

# Dynamic Programming

Michael Schatz

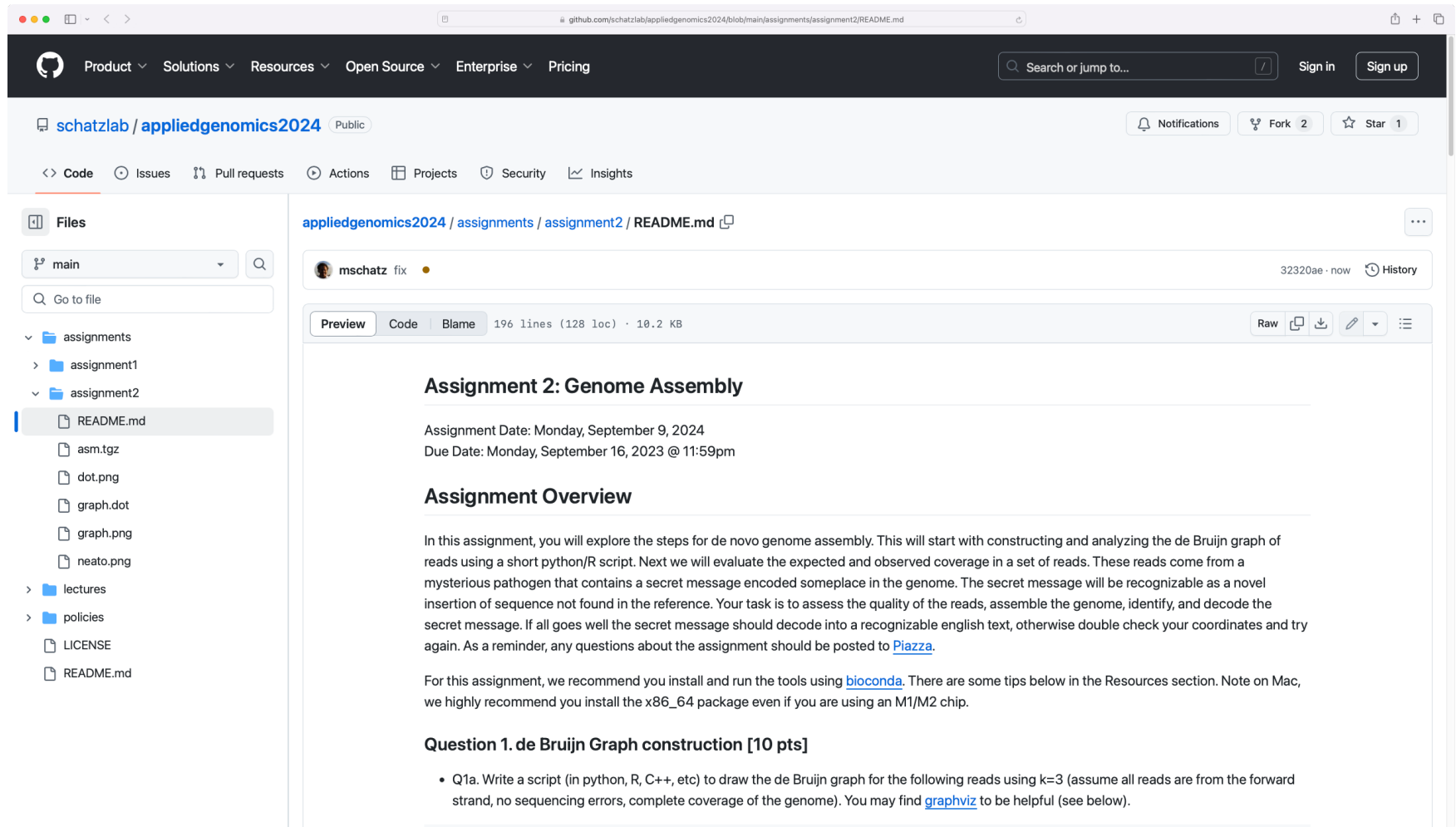
Sept 18, 2024

Lecture 7: Applied Comparative Genomics



# Assignment 2: Genome Assembly

## Due Monday Sept 16 by 11:59pm



The screenshot shows a GitHub repository page for 'schatzlab / appliedgenomics2024'. The repository is public and has 2 forks and 1 star. The file tree on the left shows the following structure:

- assignments
  - assignment1
  - assignment2
    - README.md
    - asm.tgz
    - dot.png
    - graph.dot
    - graph.png
    - neato.png
- lectures
- policies
- LICENSE
- README.md

The main content area displays the 'assignment2/README.md' file. The file is titled 'Assignment 2: Genome Assembly' and was last modified by 'mschatz' on September 9, 2024. The file contains the following text:

Assignment Date: Monday, September 9, 2024  
Due Date: Monday, September 16, 2023 @ 11:59pm

### Assignment Overview

In this assignment, you will explore the steps for de novo genome assembly. This will start with constructing and analyzing the de Bruijn graph of reads using a short python/R script. Next we will evaluate the expected and observed coverage in a set of reads. These reads come from a mysterious pathogen that contains a secret message encoded someplace in the genome. The secret message will be recognizable as a novel insertion of sequence not found in the reference. Your task is to assess the quality of the reads, assemble the genome, identify, and decode the secret message. If all goes well the secret message should decode into a recognizable english text, otherwise double check your coordinates and try again. As a reminder, any questions about the assignment should be posted to [Piazza](#).

For this assignment, we recommend you install and run the tools using [bioconda](#). There are some tips below in the Resources section. Note on Mac, we highly recommend you install the x86\_64 package even if you are using an M1/M2 chip.

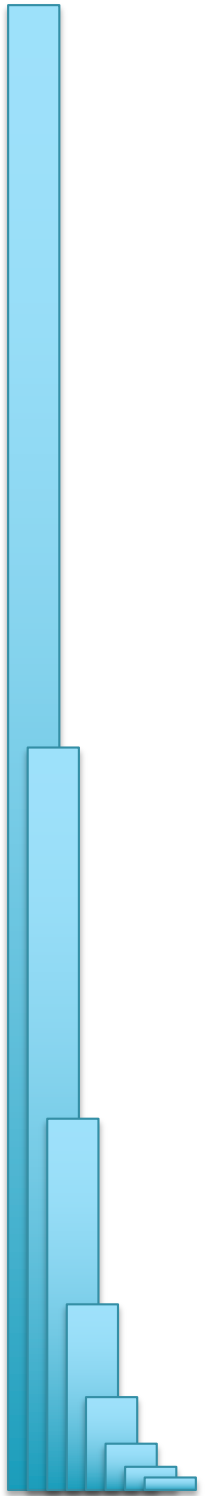
### Question 1. de Bruijn Graph construction [10 pts]

- Q1a. Write a script (in python, R, C++, etc) to draw the de Bruijn graph for the following reads using  $k=3$  (assume all reads are from the forward strand, no sequencing errors, complete coverage of the genome). You may find [graphviz](#) to be helpful (see below).

<https://github.com/schatzlab/appliedgenomics2024/tree/main/assignments/assignment2>

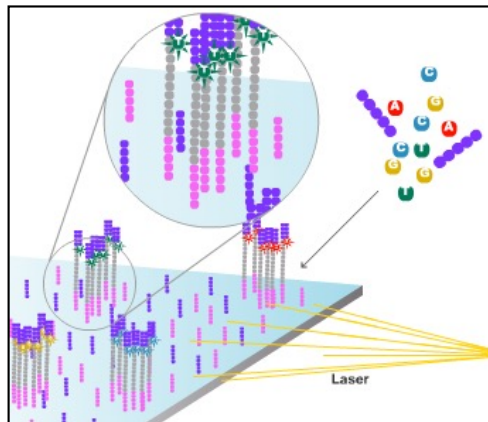
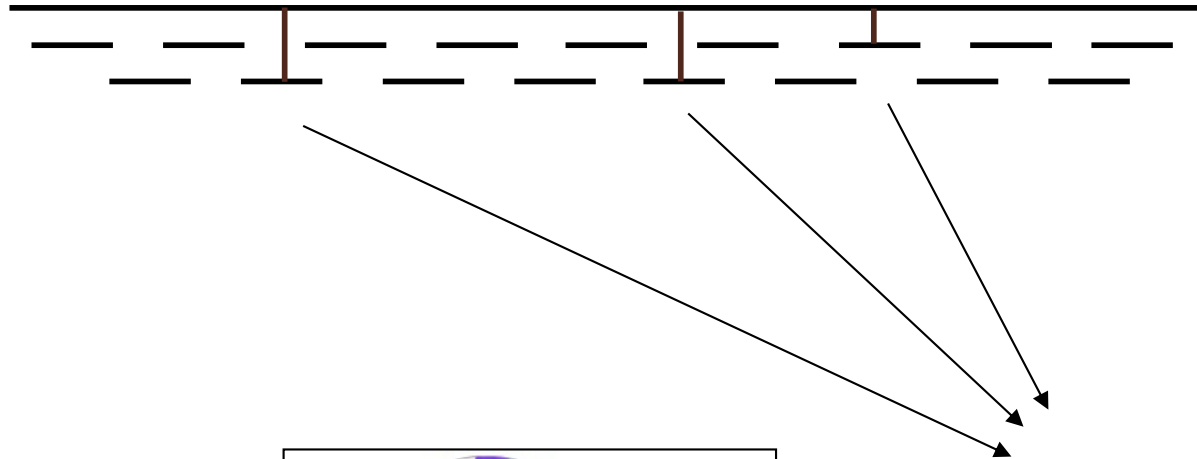
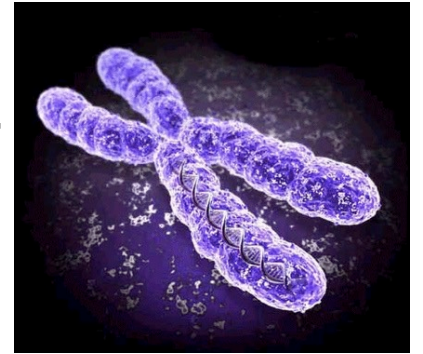
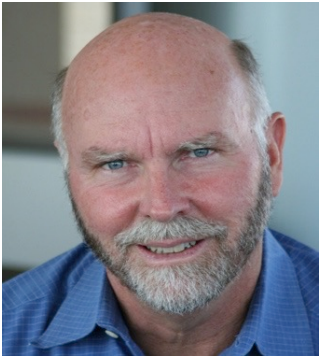
Check Piazza for questions!

# Read Mapping



# Personal Genomics

How does your genome compare to the reference?



Heart Disease

Cancer

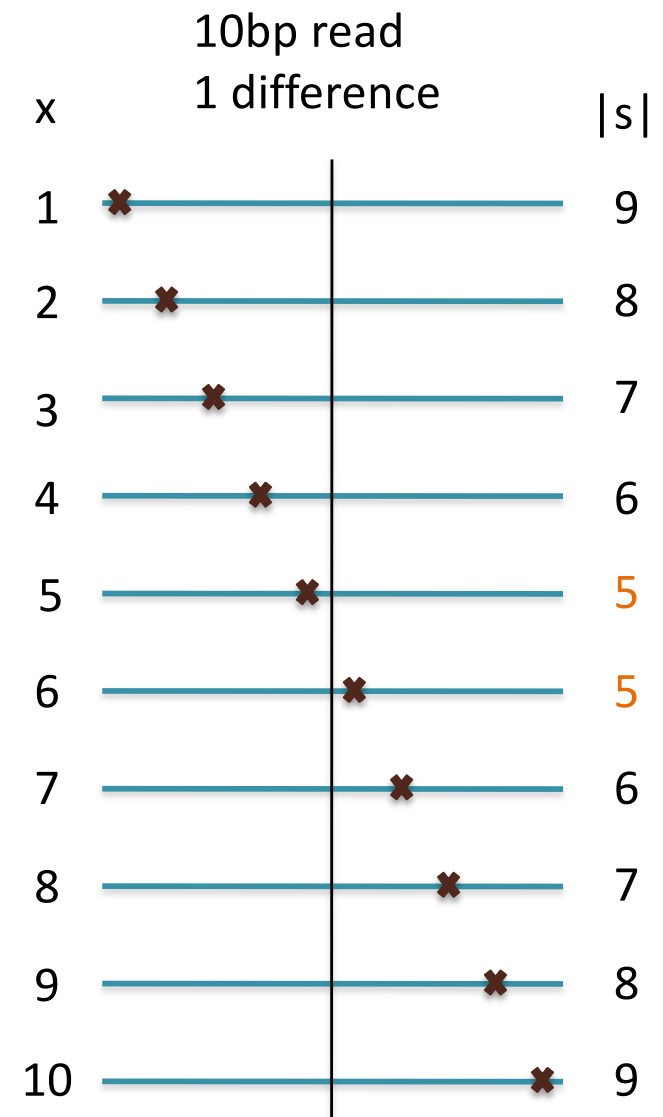
Presidential smile



# Seed-and-Extend Alignment

Theorem: An alignment of a sequence of length  $m$  with at most  $k$  differences **must** contain an exact match at least  $s = m / (k + 1)$  bp long  
(Baeza-Yates and Perleberg, 1996)

- Proof: Pigeonhole principle
  - 1 pigeon can't fill 2 holes
- Seed-and-extend search
  - Use an index to rapidly find short exact alignments to seed longer in-exact alignments
    - BLAST, MUMmer, Bowtie, BWA, SOAP, ...
  - Specificity of the depends on seed length
    - Guaranteed sensitivity for  $k$  differences
    - Also finds some (but not all) lower quality alignments <- heuristic



# Brute Force Analysis



- Brute Force:
  - At every possible offset in the genome:
    - Do all of the characters of the query match?
- Analysis
  - Simple, easy to understand
  - Genome length =  $n$  [3B]
  - Query length =  $m$  [7]
  - Comparisons:  $(n-m+1) * m$  [21B]
- Overall runtime:  $O(nm)$ 
  - [How long would it take if we double the genome size, read length?]
  - [How long would it take if we double both?]

# Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower
- Searching for GATTACA
  - $Lo = 1; Hi = 15; Mid = (1+15)/2 = 8$
  - $Middle = Suffix[8] = CC$   
=> Higher:  $Lo = Mid + 1$
  - $Lo = 9; Hi = 15; Mid = (9+15)/2 = 12$
  - $Middle = Suffix[12] = TACC$   
=> Lower:  $Hi = Mid - 1$
  - $Lo = 9; Hi = 11; Mid = (9+11)/2 = 10$
  - $Middle = Suffix[10] = GATTACC$   
=> Lower:  $Hi = Mid - 1$
  - $Lo = 9; Hi = 9; Mid = (9+9)/2 = 9$
  - $Middle = Suffix[9] = GATTACA...$   
=> Match at position 2!

#	Sequence	Pos
1	ACAGATTACC...	6
2	ACC...	13
3	AGATTACC...	8
4	ATTACAGATTACC...	3
5	ATTACC...	10
6	C...	15
7	CAGATTACC...	7
8	CC...	14
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Lo  
Hi  
→

# Binary Search Analysis

- Binary Search

Initialize search range to entire list

$mid = (hi+lo)/2$ ;  $middle = suffix[mid]$

if query matches middle: done

else if query < middle: pick low range

else if query > middle: pick hi range

Repeat until done or empty range

[WHEN?]

- Analysis

- More complicated method

- How many times do we repeat?

- How many times can it cut the range in half?

- Find smallest  $x$  such that:  $n/(2^x) \leq 1$ ;  $x = \lg_2(n)$

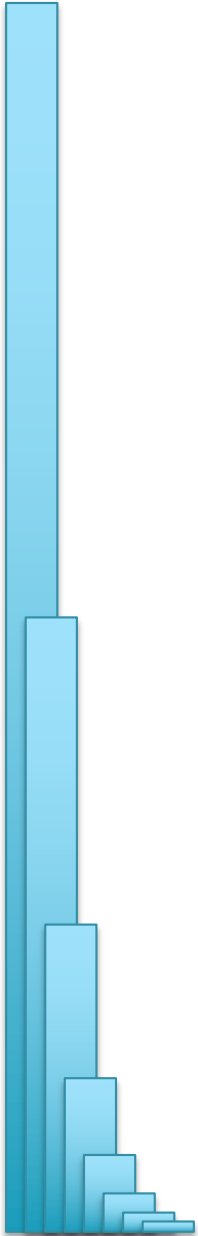
[32]

- Total Runtime:  $O(m \lg n)$

- More complicated, but **much** faster!

- Looking up a query loops 32 times instead of 3B

[How long does it take to search 6B or 24B nucleotides?]

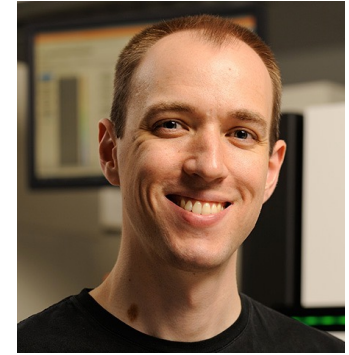




# Algorithmic challenge

How can we combine the speed of a suffix array  $O(m + \lg(n))$  (or even  $O(m)$ ) with the size of a brute force analysis ( $n$  bytes)?

What would such an index look like?



# Bowtie: Ultrafast and memory efficient alignment of short DNA sequences to the human genome

Slides Courtesy of Ben Langmead

# Algorithm Overview

## 1. Split read into segments

Read

**CCAGTAGCTCTCAGCCTTATTTACCCAGGCCTGTA**

Read (reverse complement)

**TACAGGCCTGGGTAAATAAGGCTGAGAGCTACTGG**

Policy: extract 16 nt seed every 10 nt

## Seeds

+, 0: CCAGTAGCTCTCAGCC

+, 10: **TCAGCCTTATTTACC**

+, 20: **TTACCCAGGCCTGTA**

-, 0: **TACAGGCCTGGGTAAA**

-, 10: GGTAAAATAAGGCTGA

-, 20: GGCTGAGAGCTACTGG

## 2. Lookup each segment and prioritize

## Seeds

+, 0: CCAGTAGCTCTCAGCC

+, 10: **TCAGCCTTATTTACC**

+, 20: **TTACCCAGGCCTGTA**

-, 0: **TACAGGCCTGGGTAAA**

-, 10: GGTAAAATAAGGCTGA

-, 20: GGCTGAGAGCTACTGG

Ungapped  
alignment with  
FM Index

Diagram illustrating a sequence alignment between a reference sequence and a query sequence. The reference sequence is "a a c" and the query sequence is "\$ a c a a c g". The alignment shows the query sequence shifted to the right, with red arrows indicating the alignment of "c" to "c" and "a" to "a".

### Seed alignments (as B ranges)

$$\{ [211, 212], [212, 214] \}$$

{ [653, 654], [651, 653] }

**{ [684, 685] }**

{ }

{ }

{ [624, 625] }

### 3. Evaluate end-to-end match

## Extension candidates

SA:684, chr12:1955

SA:624, chr2:462

SA:211: chr4:762

SA:213: chr12:1935

SA:652: chr12:1945

SIMD dynamic  
programming  
aligner

## SAM alignments

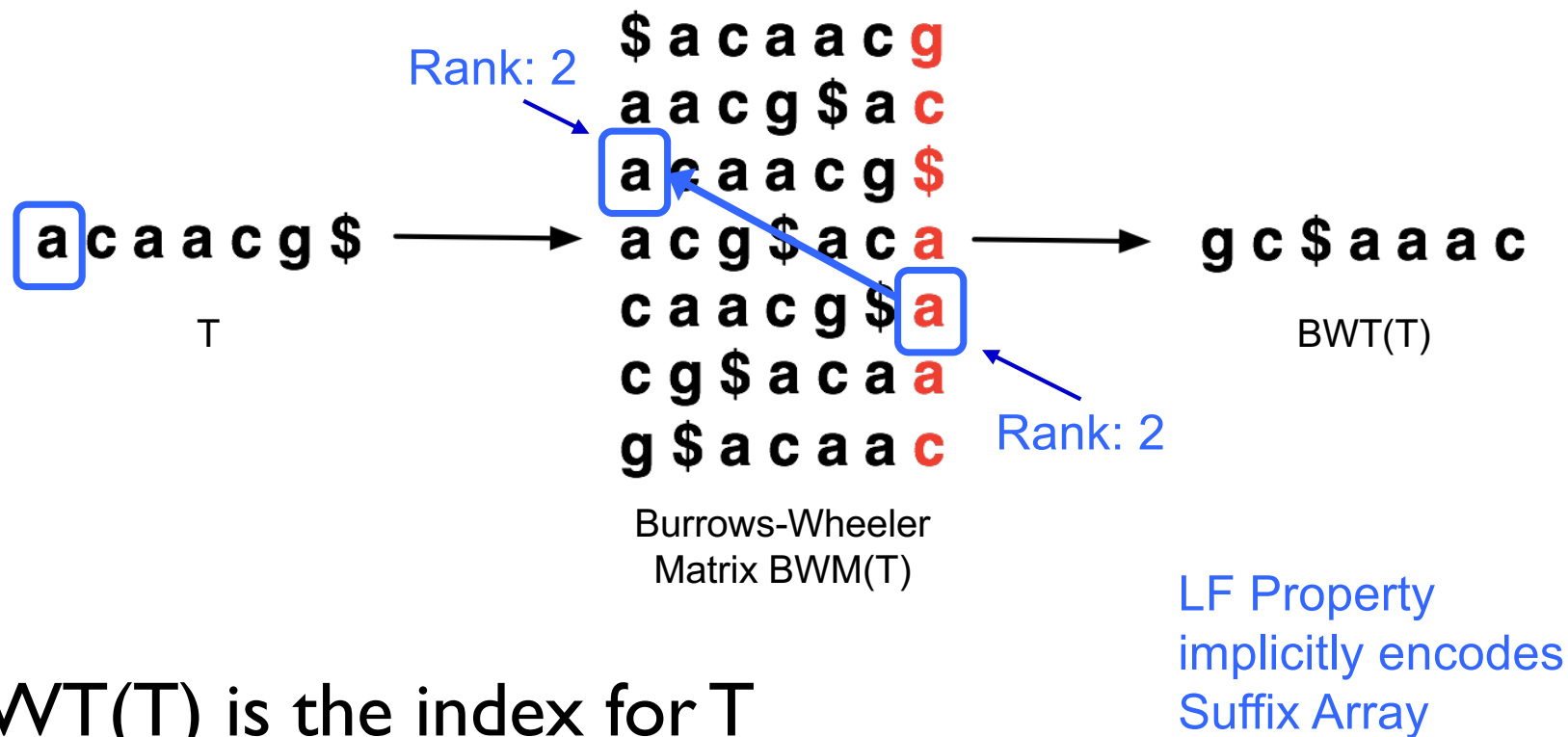
```
r1      0    chr12     1936       0  
        36M *      0      0  
→ CCAGTAGCTCTCAGCCTTATTTTACCCAGGCCTGTA  
   IAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AS:i:0   XS:i:-2   XN:i:0  
XM:i:0   XO:i:0    XG:i:0  
NM:i:0   MD:Z:36   YT:Z:UU  
YM:i:0
```

...

(Langmead & Salzberg, 2012)

# Burrows-Wheeler Transform

- Reversible permutation of the characters in a text



- BWT( $T$ ) is the index for  $T$

**A block sorting lossless data compression algorithm.**

Burrows M, Wheeler DJ (1994) *Digital Equipment Corporation*. Technical Report 124

06.BWTNotes.pdf Page 1

**BWT Algorithm Overview**

Input: A string  $S$  of length  $n$ .  
Output: A string  $B$  of length  $n$ .

**Algorithm:**

- Sort the characters of  $S$  lexicographically to form  $S_{\text{sorted}}$ .
- Append a special character (e.g., '\$') to the end of  $S$  and sort the result to form  $S_{\text{sorted\_with\_end\_char}}$ .
- For each character  $c$  in  $S$ , find its position in  $S_{\text{sorted\_with\_end\_char}}$  and record the index of the character immediately preceding it in the sorted order.
- Form the BWT string  $B$  by concatenating the characters at the recorded indices.

06.BWTNotes.pdf Page 2

**BWT Algorithm Overview**

Input: A string  $B$  of length  $n$ .  
Output: A string  $S$  of length  $n$ .

**Algorithm:**

- Form the LF array by recording the index of the character immediately preceding each character in  $B$ .
- Form the first column  $F$  by sorting the characters of  $B$ .
- Form the last column  $L$  by sorting the characters of  $B$ .
- Form the rank array  $R$  by recording the rank of each character in  $B$ .
- Form the inverse rank array  $IR$  by recording the inverse rank of each character in  $B$ .
- Form the original string  $S$  by following the LF links from the first column.

06.BWTNotes.pdf Page 3

**BWT Algorithm Overview**

Input: A string  $B$  of length  $n$ .  
Output: A string  $S$  of length  $n$ .

**Algorithm:**

- Form the LF array by recording the index of the character immediately preceding each character in  $B$ .
- Form the first column  $F$  by sorting the characters of  $B$ .
- Form the last column  $L$  by sorting the characters of  $B$ .
- Form the rank array  $R$  by recording the rank of each character in  $B$ .
- Form the inverse rank array  $IR$  by recording the inverse rank of each character in  $B$ .
- Form the original string  $S$  by following the LF links from the first column.

06.BWTNotes.pdf Page 4

**BWT Algorithm Overview**

Input: A string  $B$  of length  $n$ .  
Output: A string  $S$  of length  $n$ .

**Algorithm:**

- Form the LF array by recording the index of the character immediately preceding each character in  $B$ .
- Form the first column  $F$  by sorting the characters of  $B$ .
- Form the last column  $L$  by sorting the characters of  $B$ .
- Form the rank array  $R$  by recording the rank of each character in  $B$ .
- Form the inverse rank array  $IR$  by recording the inverse rank of each character in  $B$ .
- Form the original string  $S$  by following the LF links from the first column.

06.BWTNotes.pdf Page 5

**BWT Algorithm Overview**

Input: A string  $B$  of length  $n$ .  
Output: A string  $S$  of length  $n$ .

**Algorithm:**

- Form the LF array by recording the index of the character immediately preceding each character in  $B$ .
- Form the first column  $F$  by sorting the characters of  $B$ .
- Form the last column  $L$  by sorting the characters of  $B$ .
- Form the rank array  $R$  by recording the rank of each character in  $B$ .
- Form the inverse rank array  $IR$  by recording the inverse rank of each character in  $B$ .
- Form the original string  $S$  by following the LF links from the first column.

06.BWTNotes.pdf Page 6

**BWT Algorithm Overview**

Input: A string  $B$  of length  $n$ .  
Output: A string  $S$  of length  $n$ .

**Algorithm:**

- Form the LF array by recording the index of the character immediately preceding each character in  $B$ .
- Form the first column  $F$  by sorting the characters of  $B$ .
- Form the last column  $L$  by sorting the characters of  $B$ .
- Form the rank array  $R$  by recording the rank of each character in  $B$ .
- Form the inverse rank array  $IR$  by recording the inverse rank of each character in  $B$ .
- Form the original string  $S$  by following the LF links from the first column.

06.BWTNotes.pdf Page 7

**BWT Algorithm Overview**

Input: A string  $B$  of length  $n$ .  
Output: A string  $S$  of length  $n$ .

**Algorithm:**

- Form the LF array by recording the index of the character immediately preceding each character in  $B$ .
- Form the first column  $F$  by sorting the characters of  $B$ .
- Form the last column  $L$  by sorting the characters of  $B$ .
- Form the rank array  $R$  by recording the rank of each character in  $B$ .
- Form the inverse rank array  $IR$  by recording the inverse rank of each character in  $B$ .
- Form the original string  $S$  by following the LF links from the first column.

06.BWTNotes.pdf Page 8

**BWT Algorithm Overview**

Input: A string  $B$  of length  $n$ .  
Output: A string  $S$  of length  $n$ .

**Algorithm:**

- Form the LF array by recording the index of the character immediately preceding each character in  $B$ .
- Form the first column  $F$  by sorting the characters of  $B$ .
- Form the last column  $L$  by sorting the characters of  $B$ .
- Form the rank array  $R$  by recording the rank of each character in  $B$ .
- Form the inverse rank array  $IR$  by recording the inverse rank of each character in  $B$ .
- Form the original string  $S$  by following the LF links from the first column.

06.BWTNotes.pdf Page 9

**BWT Algorithm Overview**

Input: A string  $B$  of length  $n$ .  
Output: A string  $S$  of length  $n$ .

**Algorithm:**

- Form the LF array by recording the index of the character immediately preceding each character in  $B$ .
- Form the first column  $F$  by sorting the characters of  $B$ .
- Form the last column  $L$  by sorting the characters of  $B$ .
- Form the rank array  $R$  by recording the rank of each character in  $B$ .
- Form the inverse rank array  $IR$  by recording the inverse rank of each character in  $B$ .
- Form the original string  $S$  by following the LF links from the first column.

Recompute the original text when the BWT is ACTTGA\$TTAA

# Burrows-Wheeler Transform

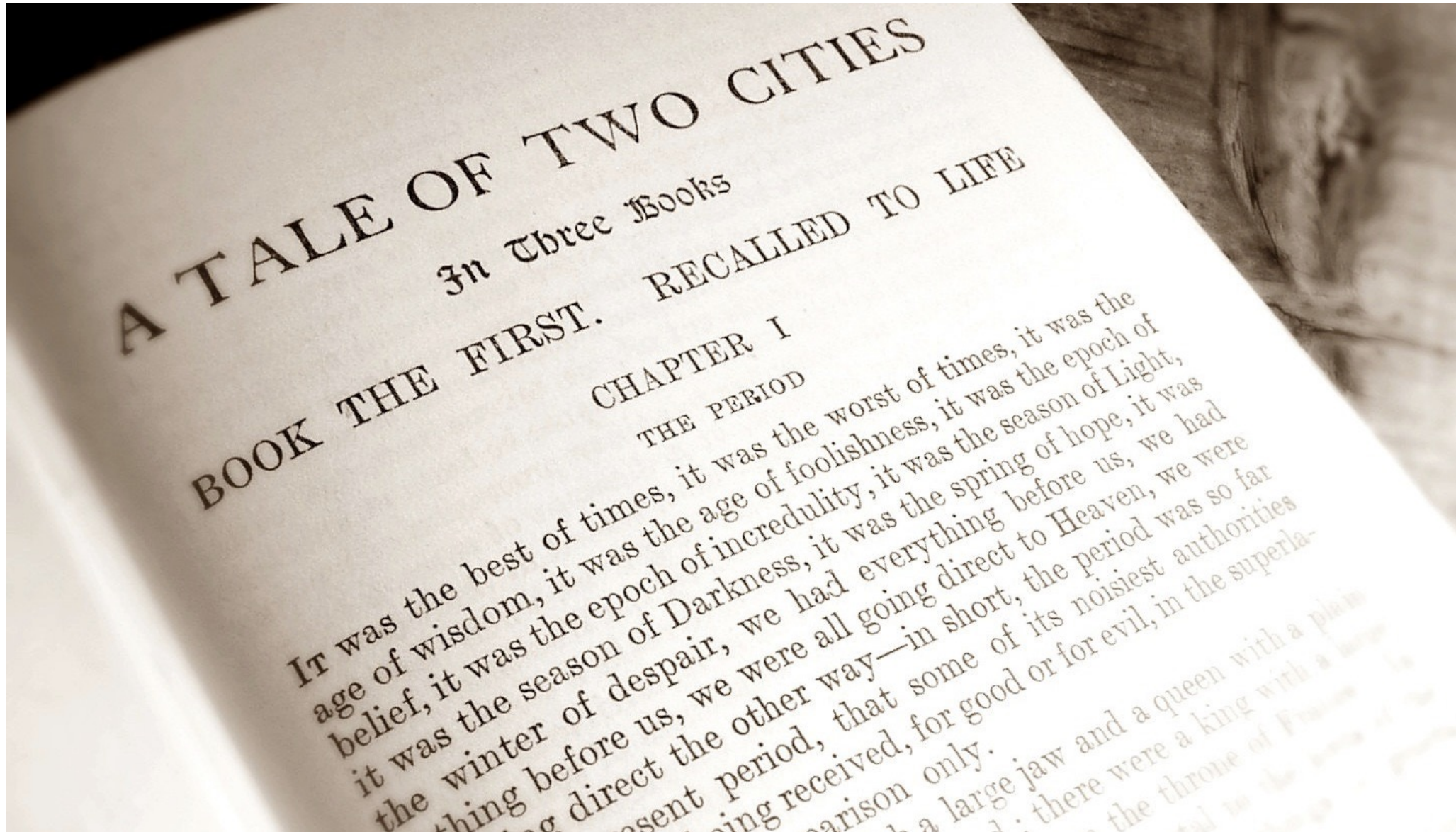
- Recreating T from BWT(T)
  - Start in the first row and apply **LF** repeatedly, accumulating predecessors along the way



[Decode this BWT string: ACTGA\$TTA ]



# Run Length Encoding



# Run Length Encoding

**ref[614] :**

It\_was\_the\_best\_of\_times,\_it\_was\_the\_worst\_of\_times,\_it\_was\_the\_age\_of\_wisdom,\_it\_was\_the\_age\_of\_foolishness,\_it\_was\_the\_epoch\_of\_belief,\_it\_was\_the\_epoch\_of\_incredulity,\_it\_was\_the\_season\_of\_Light,\_it\_wa\_s\_the\_season\_of\_Darkness,\_it\_was\_the\_spring\_of\_hope,\_it\_was\_the\_wint\_er\_of\_despair,\_we\_had\_everything\_before\_us,\_we\_had\_nothing\_before\_us,\_we\_were\_all\_going\_direct\_to\_Heaven,\_we\_were\_all\_going\_direct\_the\_o ther\_way\_-\_in\_short,\_the\_period\_was\_so\_far\_like\_the\_present\_period,\_that\_some\_of\_its\_noisiest\_authorities\_insisted\_on\_its\_being\_received,\_for\_good\_or\_for\_evil,\_in\_the\_superlative\_degree\_of\_comparison\_only.\$

## ***Run Length Encoding:***

- Replace a “run” of a character X with a single X followed by the length of the run
- GAAAAAAAAATTACA => GA8T2ACA (reverse is also easy to implement)
- If your text contains numbers, then you will need to use a (slightly) more sophisticated encoding



# Run Length Encoding

**ref[614]:**

It\_was\_the\_best\_of\_times,\_it\_was\_the\_worst\_of\_times,\_it\_was\_the\_age\_of\_wisdom,\_it\_was\_the\_age\_of\_foolishness,\_it\_was\_the\_epoch\_of\_belief,\_it\_was\_the\_epoch\_of\_incredulity,\_it\_was\_the\_season\_of\_Light,\_it\_wa\_s\_the\_season\_of\_Darkness,\_it\_was\_the\_spring\_of\_hope,\_it\_was\_the\_wint\_er\_of\_despair,\_we\_had\_everything\_before\_us,\_we\_had\_nothing\_before\_us,\_we\_were\_all\_going\_direct\_to\_Heaven,\_we\_were\_all\_going\_direct\_the\_o ther\_way\_-\_in\_short,\_the\_period\_was\_so\_far\_like\_the\_present\_period,\_that\_some\_of\_its\_noisiest\_authorities\_insisted\_on\_its\_being\_received,\_for\_good\_or\_for\_evil,\_in\_the\_superlative\_degree\_of\_comparison\_only.\$

**rle(ref) [614]:**

It\_was\_the\_best\_of\_times,\_it\_was\_the\_worst\_of\_times,\_it\_was\_the\_age\_of\_wisdom,\_it\_was\_the\_age\_of\_fo2lishnes2,\_it\_was\_the\_epoch\_of\_belief,\_it\_was\_the\_epoch\_of\_incredulity,\_it\_was\_the\_season\_of\_Light,\_it\_wa\_s\_the\_season\_of\_Darknes2,\_it\_was\_the\_spring\_of\_hope,\_it\_was\_the\_wint\_er\_of\_despair,\_we\_had\_everything\_before\_us,\_we\_had\_nothing\_before\_us,\_we\_were\_al2\_going\_direct\_to\_Heaven,\_we\_were\_al2\_going\_direct\_the\_o ther\_way\_-\_in\_short,\_the\_period\_was\_so\_far\_like\_the\_present\_period,\_that\_some\_of\_its\_noisiest\_authorities\_insisted\_on\_its\_being\_received,\_for\_go2d\_or\_for\_evil,\_in\_the\_superlative\_degre2\_of\_comparison\_only.\$

# Run Length Encoding

**ref[614] :**

It\_was\_the\_best\_of\_times,\_it\_was\_the\_worst\_of\_times,\_it\_was\_the\_age\_of\_wisdom,\_it\_was\_the\_age\_of\_foolishness,\_it\_was\_the\_epoch\_of\_belief,\_it\_was\_the\_epoch\_of\_incredulity,\_it\_was\_the\_season\_of\_Light,\_it\_wa\_s\_the\_season\_of\_Darkness,\_it\_was\_the\_spring\_of\_hope,\_it\_was\_the\_wint\_er\_of\_despair,\_we\_had\_everything\_before\_us,\_we\_had\_nothing\_before\_us,\_we\_were\_all\_going\_direct\_to\_Heaven,\_we\_were\_all\_going\_direct\_the\_o ther\_way\_-\_in\_short,\_the\_period\_was\_so\_far\_like\_the\_present\_period,\_that\_some\_of\_its\_noisiest\_authorities\_insisted\_on\_its\_being\_received,\_for\_good\_or\_for\_evil,\_in\_the\_superlative\_degree\_of\_comparison\_only.\$

**bwt[614] :**

.dlmssfty sesdtrsns\_y\_\_\$yfofeeeeetggsfefefggedrofr,llreef-,fs,,,,,,,,, ,nfrsdnnherghettedndeteegenstee,sssst,esssnssffteedttttttttttr,, ,eeefehh\_\_p\_\_fpDwwwwwwwwweehl\_ew\_\_\_\_\_eoo\_neeeoaaeoo\_\_\_\_sephhrrhvh hwwegmghhhhhhhkrrwvhhssHrrrvtrribbdbcbs\_\_thwpppvmmirdnnib\_\_eoooooo oooooo\_\_\_\_eenennnnnaai\_\_ecc\_\_ttttttttttttttttttts\_tsgltsLlvtt\_\_hhoor e\_wrraddwlors\_\_\_\_\_r\_\_lteirillre\_ouaanooiioeooooiihkiiiiiiio\_\_iei tsppioi\_\_\_\_\_ggnodsc\_sss\_gfhf\_fffhwh\_nsmo\_\_uee\_sioooaeeeeoo\_ii cgppeeaoaeooeesseuutetaaaaaaaaaaaaaai\_\_ei\_in\_\_aaie\_eeerei\_hrsssnacciIi iiiiiisn\_\_\_\_\_oyoui\_\_a\_iids\_\_aiaae\_\_\_\_\_tlar

# Run Length Encoding

**ref[614] :**

It\_was\_the\_best\_of\_times,\_it\_was\_the\_worst\_of\_times,\_it\_was\_the\_age\_of\_wisdom,\_it\_was\_the\_age\_of\_foolishness,\_it\_was\_the\_epoch\_of\_belief,\_it\_was\_the\_epoch\_of\_incredulity,\_it\_was\_the\_season\_of\_Light,\_it\_wa\_s\_the\_season\_of\_Darkness,\_it\_was\_the\_spring\_of\_hope,\_it\_was\_the\_wint\_er\_of\_despair,\_we\_had\_everything\_before\_us,\_we\_had\_nothing\_before\_us,\_we\_were\_all\_going\_direct\_to\_Heaven,\_we\_were\_all\_going\_direct\_the\_o ther\_way\_-\_in\_short,\_the\_period\_was\_so\_far\_like\_the\_present\_period,\_that\_some\_of\_its\_noisiest\_authorities\_insisted\_on\_its\_being\_received,\_for\_good\_or\_for\_evil,\_in\_the\_superlative\_degree\_of\_comparison\_only.\$

**bwt[614] :**

.dlmssfty sesdtrsns\_y\_\_\$yfofeeeeetggsfefefggeedrofr,llreef-,fs,,,,,,,,, ,nfrsdnnherghettedndeteegenstee,sssst,esssnssffteedtttttttttttr,, ,eeefehh\_\_p\_\_fpDwwwwwwwwweehl\_ew\_\_\_\_\_eoo\_neeeoaaeoo\_\_\_\_\_sephhrrhvh hwwegmghhhhhhhkrrwvhhssHrrrvtrribbdbcbs\_\_thwpppvmmirdnnib\_\_eoooooo oooooo\_\_\_\_\_eennnnnnnai\_\_\_\_\_ecc\_\_\_\_\_ttttttttttttttttts\_tsgltsLlvtt\_\_\_\_hhoor e\_wrraddwlors\_\_\_\_\_r\_\_lteirillre\_ouaanooiioeooooiihkiiiiiiio\_\_iei tsppioi\_\_\_\_\_ggnodsc\_sss\_gfhf\_fffhwh\_nsmo\_\_uee\_sioooaeeeeoo\_ii cgppeeaoaeooeesseuutetaaaaaaaaaaaaaai\_\_ei\_in\_\_aaie\_eereei\_hrsssnacciIi iiiiiisn\_\_\_\_\_tlar

Why does the BWT tend to make runs in english text?

# Run Length Encoding

**bwt[614] :**

```
.dlmssftysestrns_y__$yfofeeeeetggsfefefggedrofr,llreef-,fs,,,,,,,,,  
,,nfrsdnnherghettedndeteegenstee,sssst,esssnssffteedtttttttttttr,,  
,,eeefehh__p__fpDwwwwwwwwweehl_ew_____eoo_neeeoaaeoo_____sephhrrhvh  
hwwegmghhhhhhhkrrwwhhssHrrrvtrribbdbcbvs__thwwpppvmmirdnnib__eoooooo  
oooooo_____eennnnnnnai__ecc__ttttttttttttttttttts_tsgltsLlvtt__hhoor  
e_wrraddwlors_____r__lteirillre_ouaanooiioeooooiihkiiiiio__iei  
tsppioi_____ggnodsc_sss_gfhf_fffhwh_nsmo__uee_sioooaeeeeoo__ii  
cgppeeaoaeooesseuutetaaaaaaaaaaaaaai__ei_in__aaie_eeerei_hrsssnacciIi  
iiiiisn_____oyoui__a_iids__aiaee_____tlar
```

**rle(bwt) [464] :**

```
.dlms2ftysestrns_y_2$yfofe4tg2sfefefg2e2drofr,l2re2f-,fs,9nfrsdn2  
herghet2edndete2ge2nste2,s5t,es3ns2f2te2dt10r,4e3feh2_2p_2fpDw11e2h  
l_ew_5eo2_ne3oa2eo2_4seph2r2hvh2w2egmgh7kr2w2h2s2Hr3vtr2ib2dbcbvs_2t  
hw2p3vm2irdn2ib_2eo12_4e2n6a2i_3ec2_2t18s_tsgltsLlvtt2_3h2o2re_wr2ad2  
wlors_9r_2lteiril2re_oua2no2i2oeo4i3hki6o_2ieitsp2ioi_12g2nodsc_s3_g  
fhf_f3hwh_nsmo_2ue2_sio3ae4o2_i2cgp2e2aoaeo2e2s2eu2teta11i_2ei_in_2a  
2ie_e3rei_hrs3nac2i2Ii7sn_15oyoui_2a_i3ds_2ai2ae2_21tlar
```

# Run Length Encoding

**bwt[614] :**

```
.dlmssftysestrns_y__$yfofeeeeetggsfefefggedrofr,llreef-,fs,,,,,,,,,
,,nfrsdnnherghettedndeteegenstee,sssst,esssnssfsteedttrtr,,
,,eeefehh__p__fpDwwwwwwweehl_ew_____eoo_neeeoaaeoo_____sephhrrhvh
hwwegmghhhhhhhkrrwwhhssHrrrvtrribbdbcbvs__thwwpppvmmirdnnib__eooooo
ooooo_____eennnnnnaai__ecc__ttttttttttttttts_tsgltsLlvt__hhoor
e_wrraddwlors_____r_lteirillre_ouaanooioeooooiihkiiiiio__iei
tsppioi_____ggnodsc_sss_gfhf_fffhwh_nsmo__uee_sioooaeeeeoo__ii
cgppeaoaeooesseuutetaaaaaaaaai__ei_in__aaie_eeerei_hrssnacciIi
iiiiisn_____oyoui__a_iids__aiaee_____tlar
```

**rle(bwt) [464] :**

```
.dlms2ftysestrns_y_2$yfofe4tg2sfefefg2e2drofr,l2re2f-,fs,9nfrsdn2
herghet2edndete2ge2nste2,s5t,es3ns2f2te2dt10r,4e3feh2_2p_2fpDw11e2h
l_ew_5eo2_ne3oa2eo2_4seph2r2hvh2w2egmgh7kr2w2h2s2Hr3vtr2ib2dbcbvs_2t
hw2p3vm2irdn2ib_2eo12_4e2n6a2i_3ec2_2t18s_tsgltsLlvt2_3h2o2re_wr2ad2
wlors_9r_2lteiril2re_oua2no2i2oeo4i3hki6o_2ieitsp2ioi_12g2nodsc_s3_g
fhf_f3hwh_nsmo_2ue2_sio3ae4o2_i2cgp2e2aoaeo2e2s2eu2tet11i_2ei_in_2a
2ie_e3rei_hrs3nac2i2Ii7sn_15oyoui_2a_i3ds_2ai2ae2_21tlar
```

# Run Length Encoding

**ref[614] :**

It\_was\_the\_best\_of\_times,\_it\_was\_the\_worst\_of\_times,\_it\_was\_the\_age\_of\_wisdom,\_it\_was\_the\_age\_of\_foolishness,\_it\_was\_the\_epoch\_of\_belief,\_it\_was\_the\_epoch\_of\_incredulity,\_it\_was\_the\_season\_of\_Light,\_it\_wa\_s\_the\_season\_of\_Darkness,\_it\_was\_the\_spring\_of\_hope,\_it\_was\_the\_wint\_er\_of\_despair,\_we\_had\_everything\_before\_us,\_we\_had\_nothing\_before\_us,\_we\_were\_all\_going\_direct\_to\_Heaven,\_we\_were\_all\_going\_direct\_the\_o ther\_way\_-\_in\_short,\_the\_period\_was\_so\_far\_like\_the\_present\_period,\_that\_some\_of\_its\_noisiest\_authorities\_insisted\_on\_its\_being\_received,\_for\_good\_or\_for\_evil,\_in\_the\_superlative\_degree\_of\_comparison\_only.\$

**rle(bwt) [464] :**

.dlms2ftysesdtrsns\_y\_2\$\_yfofe4tg2sfefefg2e2drofr,l2re2f-,fs,9nfrsdn2 hereghet2edndete2ge2nste2,s5t,es3ns2f2te2dt10r,4e3feh2\_2p\_2fpDw11e2h l\_ew\_5eo2\_ne3oa2eo2\_4seph2r2hvh2w2egmgh7kr2w2h2s2Hr3vtr2ib2dbcbvs\_2t hw2p3vm2irdn2ib\_2eo12\_4e2n6a2i\_3ec2\_2t18s\_tsgltsLlvt2\_3h2o2re\_wr2ad2 wlors\_9r\_2lteiril2re\_oua2no2i2oeo4i3hki6o\_2ieitsp2ioi\_12g2nodsc\_s3\_g fhf\_f3hwh\_nsmo\_2ue2\_sio3ae4o2\_i2cgp2e2aoaao2e2s2eu2tet11i\_2ei\_in\_2a 2ie\_e3rei\_hrs3nac2i2Ii7sn\_15oyoui\_2a\_i3ds\_2ai2ae2\_21tlar

# Run Length Encoding

**ref[614]:**

It\_was\_the\_best\_of\_times,\_it\_was\_the\_worst\_of\_times,\_it\_was\_the\_age\_of\_wisdom,\_it\_was\_the\_age\_of\_foolishness,\_it\_was\_the\_epoch\_of\_belief,\_it\_was\_the\_epoch\_of\_incredulity,\_it\_was\_the\_season\_of\_Light,\_it\_wa\_s\_the\_season\_of\_Darkness,\_it\_was\_the\_spring\_of\_hope,\_it\_was\_the\_wint\_er\_of\_despair,\_we\_had\_everything\_before\_us,\_we\_had\_nothing\_before\_us,\_we\_were\_all\_going\_direct\_to\_Heaven,\_we\_were\_all\_going\_direct\_the\_o ther\_way\_-\_in\_short,\_the\_period\_was\_so\_far\_like\_the\_present\_period,\_that\_some\_of\_its\_noisiest\_authorities\_insisted\_on\_its\_being\_received,\_for\_good\_or\_for\_evil,\_in\_the\_superlative\_degree\_of\_comparison\_only.\$

**rle(bwt) [464]:**

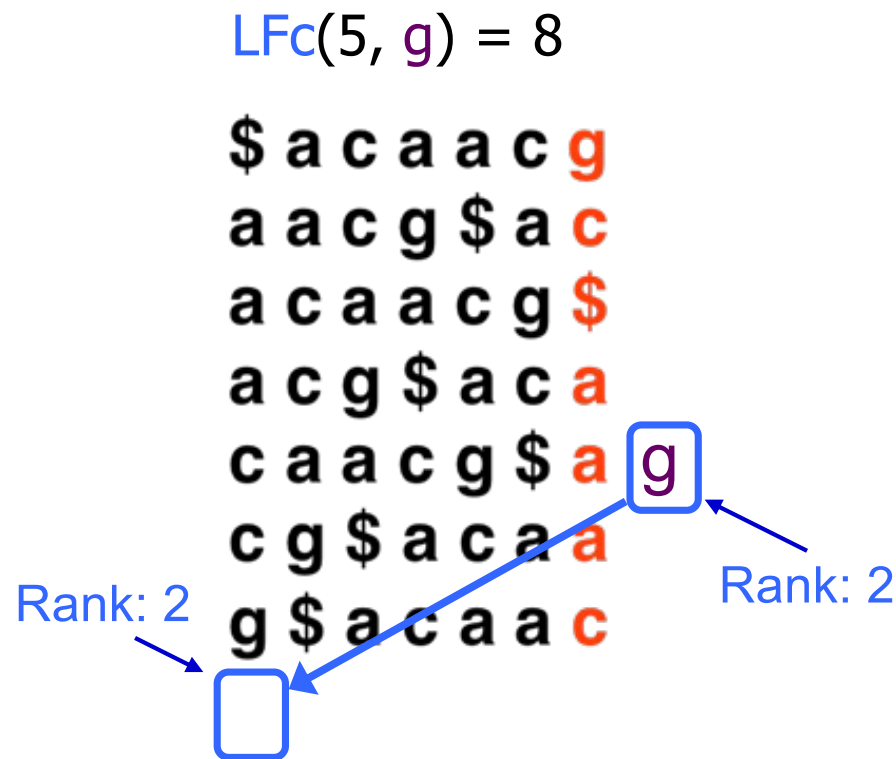
.dlms2ftysesdtrsns\_y\_2\$\_yfofe4tg2sfefefg2e2drofr,l2re2f-,fs,9nfrsdn2 hereghet2edndete2ge2nste2,s5t,es3ns2f2te2dt10r,4e3feh2\_2p\_2fpDw11e2h l\_ew\_5eo2\_ne3oa2eo2\_4seph2r2hvh2w2egmgh7kr2w2h2s2Hr3vtr2ib2dbcbvs\_2t hw2p3vm2irdn2ib\_2eo12\_4e2n6a2i\_3ec2\_2t18s\_tsgltsLlvt2\_3h2o2re\_wr2ad2 wlors\_9r\_2lteiril2re\_oua2no2i2oeo4i3hki6o\_2ieitsp2ioi\_12g2nodsc\_s3\_g fhf\_f3hwh\_nsmo\_2ue2\_sio3ae4o2\_i2cgp2e2aoaao2e2s2eu2tet11i\_2ei\_in\_2a 2ie\_e3rei

**Saved 614-464 = 150 bytes (24%) with zero loss of information!**

**Common to save 50% to 90% on real world files with bzip2**

# BWT Exact Matching

- **LFc**(r, c) does the same thing as **LF**(r) but it ignores r's actual final character and “pretends” it's c:



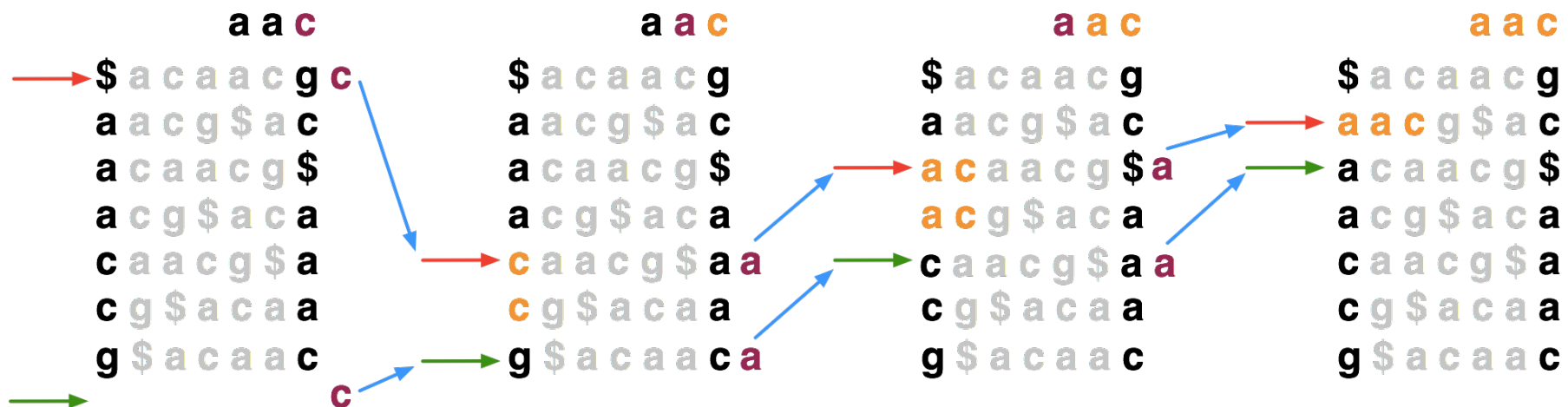


# BWT Exact Matching

- Start with a range, (**top**, **bot**) encompassing all rows and repeatedly apply **LFc**:

**top** = **LFc**(**top**, **qc**); **bot** = **LFc**(**bot**, **qc**)

**qc** = the next character to the left in the query



Ferragina P, Manzini G: Opportunistic data structures with applications. *FOCS. IEEE Computer Society; 2000.*

[Search for TTA this BWT string: ACTGA\$TTA ]



PREPRINT

# BWT construction and search at the terabase scale

Heng Li<sup>1,2,3,\*</sup>

<sup>1</sup>Department of Data Science, Dana-Farber Cancer Institute, 450 Brookline Ave, Boston, MA 02215, USA, <sup>2</sup>Department of Biomedical Informatics, Harvard Medical School, 10 Shattuck St, Boston, MA 02215, USA and <sup>3</sup>Broad Institute of MIT and Harvard, 415 Main St, Cambridge, MA 02142, USA

\*Corresponding author. [hli@ds.dfci.harvard.edu](mailto:hli@ds.dfci.harvard.edu)

## Abstract

**Motivation:** Burrows-Wheeler Transform (BWT) is a common component in full-text indices. Initially developed for data compression, it is particularly powerful for encoding redundant sequences such as pangenome data. However, BWT construction is resource intensive and hard to be parallelized, and many methods for querying large full-text indices only report exact matches or their simple extensions. These limitations have hampered the biological applications of full-text indices.

**Results:** We developed ropebwt3 for efficient BWT construction and query. Ropebwt3 could index 100 assembled human genomes in 21 hours and index 7.3 terabases of commonly studied bacterial assemblies in 26 days. This was achieved using 82 gigabytes of memory at the peak without working disk space. Ropebwt3 can find maximal exact matches and inexact alignments under affine-gap penalties, and can retrieve all distinct local haplotypes matching a query sequence. It demonstrates the feasibility of full-text indexing at the terabase scale.

**Availability and implementation:** <https://github.com/lh3/ropebwt3>

# Algorithm Overview

## 1. Split read into segments

Read

**CCAGTAGCTCTCAGCCTTATTTTACCCAGGCCTGTA**

Read (reverse complement)

**TACAGGCCTGGGTAAATAAGGCTGAGAGCTACTGG**

Policy: extract 16 nt seed every 10 nt

## Seeds

+, 0: CCAGTAGCTCTCAGCC

+, 10: **TCAGCCTTATTTACC**

+, 20: **TTACCCAGGCCTGTA**

-, 0: **TACAGGCCTGGGTAAA**

-, 10: GGTAAAATAAGGCTGA

-, 20: GGCTGAGAGCTACTGG

## 2. Lookup each segment and prioritize

## Seeds

+, 0: CCAGTAGCTCTCAGCC

+, 10: **TCAGCCTTATTTACC**

+, 20: **TTACCCAGGCCTGTA**

-, 0: **TACAGGCCTGGGTAAA**

-, 10: GGTAAAATAAGGCTGA

-, 20: GGCTGAGAGCTACTGG

Ungapped  
alignment with  
FM Index

Diagram illustrating a sequence alignment between a reference sequence and a query sequence. The reference sequence is "a a c" and the query sequence is "\$ a c a a c g". The alignment shows the query sequence shifted to the right, with red arrows indicating the alignment of "c" to "c" and "a" to "a".

### Seed alignments (as B ranges)

$$\{ [211, 212], [212, 214] \}$$

$\{ [653, 654], [651, 653] \}$

**{ [684, 685] }**

{ }

{ }

{ [624, 625] }

### 3. Evaluate end-to-end match

## Extension candidates

SA:684, chr12:1955

SA:624, chr2:462

SA:211: chr4:762

SA:213: chr12:1935

SA:652: chr12:1945

SIMD dynamic  
programming  
aligner

A 10x10 grid illustrating the cumulative distribution function of the number of successes in 10 trials. The grid is divided into blue and green regions. The blue region represents the probability of 0 to 6 successes, and the green region represents the probability of 7 to 10 successes. The cumulative probabilities are labeled in the grid cells: 0, 0.35, 0.65, 0.85, 0.95, 0.99, 1.00, 1.00, 1.00, 1.00.

## SAM alignments

```

r1      0      chr12      1936      0
36M *      0      0
CCAGTAGCTCTCAGCCTTATTTTACCCAGGCCTGTA
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
AS:i:0    XS:i:-2    XN:i:0
XM:i:0    XO:i:0    XG:i:0
NM:i:0    MD:Z:36    YT:Z:UU
YM:i:0

```

...

(Langmead & Salzberg, 2012)

# Similarity metrics

- Hamming distance

- Count the number of substitutions to transform one string into another

MIKESCHATZ

| | X | | XXXX |

MICESHATZZ

5

- Edit distance

- The minimum number of substitutions, insertions, or deletions to transform one string into another

MIKESCHAT-Z

| | X | | X | | | X |

MICES-HATZZ

3

# Edit Distance Example

AGCACACA → ACACACTA in 4 steps

AGCACACA → (1. change G to C)

ACCACACA → (2. delete C)

ACACACA → (3. change A to T)

ACACACT → (4. insert A after T)

ACACACTA → done

[Is this the best we can do?]

# Edit Distance Example

AGCACACA → ACACACTA in 3 steps

AGCACACA → (1. change G to C)

ACCACACA → (2. delete C)

ACACACA → (3. insert T after 3<sup>rd</sup> C)

ACACACTA → done

[Is this the best we can do?]

# Reverse Engineering Edit Distance

$$D(\text{AGCACACA}, \text{ACACACTA}) = ?$$

Imagine we already have the optimal alignment of the strings, the last column can only be 1 of 3 options:

... <b>M</b>	... <b>I</b>	... <b>D</b>
...A	...-	...A
...A	...A	...-

The optimal alignment of last two columns is then 1 of 9 possibilities

... <b>MM</b>	... <b>IM</b>	... <b>DM</b>	... <b>MI</b>	... <b>II</b>	... <b>DI</b>	... <b>MD</b>	... <b>ID</b>	... <b>DD</b>
...CA	...-A	...CA	...A-	...--	...A-	...CA	...-A	...CA
...TA	...TA	...-A	...TA	...TA	...-A	...A-	...A-	...--

The optimal alignment of the last three columns is then 1 of 27 possibilities...

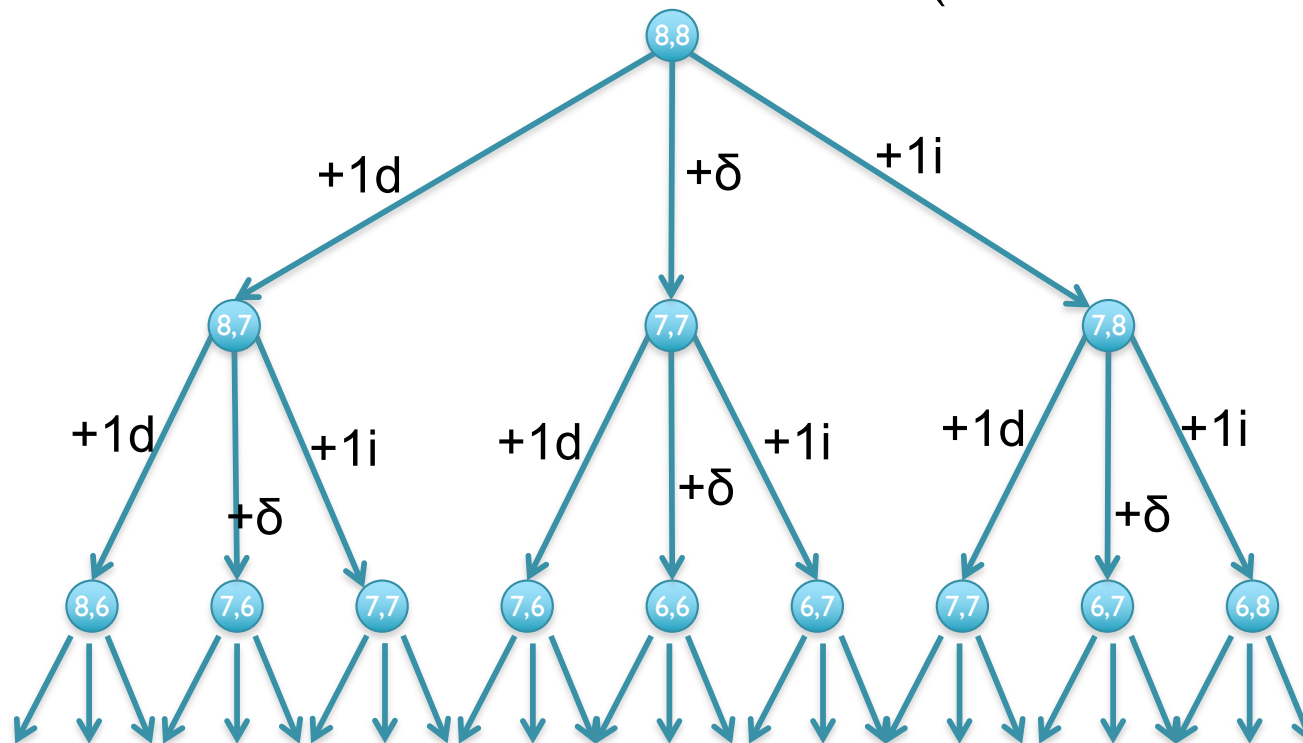
... <b>M</b> ...	... <b>I</b> ...	... <b>D</b> ...
...X...	...-...	...X...
...Y...	...Y...	...-...

Eventually spell out every possible sequence of {I,M,D}

# Recursive solution

- Computation of  $D$  is a recursive process.
  - At each step, we only allow matches, substitutions, and indels
  - $D(i,j)$  in terms of  $D(i',j')$  for  $i' \leq i$  and  $j' \leq j$ .

$$D(\text{AGCACACA}, \text{ACACACTA}) = \min \{ D(\text{AGCACACA}, \text{ACACACT}) + 1, \\ D(\text{AGCACAC}, \text{ACACACTA}) + 1, \\ D(\text{AGCACAC}, \text{ACACACT}) + \delta(\text{A}, \text{A}) \}$$



[What is the running time?]



# Dynamic Programming

- We could code this as a recursive function call...  
...with an exponential number of function evaluations
- There are only  $(n+1) \times (m+1)$  pairs  $i$  and  $j$ 
  - We are evaluating  $D(i,j)$  multiple times
- Compute  $D(i,j)$  bottom up.
  - Start with smallest  $(i,j) = (1,1)$ .
  - Store the intermediate results in a table.
    - Compute  $D(i,j)$  *after*  $D(i-1,j)$ ,  $D(i,j-1)$ , and  $D(i-1,j-1)$

# Recurrence Relation for D

Find the edit distance (minimum number of operations to convert one string into another) in  $O(mn)$  time

- Base conditions:

- $D(i,0) = i$ , for all  $i = 0, \dots, n$
- $D(0,j) = j$ , for all  $j = 0, \dots, m$

- For  $i > 0, j > 0$ :

$$D(i,j) = \min \left\{ \begin{array}{ll} D(i-1,j) + 1, & // \text{align } 0 \text{ chars from } S, 1 \text{ from } T \\ D(i,j-1) + 1, & // \text{align } 1 \text{ chars from } S, 0 \text{ from } T \\ D(i-1,j-1) + \delta(S(i),T(j)) & // \text{align } 1+1 \text{ chars} \end{array} \right\}$$

[Why do we want the min?]

# Dynamic Programming Matrix

		<b>A</b>	<b>C</b>	<b>A</b>	<b>C</b>	<b>A</b>	<b>C</b>	<b>T</b>	<b>A</b>
	0	1	2	3	4	5	6	7	8
<b>A</b>	1								
<b>G</b>	2								
<b>C</b>	3								
<b>A</b>	4								
<b>C</b>	5								
<b>A</b>	6								
<b>C</b>	7								
<b>A</b>	8								

[What does the initialization mean?]

# Dynamic Programming Matrix

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

$$D[A,A] = \min\{D[A,]+1, D[,A]+1, D[,]+ \delta(A,A)\}$$

# Dynamic Programming Matrix

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

$$D[A,AC] = \min\{D[A,A]+1, D[,AC]+1, D[,A]+\delta(A,C)\}$$

# Dynamic Programming Matrix

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

$$D[A,ACA] = \min\{D[A,AC]+1, D[,ACA]+1, D[,AC]+\delta(A,A)\}$$

# Dynamic Programming Matrix

		<b>A</b>	<b>C</b>	<b>A</b>	<b>C</b>	<b>A</b>	<b>C</b>	<b>T</b>	<b>A</b>
	<u>0</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>	<u>8</u>
<b>A</b>	1	0	1	2	3	4	5	6	<u>7</u>
<b>G</b>	2								
<b>C</b>	3								
<b>A</b>	4								
<b>C</b>	5								
<b>A</b>	6								
<b>C</b>	7								
<b>A</b>	8								

$$D[A, ACACACTA] = 7$$

-----A

\*\*\*\*\*|

ACACACTA

[What about the other A?]

# Dynamic Programming Matrix

		<b>A</b>	<b>C</b>	<b>A</b>	<b>C</b>	<b>A</b>	<b>C</b>	<b>T</b>	<b>A</b>
	<u>0</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>	<u>8</u>
<b>A</b>	1	0	1	2	3	<u>4</u>	5	6	7
<b>G</b>	2	1	1	2	3	4	<u>5</u>	<u>6</u>	<u>7</u>
<b>C</b>	3								
<b>A</b>	4								
<b>C</b>	5								
<b>A</b>	6								
<b>C</b>	7								
<b>A</b>	8								

$$D[AG, ACACACTA] = 7$$

-----AG--

\*\*\*\*\* | \*\*\*

ACACACTA



# Dynamic Programming Matrix

		A	C	A	C	A	C	T	A
	<u>0</u>	1	2	3	4	5	6	7	8
A	1	<u>0</u>	1	2	3	4	5	6	7
G	2	<u>1</u>	1	2	3	4	5	6	7
C	3	2	<u>1</u>	2	2	3	4	5	6
A	4	3	2	<u>1</u>	2	2	3	4	5
C	5	4	3	2	<u>1</u>	2	2	3	4
A	6	5	4	3	2	<u>1</u>	2	3	3
C	7	6	5	4	3	2	<u>1</u>	<u>2</u>	3
A	8	7	6	5	4	3	2	2	<u>2</u>

$$D[\text{AGCACACA}, \text{ACACACTA}] = 2$$

AGCACAC-A

|\*| | | | |\*|

A-CACACTA

[Can we do it any better?]