Variant Calling

Michael Schatz

October 3, 2022

Lecture 10: Applied Comparative Genomics

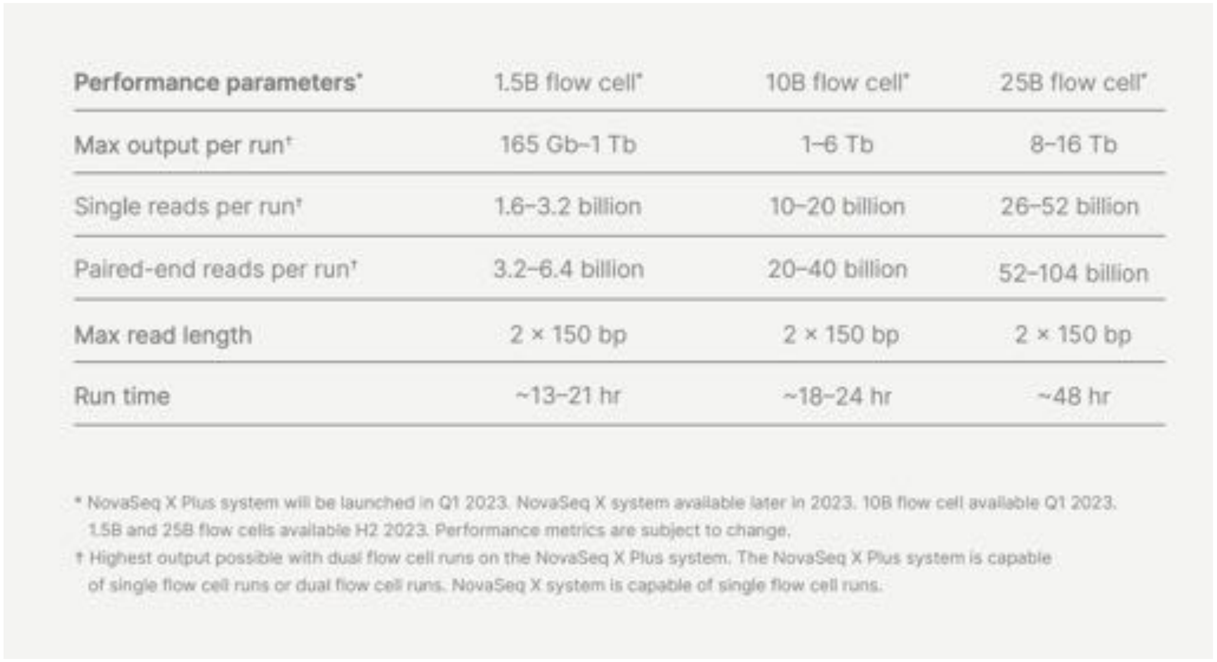
THE NOBEL PRIZE IN PHYSIOLOGY OR MEDICINE 2022

Illustration: Niklas Elmehed

Svante Pääbo

"for his discoveries concerning the genomes of extinct hominins and human evolution"

THE NOBEL ASSEMBLY AT KAROLINSKA INSTITUTET



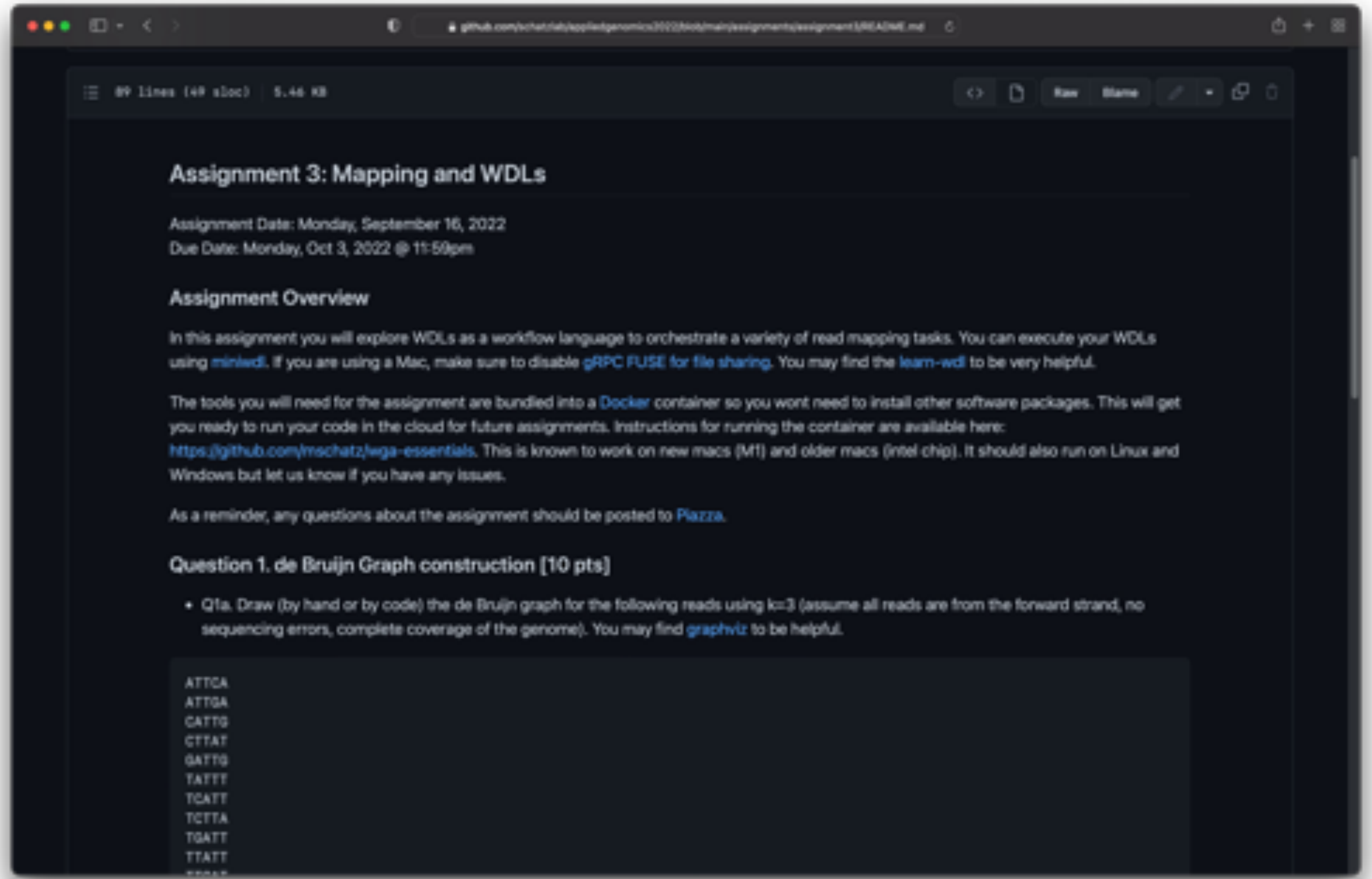
Performance parameters*	1.5B flow cell*	10B flow cell*	25B flow cell*
Max output per run†	165 Gb–1 Tb	1–6 Tb	8–16 Tb
Single reads per run†	1.6–3.2 billion	10–20 billion	26–52 billion
Paired-end reads per run†	3.2–6.4 billion	20–40 billion	52–104 billion
Max read length	2 × 150 bp	2 × 150 bp	2 × 150 bp
Run time	~13–21 hr	~18–24 hr	~48 hr

* NovaSeq X Plus system will be launched in Q1 2023. NovaSeq X system available later in 2023. 10B flow cell available Q1 2023. 1.5B and 25B flow cells available H2 2023. Performance metrics are subject to change.

† Highest output possible with dual flow cell runs on the NovaSeq X Plus system. The NovaSeq X Plus system is capable of single flow cell runs or dual flow cell runs. NovaSeq X system is capable of single flow cell runs.

Illumina says its NovaSeq X machine will get the price of sequencing down to $200 per human genome.

Assignment 3: Mapping and WDL
Due Monday Oct 3 by 11:59pm



89 lines (49 sloc) | 5.46 KB

Assignment 3: Mapping and WDLs

Assignment Date: Monday, September 16, 2022
Due Date: Monday, Oct 3, 2022 @ 11:59pm

Assignment Overview

In this assignment you will explore WDLs as a workflow language to orchestrate a variety of read mapping tasks. You can execute your WDLs using miniwdl. If you are using a Mac, make sure to disable gRPC FUSE for file sharing. You may find the learn-wdl to be very helpful.

The tools you will need for the assignment are bundled into a Docker container so you wont need to install other software packages. This will get you ready to run your code in the cloud for future assignments. Instructions for running the container are available here: https://github.com/mschatz/wga-essentials. This is known to work on new macs (M1) and older macs (intel chip). It should also run on Linux and Windows but let us know if you have any issues.

As a reminder, any questions about the assignment should be posted to Piazza.

Question 1. de Bruijn Graph construction [10 pts]

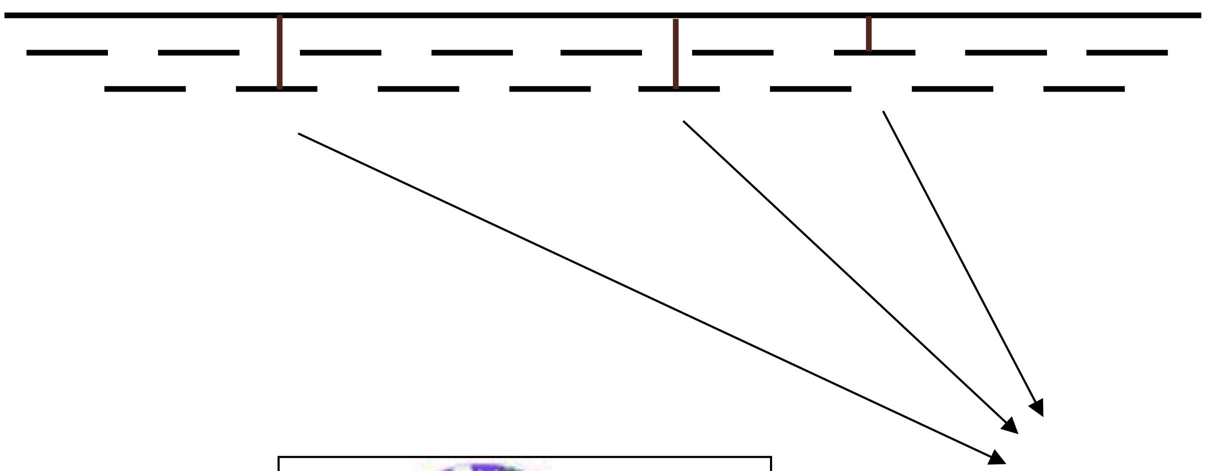
- Q1a. Draw (by hand or by code) the de Bruijn graph for the following reads using k=3 (assume all reads are from the forward strand, no sequencing errors, complete coverage of the genome). You may find graphviz to be helpful.

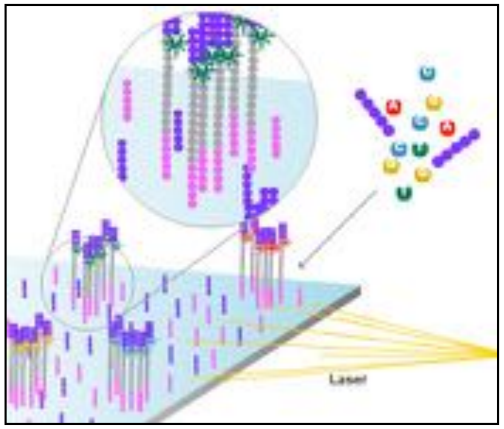
```
ATTCA
ATTGA
CATTG
CTTAT
GATTG
TATTT
TCATT
TCTTA
TGATT
TTATT
```

https://github.com/schatzlab/appliedgenomics2022/tree/main/assignments/assignment3
Check Piazza for questions!

Personal Genomics

How does your genome compare to the reference?





Heart Disease

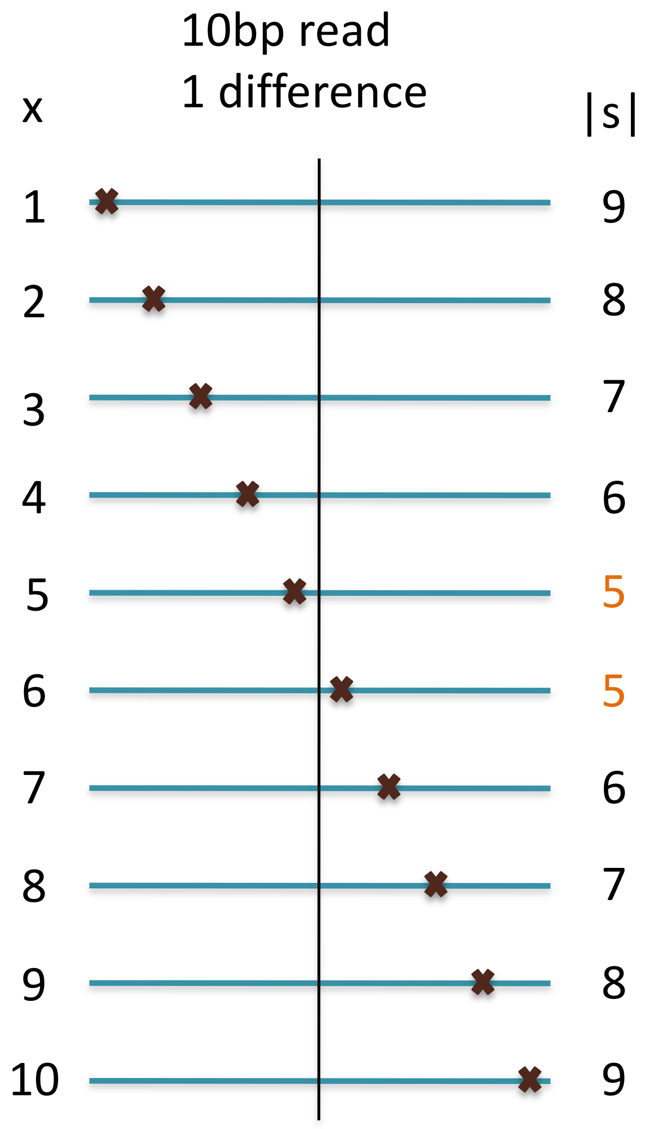
Cancer

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

Exact Matching Review & Overview

Where is GATTACA in the human genome?

Brute Force
(3 GB)

```
BANANA
BAN
 ANA
  NAN
   ANA
```

O(m * n)

Slow & Easy

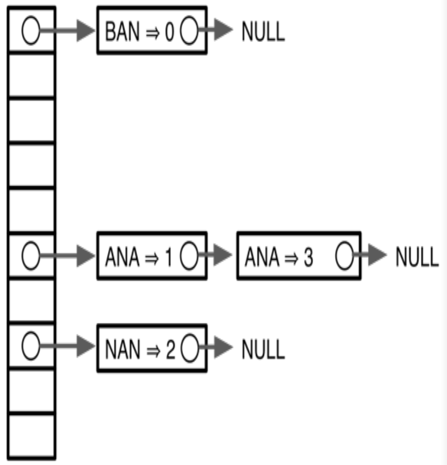
Suffix Array
(>15 GB)

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

O(m + lg n)

Full-text index

Hash Table
(>15 GB)



O(1)

Fixed-length lookup

BWT
(3 GB)

BANANA$

=>

$BANANA
A$BANAN
ANA$BAN
ANANA$B
BANANA$
NA$BANA
NANA$BA

=>

ANNB$AA

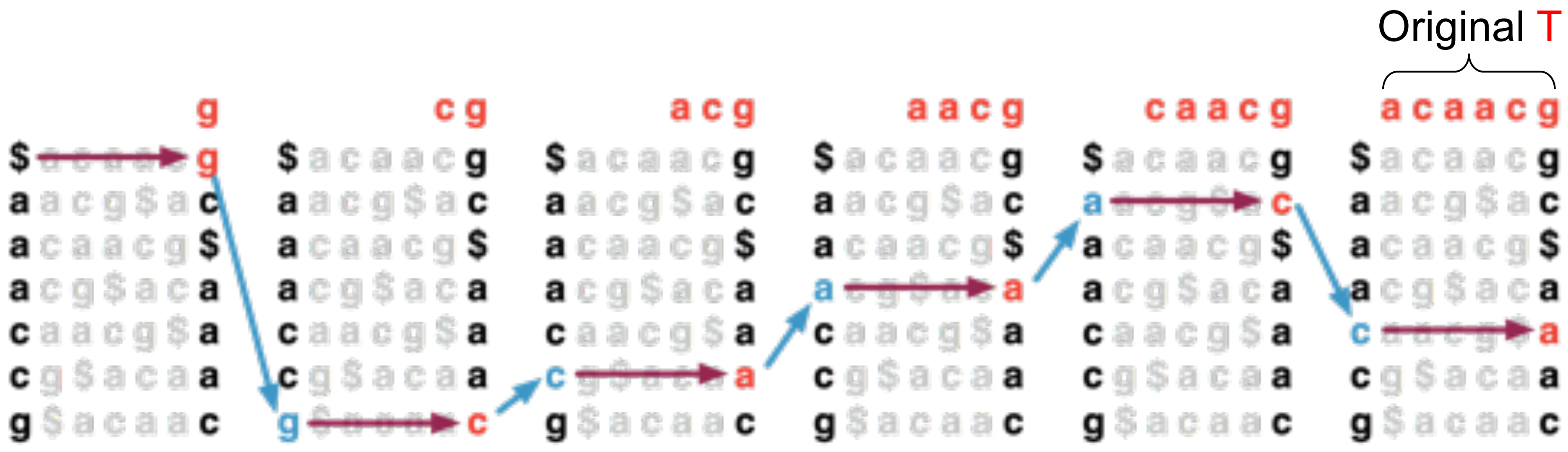
O(m)

Full-text and concise

*** These are general techniques applicable to any text search problem ***

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

```
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.$
```

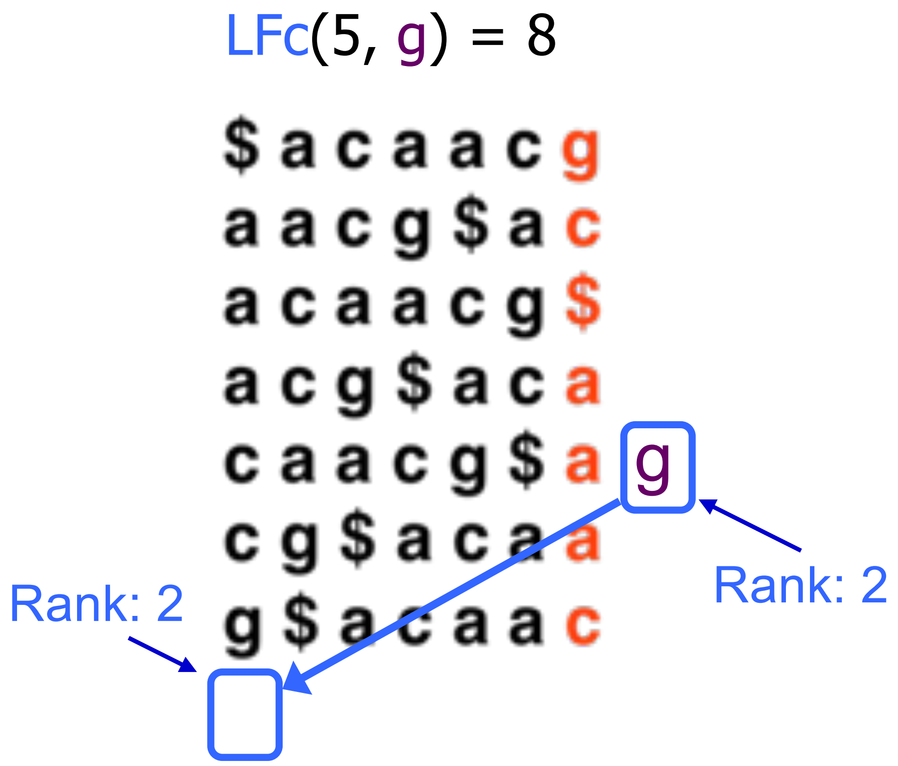
```
rle(bwt)[464]:
.dlms2ftysesdtrsns_y_2$_yfofe4tg2sfefefg2e2drofr,l2re2f-,fs,9nfrsdn2
hereghet2edndete2ge2nste2,s5t,es3ns2f2te2dt10r,4e3feh2_2p_2fpDw11e2h
l_ew_5eo2_ne3oa2eo2_4seph2r2hvh2w2egmgh7kr2w2h2s2Hr3vtr2ib2dbcbvs_2t
hw2p3vm2irdn2ib_2eo12_4e2n6a2i_3ec2_2t18s_tsgltsLlvt2_3h2o2re_wr2ad2
wlors_9r_2lteiril2re_oua2no2i2oeo4i3hki6o_2ieitsp2ioi_12g2nodsc_s3_g
fhf_f3hwh_nsmo_2ue2_sio3ae4o2_i2cgp2e2aoaeo2e2s2eu2teta11i_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:

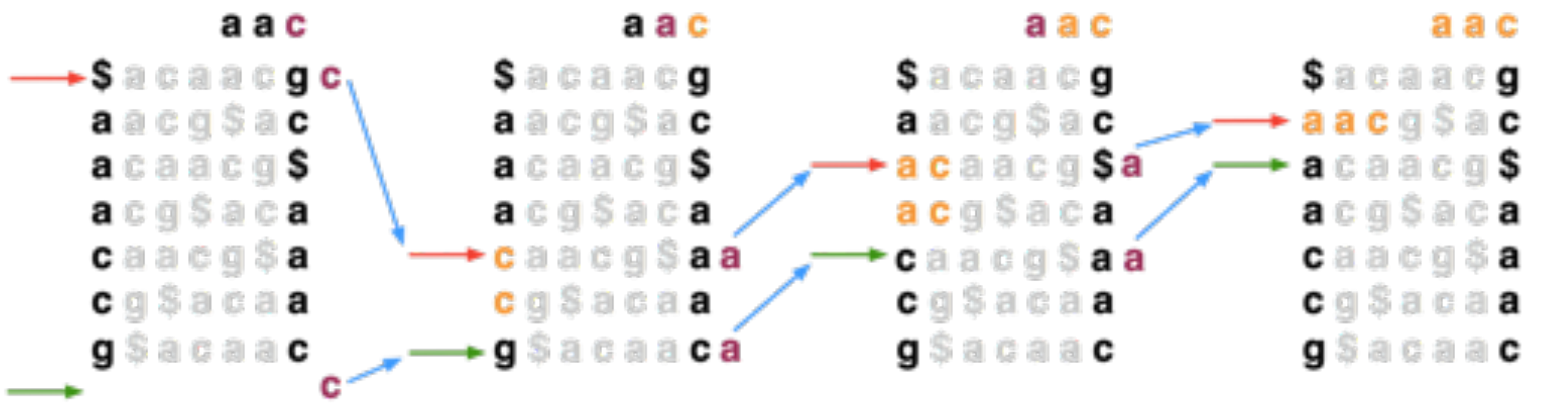


BWT Exact Matching

- 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

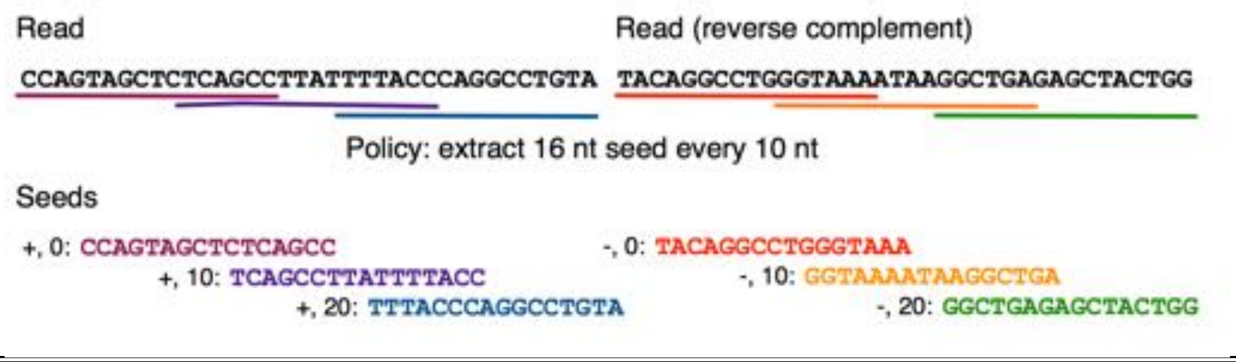


Ferragina P, Manzini G: Opportunistic data structures with applications. *FOCS. IEEE Computer Society; 2000.*

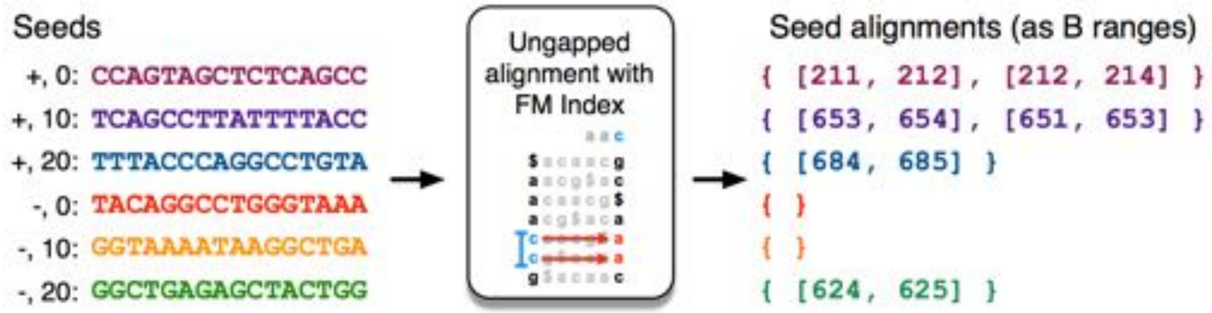
[Search for TTA this BWT string: ACTGA$TTA]

Algorithm Overview

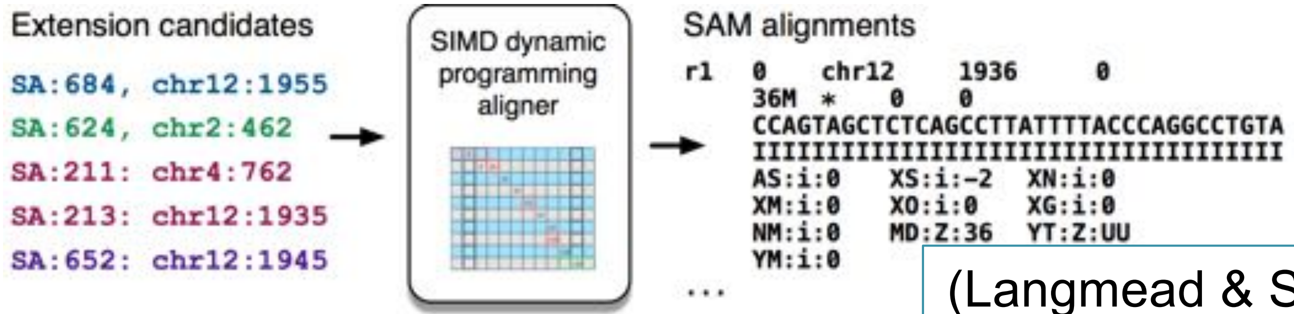
1. Split read into segments



2. Lookup each segment and prioritize



3. Evaluate end-to-end match



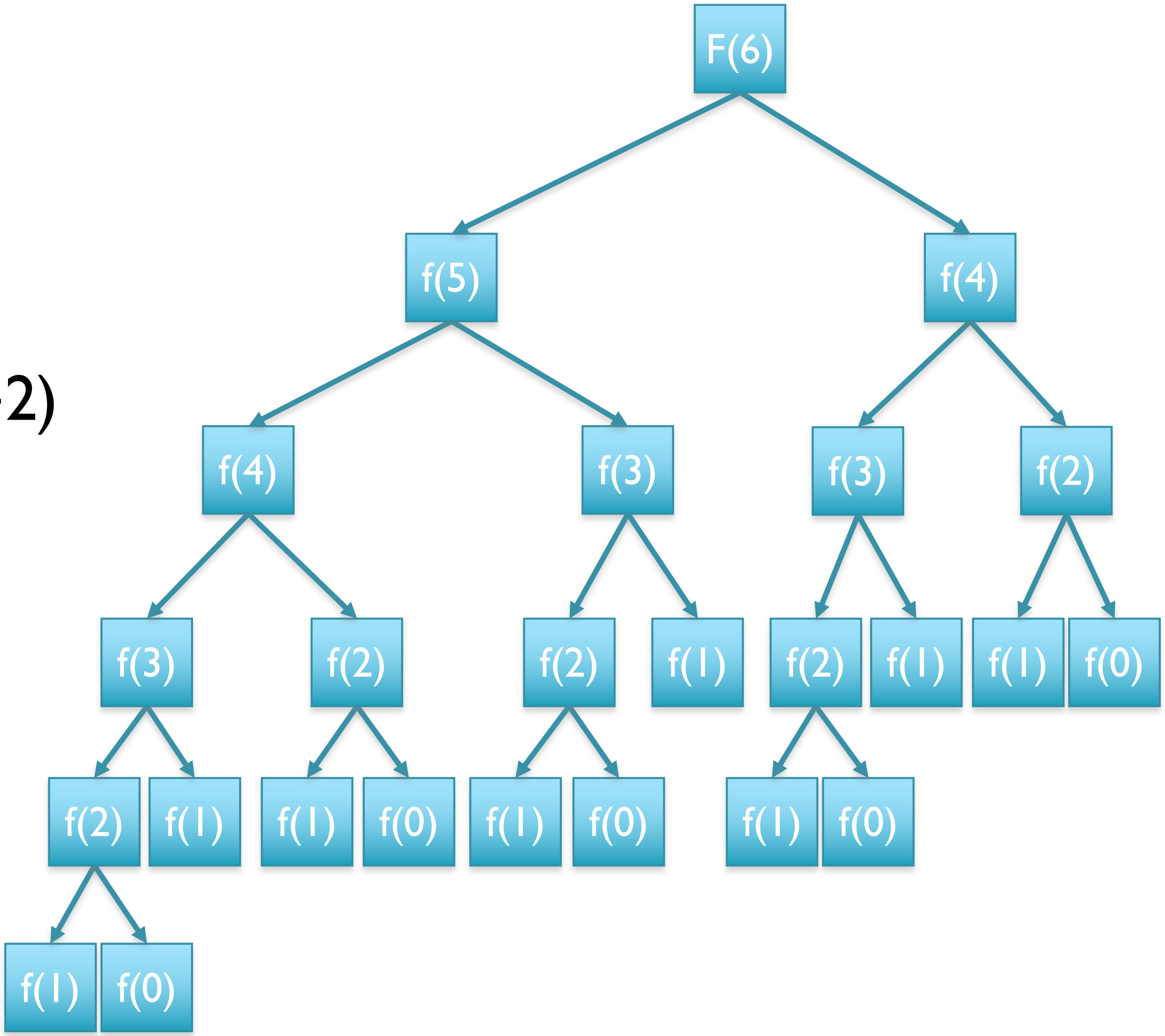
```
r1   0    chr12    1936     0
     36M  *    0    0
     CCAGTAGCTCTCAGCCTTATTTTACCCAGGCCTGTA
     IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
     AS:i:0   XS:i:-2   XN:i:0
     XM:i:0   XO:i:0    XG:i:0
     NM:i:0   MD:Z:36   YT:Z:UU
     YM:i:0
...
```

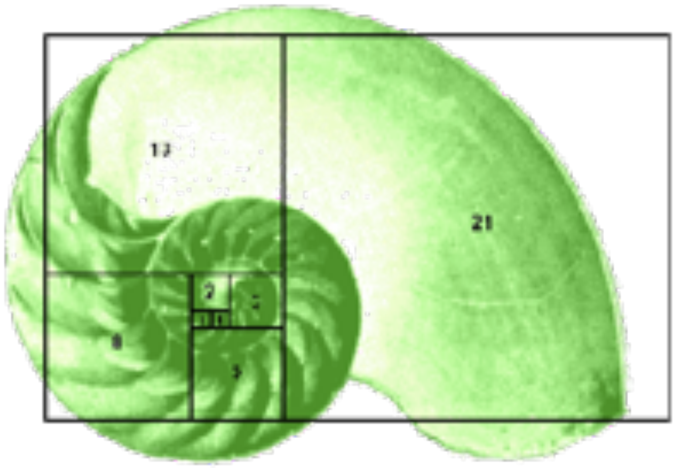
(Langmead & Salzberg, 2012)

Part 2: Dynamic Programming

Fibonacci Sequence

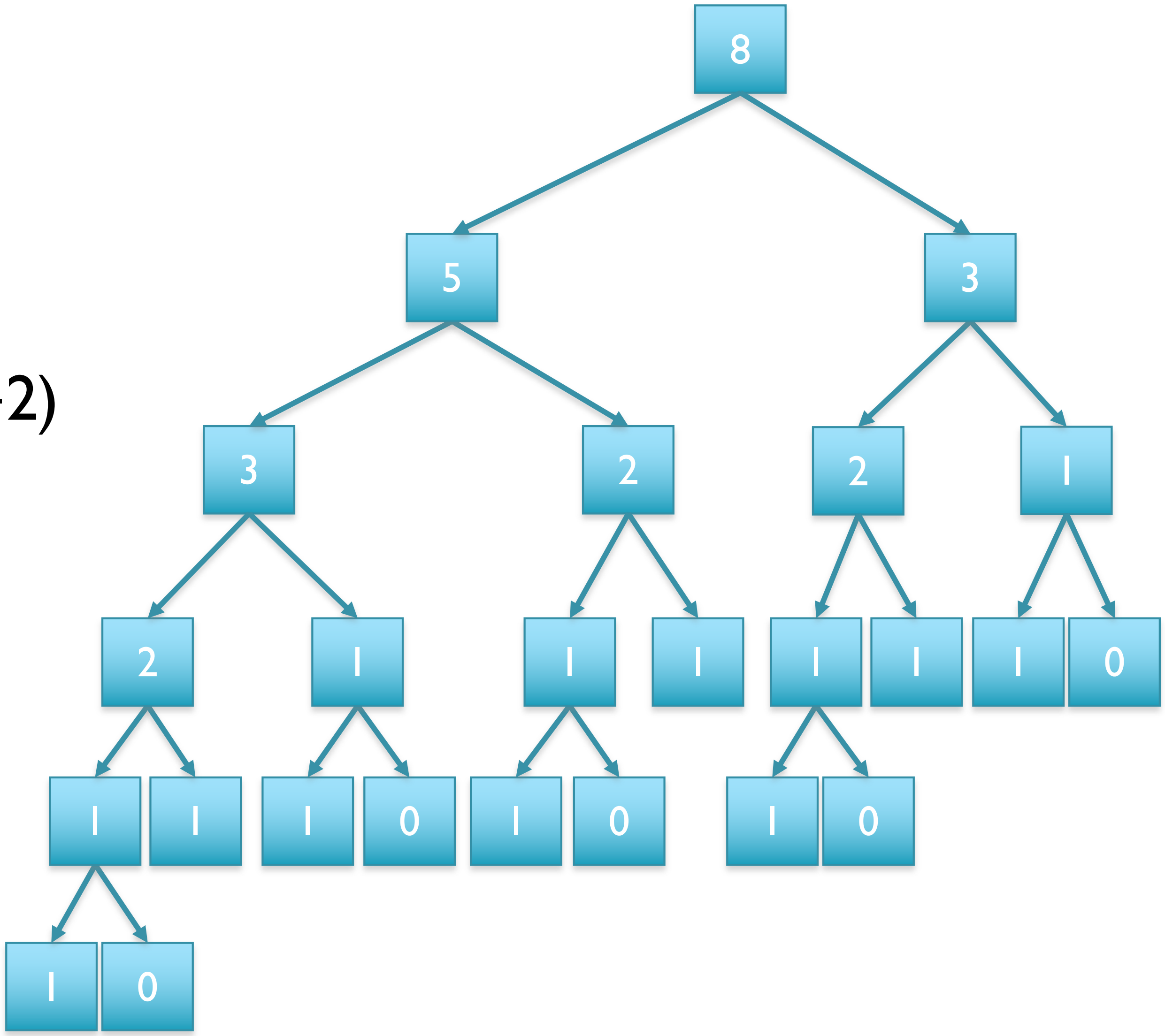
```
def fib(n):
  if n == 0 or n == 1:
     return n
  else:
  return fib(n−1) + fib(n−2)
```

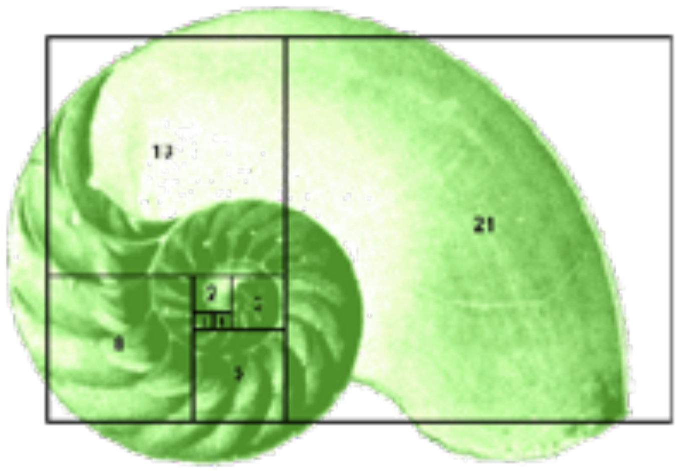




Fibonacci Sequence

```
def fib(n):
  if n == 0 or n == 1:
     return n
  else:
  return fib(n−1) + fib(n−2)
```





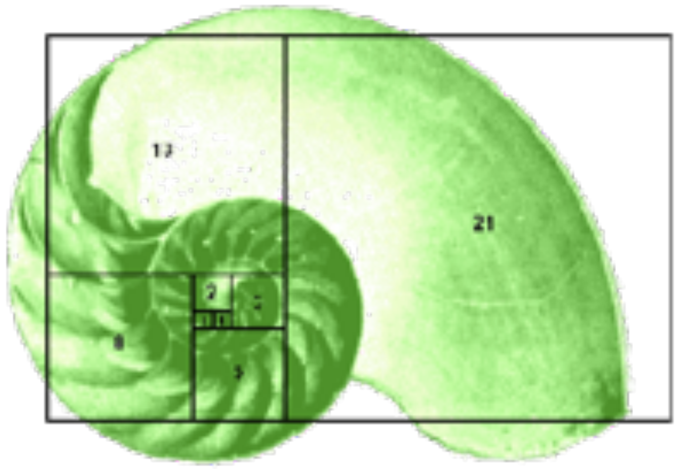
[What is the running time?]

Bottom-up Fibonacci Sequence

```
def fib(n):
  table = [0] * (n+1)
  table[0] = 0
  table[1] = 1
  for i in range(2,n+1):
    table[i] = table[i–2] + table[i–1]
return table[n]
```

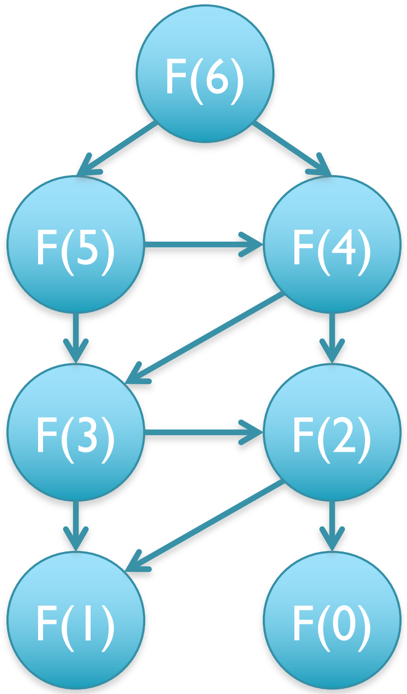
0	1	2	3	4	5	6
0	1	1	2	3	5	8

[What is the running time?]



Dynamic Programming

- General approach for solving (some) complex problems
  - When applicable, the method takes far less time than naive methods.
    - Polynomial time ($O(n)$ or $O(n^2)$) instead of exponential time ($O(2^n)$ or $O(3^n)$)
- Requirements:
  - **Overlapping subproblems**
  - **Optimal substructure**
- Applications:
  - Fibonacci
  - Longest Increasing Subsequence
  - Sequence alignment, Dynamic Time Warp, Viterbi



- Not applicable:
  - Traveling salesman problem, Clique finding, Subgraph isomorphism, …
  - The cheapest flight from airport A to airport B involves a single connection through airport C, but the cheapest flight from airport A to airport C involves a connection through some other airport D.

In-exact alignment

- Where is GATTACA *approximately* in the human genome?
  - And how do we efficiently find them?

- It depends…
  - Define 'approximately'
    - Hamming Distance, Edit distance, or Sequence Similarity
    - Ungapped vs Gapped vs Affine Gaps
    - Global vs Local
    - All positions or the single 'best'?
  - Efficiency depends on the data characteristics & goals
    - Smith-Waterman: Exhaustive search for optimal alignments
    - BLAST: Hash-table based homology searches
    - Bowtie: BWT alignment for short read mapping

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 3rd C)

ACACACTA → done

[Is this the best we can do?]

Reverse Engineering Edit Distance

```
D(AGCACACA, ACACACTA) = ?
```

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

```
…M      …I      …D
…A      …-      …A
…A      …A      …-
```

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

```
…MM  …IM  …DM      …MI  …II  …DI      …MD  …ID  …DD
…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…

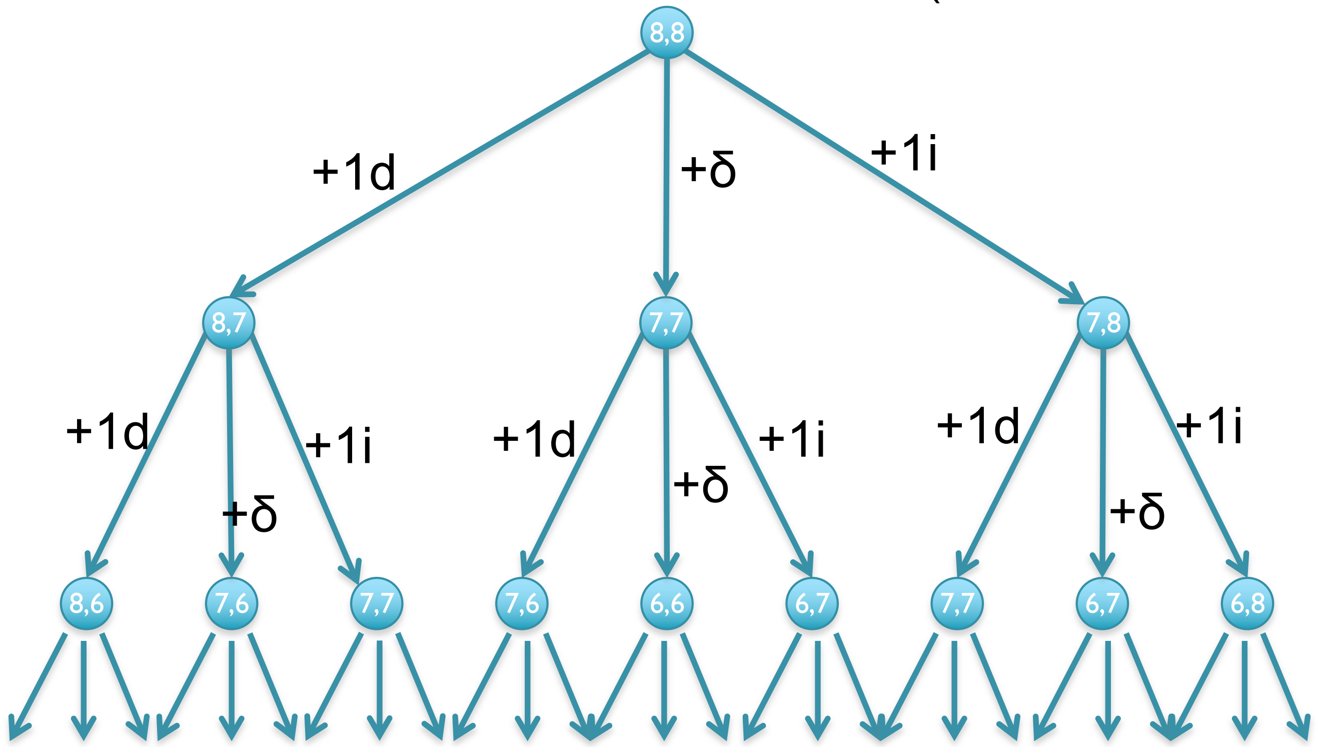
```
…M…     …I…     …D…
…X…     …-…     …X…
…Y…     …Y…     …-…
```

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' ≤ i and j' ≤ j.

D(AGCACACA, ACACACTA) = min{D(AGCACACA, ACACACT) + 1,
D(AGCACAC, ACACACTA) + 1,
D(AGCACAC, ACACACT) +δ(A, 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) x (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 $i = 0,\ldots,n$
  - $D(0,j) = j$, for all $j = 0,\ldots,m$

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

```
D(i,j) = min {
        D(i-1,j) + 1,            // align 0 chars from S, 1 from T
        D(i,j-1) + 1,            // align 1 chars from S, 0 from T
        D(i-1,j-1) + δ(S(i),T(j)) // align 1+1 chars
    }
```

[Why do we want the min?]

Dynamic Programming Matrix

		A	C	A	C	A	C	T	A
	0	1	2	3	4	5	6	7	8
A	1								
G	2								
C	3								
A	4								
C	5								
A	6								
C	7								
A	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]+δ(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]+δ(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	2	3	4	5	6	7
G	2								
C	3								
A	4								
C	5								
A	6								
C	7								
A	8								

D[A,ACACACTA] = 7

```
-------A
*******|
ACACACTA
```

[What about the other 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	2	3	4	5	6	7
G	2	1	1	2	3	4	5	6	7
C	3								
A	4								
C	5								
A	6								
C	7								
A	8								

D[AG,ACACACTA] = 7

```
----AG--
****|***
ACACACTA
```

Dynamic Programming Matrix

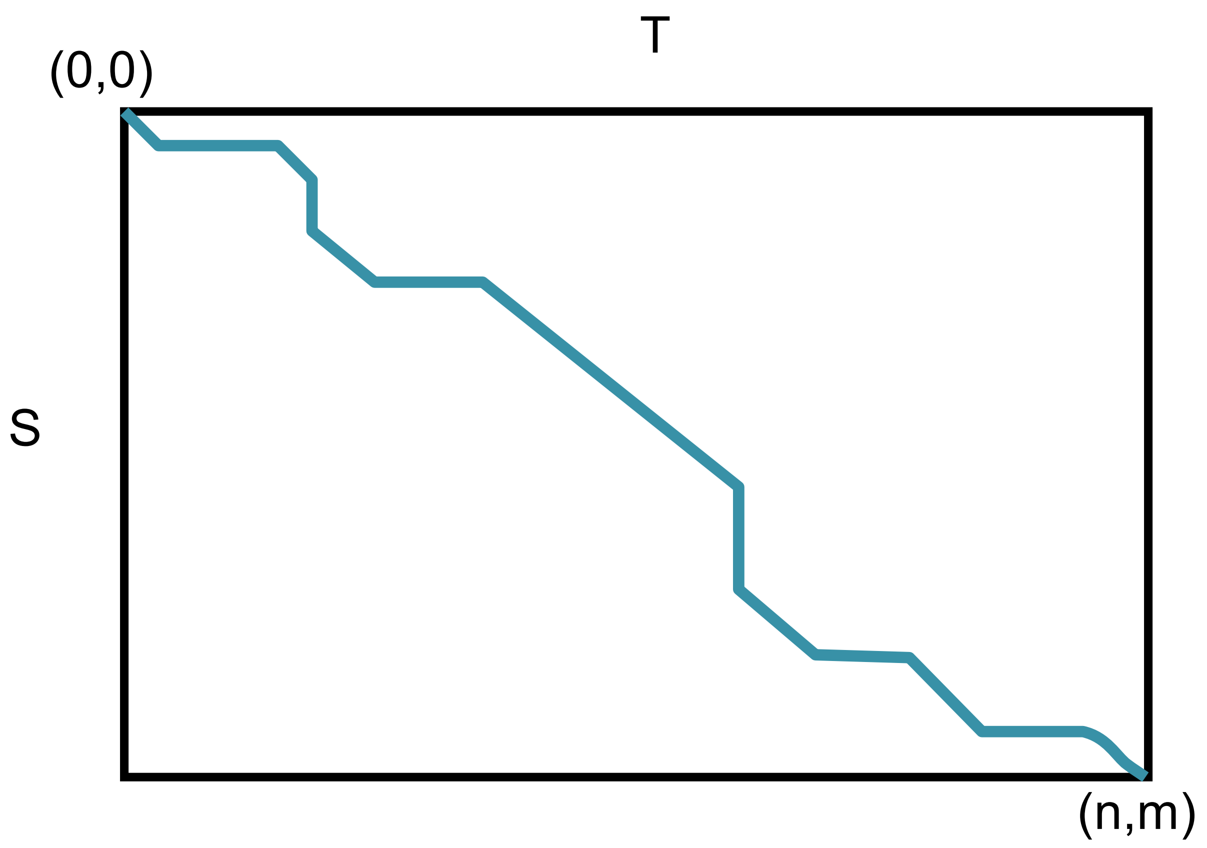
		A	**C**	**A**	**C**	**A**	**C**	**T**	**A**
	$\underline{0}$	1	2	3	4	5	6	7	8
A	1	$\underline{0}$	1	2	3	4	5	6	7
G	2	$\underline{1}$	1	2	3	4	5	6	7
C	3	2	$\underline{1}$	2	2	3	4	5	6
A	4	3	2	$\underline{1}$	2	2	3	4	5
C	5	4	3	2	$\underline{1}$	2	2	3	4
A	6	5	4	3	2	$\underline{1}$	2	3	3
C	7	6	5	4	3	2	$\underline{1}$	$\underline{2}$	3
A	8	7	6	5	4	3	2	2	$\underline{2}$

D[AGCACACA,ACACACTA] = 2

```
AGCACAC-A
|*|||||*|
A-CACACTA
```

[Can we do it any better?]

Global Alignment Schematic



- A high quality alignment will stay close to the diagonal
  - If we are only interested in high quality alignments, we can skip filling in cells that can't possibly lead to a high quality alignment
  - Find the global alignment with at most edit distance d: O(2dn)

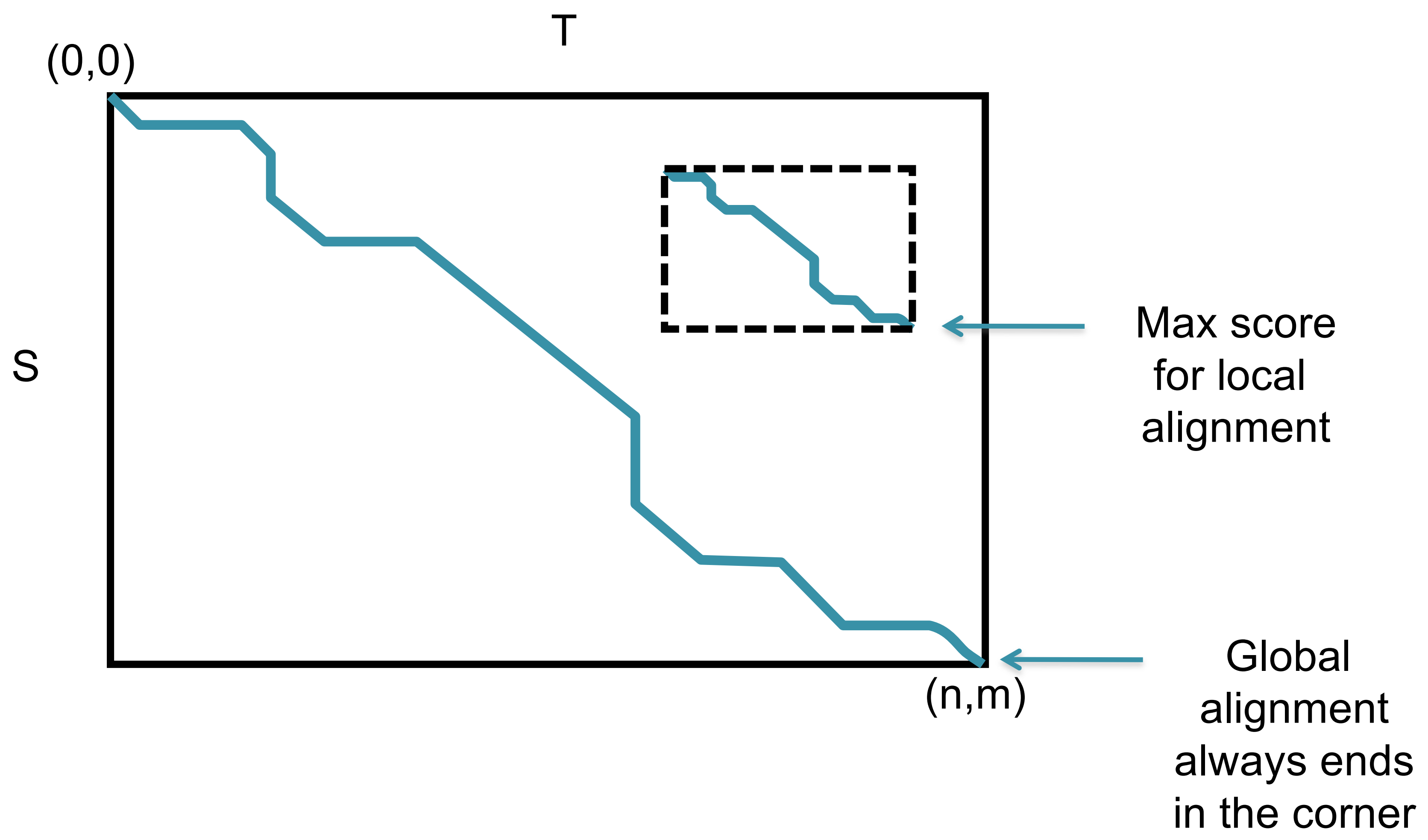
Nathan Edwards

Local vs. Global Alignment

- The Global Alignment Problem tries to find the best end-to-end alignment between the two strings
  - Only applicable for very closely related sequences

- The Local Alignment Problem tries to find pairs of **substrings** with highest similarity.
  - Especially important if one string is substantially longer than the other
  - Especially important if there is only a distant evolutionary relationship

Global vs Local Alignment Schematic


T
(0,0)
S
(n,m)
Max score for local alignment
Global alignment always ends in the corner

Nathan Edwards

Local vs. Global Alignment (cont’ d)

- Global Alignment

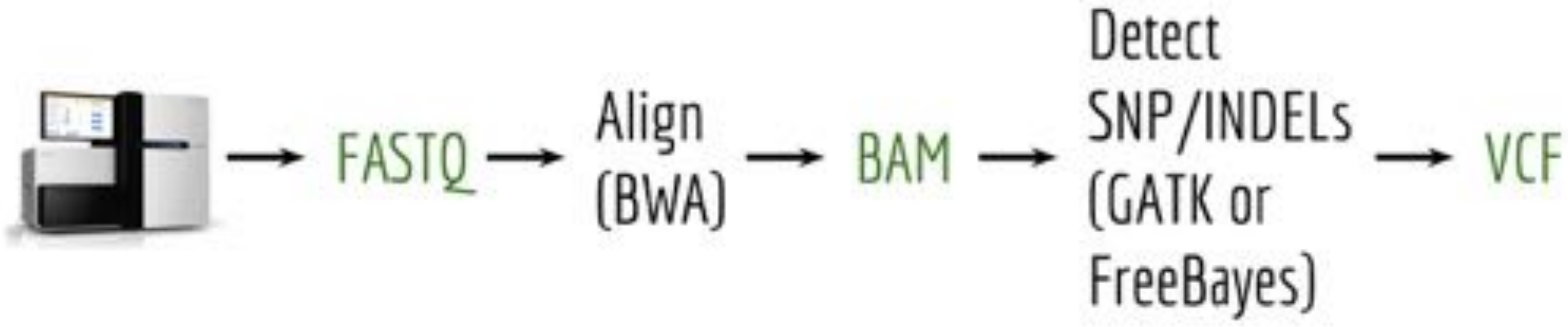
```
--T—-CC-C-AGT—-TATGT-CAGGGGACACG—A-GCATGCAGA-GAC
  |  || |  ||  | | | |||    || | | |  | ||||   |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG—T-CAGAT--C
```

- Local Alignment—better alignment to find conserved segment

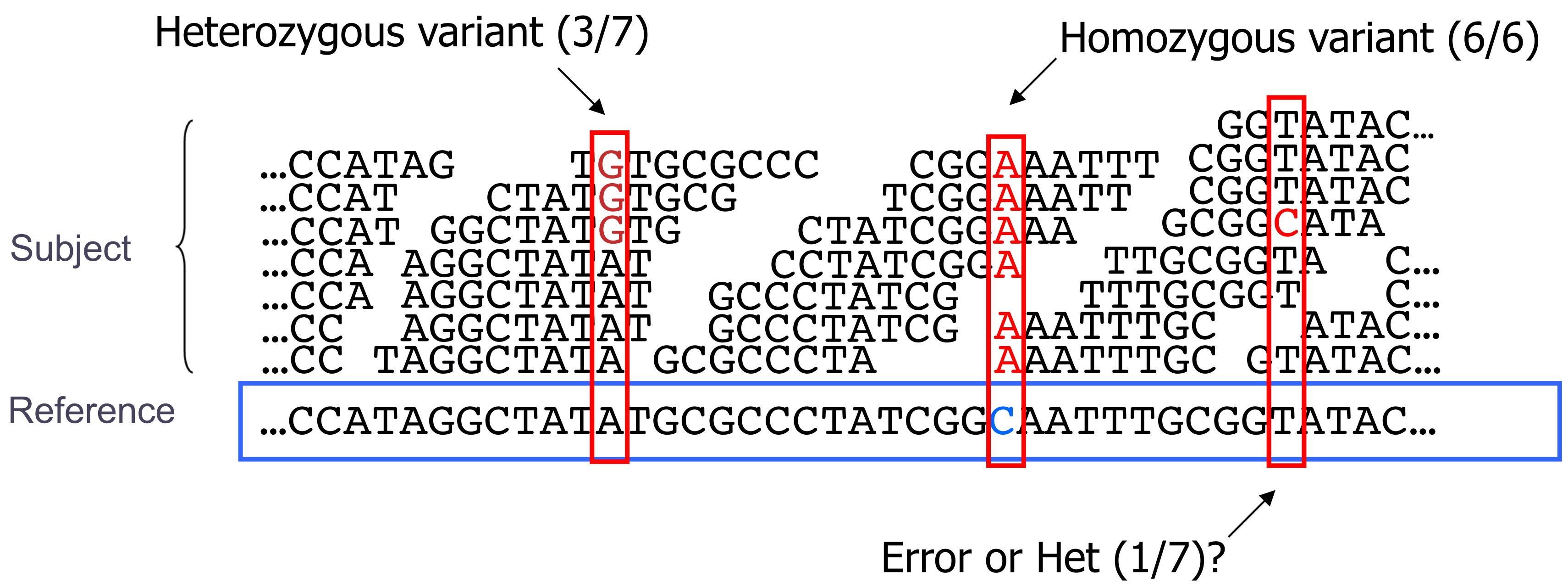
```
                  tccCAGTTATGTCAGgggacacgagcatgcagagac
                     ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

Part 3: Variant Calling

Variant Calling Overview



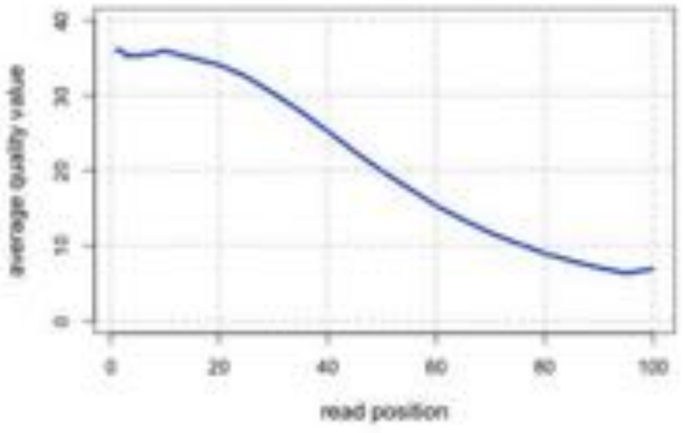
Genotyping Theory


Heterozygous variant (3/7)
Homozygous variant (6/6)
Subject
GGTATAC…
…CCATAG TGTGCGCCC CGGAAATTT CGGTATAC
…CCAT CTATGTGCG TCGGAAATT CGGTATAC
…CCAT GGCTATGTG CTATCGGAAA GCGGCATA
…CCA AGGCTATAT CCTATCGGA TTGCGGTA C…
…CCA AGGCTATAT GCCCTATCG TTTGCGGT C…
…CC AGGCTATAT GCCCTATCG AAATTTGC ATAC…
…CC TAGGCTATA GCGCCCTA AAATTTGC GTATAC…
Reference
…CCATAGGCTATATGCGCCCTATCGGCAATTTGCGGTATAC…
Error or Het (1/7)?

- If there were no sequencing errors, identifying SNPs would be very easy: any time a read disagrees with the reference, it must be a variant!

- Sequencing instruments make mistakes
  – Quality of read decreases over the read length

- A single read differing from the reference is probably just an error, but it becomes more likely to be real as we see it multiple times


average quality value
read position