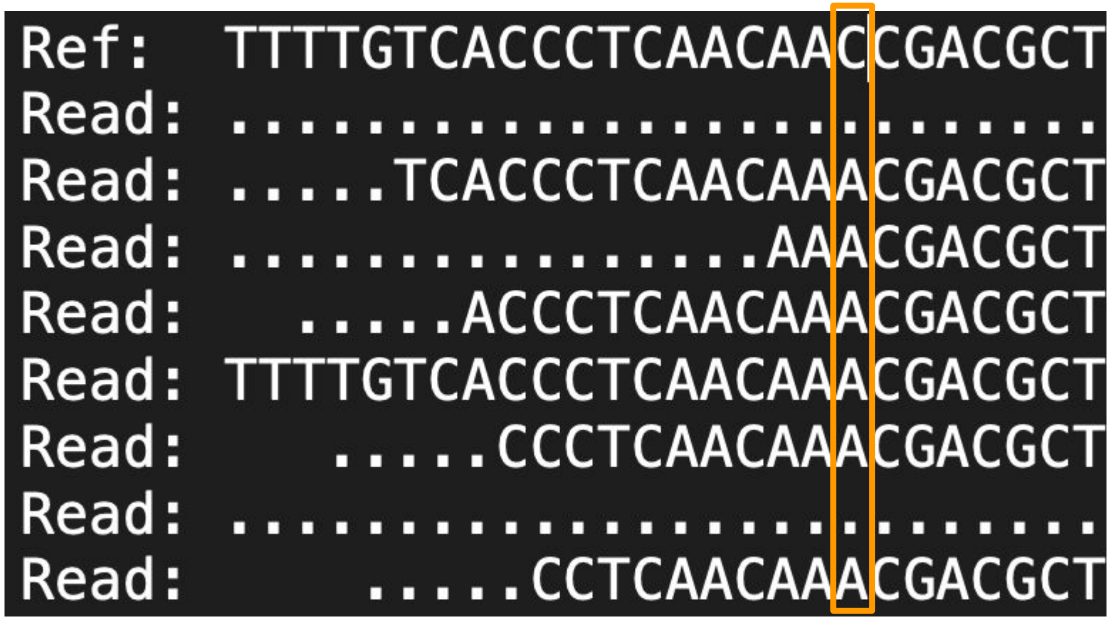
How do you map a read to the genome?

Assume you have a read of length L and a hash table k-mer positions.

You can cut the read into L/k fragments and try to find the position of each fragment. x


Genome
Read (with 1 substitution in the middle)
S
Mapped to a position A
Mapped to a position A + k
Not found in hash or mapped to somewhere else

How do you identify mutations from aligned reads?



```
Ref:  TTTTGTCACCCTCAACAACCGACGCT
Read: ..........................
Read: .....TCACCCTCAACAAACGACGCT
Read: ...............AAACGACGCT
Read:   .....ACCCTCAACAAACGACGCT
Read: TTTTGTCACCCTCAACAAACGACGCT
Read:    .....CCCTCAACAAACGACGCT
Read: ..........................
Read:     .....CCTCAACAAACGACGCT
```

After aligning reads to the reference genome, your program should be able to compare the reference and the reads at each position and check if a different nucleotide is present in the reads.

Advanced read mapping techniques

Even better fragment-based alignment

Published: 04 March 2012

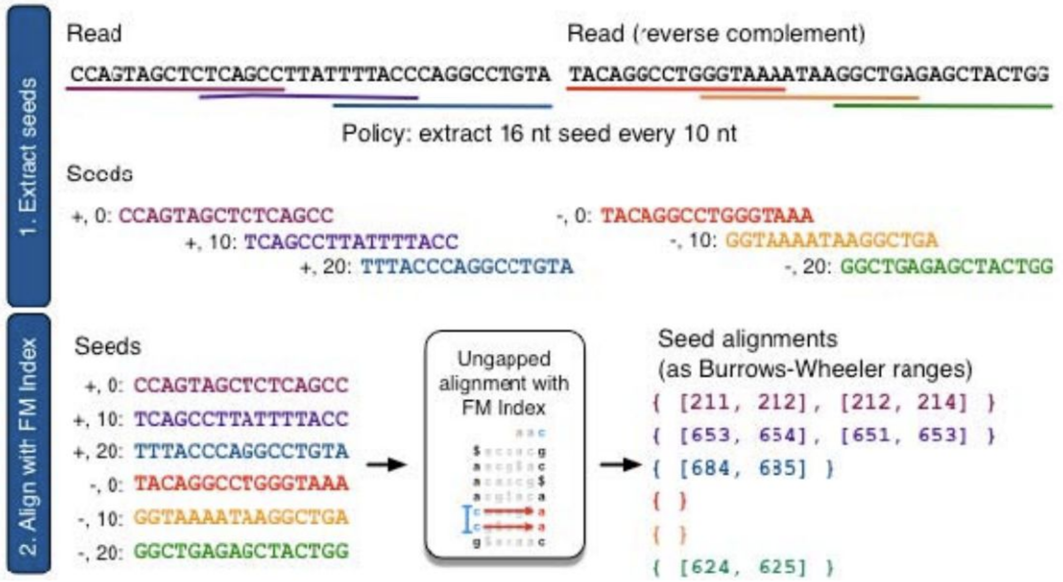
Fast gapped-read alignment with Bowtie 2

Ben Langmead & Steven L Salzberg

Nature Methods **9**, 357–359 (2012) | Cite this article

82k Accesses | **27158** Citations | **129** Altmetric | Metrics

Instead of cutting a read of length L into L/k fragments, apply a sliding window of length k across the read (this allows you to get more than L/k fragments)



What if your fragments are mapping to many different positions?

JOURNAL ARTICLE

The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote

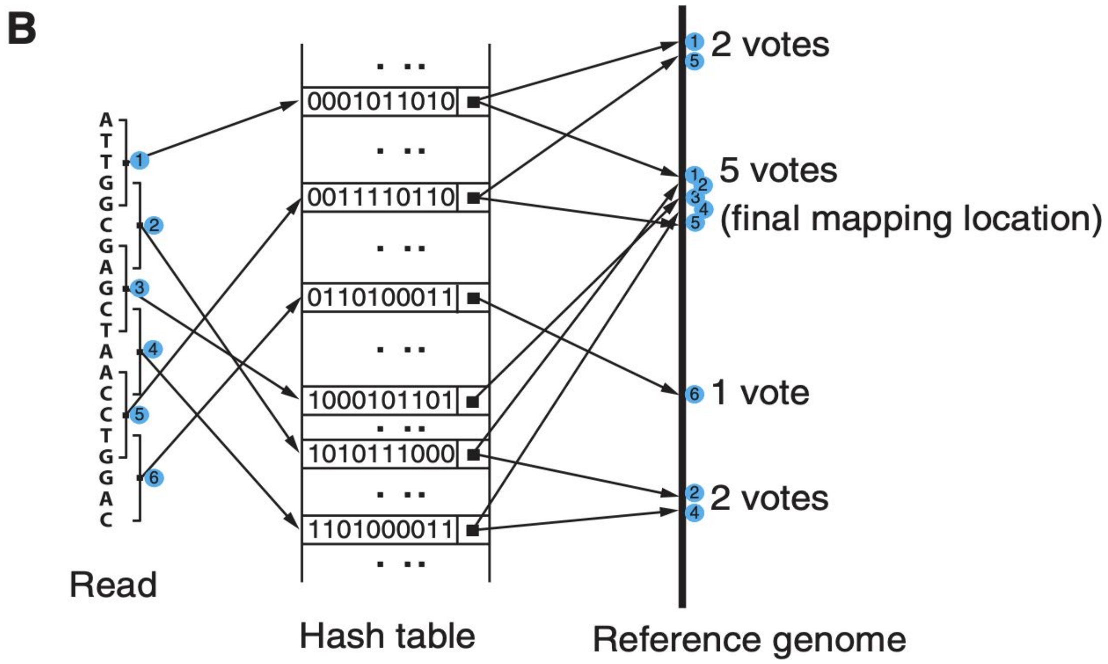
Yang Liao, Gordon K. Smyth, Wei Shi

Nucleic Acids Research, Volume 41, Issue 10, 1 May 2013, Page e108, https://doi.org/10.1093/nar/gkt214

Published: 03 April 2013 **Article history** ▾

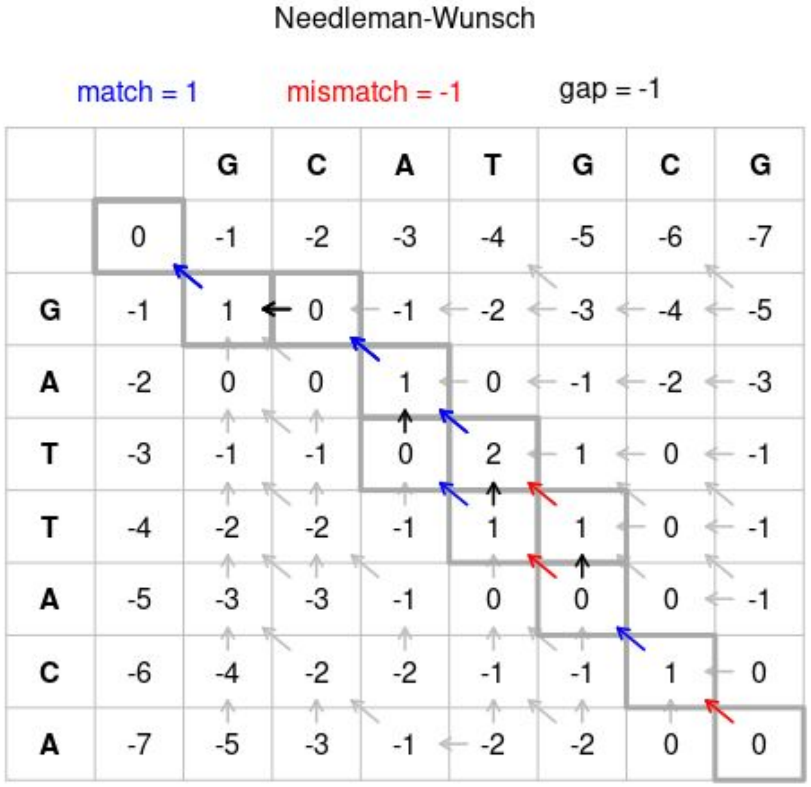
Each fragment in a read casts a "vote" on each of the potential genomic regions where it came from.

The region with the majority of the votes from all fragments wins.



Needleman-Wunsch alignment for two sequences (in Chapter 5 of textbook)

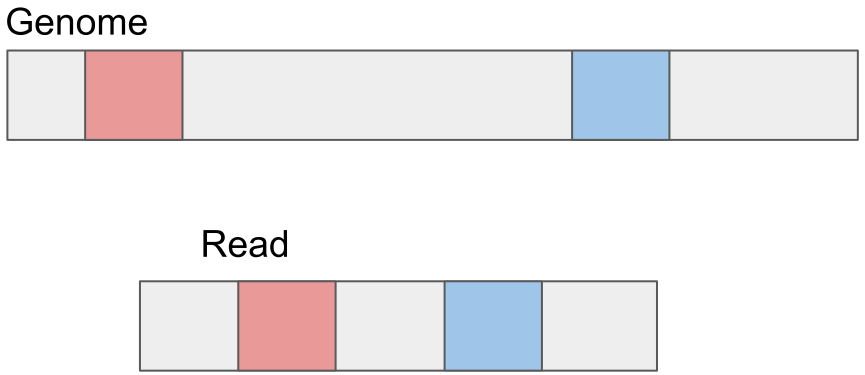
This allows you to find any arbitrary indels.



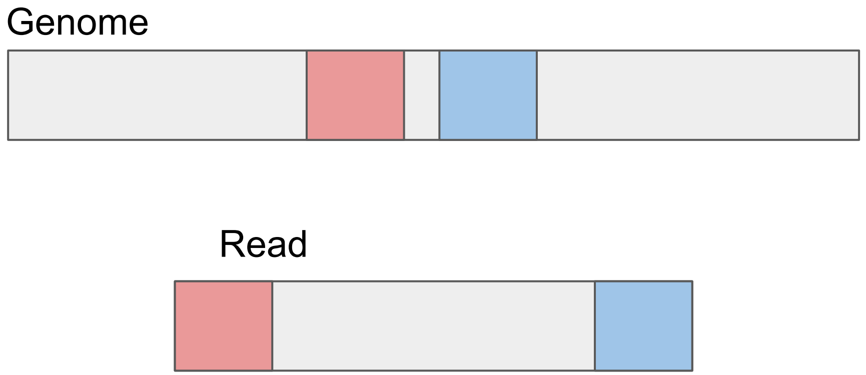
		G	C	A	T	G	C	G
	0	-1	-2	-3	-4	-5	-6	-7
G	-1	1	0	-1	-2	-3	-4	-5
A	-2	0	0	1	0	-1	-2	-3
T	-3	-1	-1	0	2	1	0	-1
T	-4	-2	-2	-1	1	1	0	-1
A	-5	-3	-3	-1	0	0	0	-1
C	-6	-4	-2	-2	-1	-1	1	0
A	-7	-5	-3	-1	-2	-2	0	0

Wikipedia

How would indels look like on your read?



Deletion: some fragments of your read will map to positions that are more far apart on the genome



Insertion: some fragments of your read will map to positions that are closer on the genome