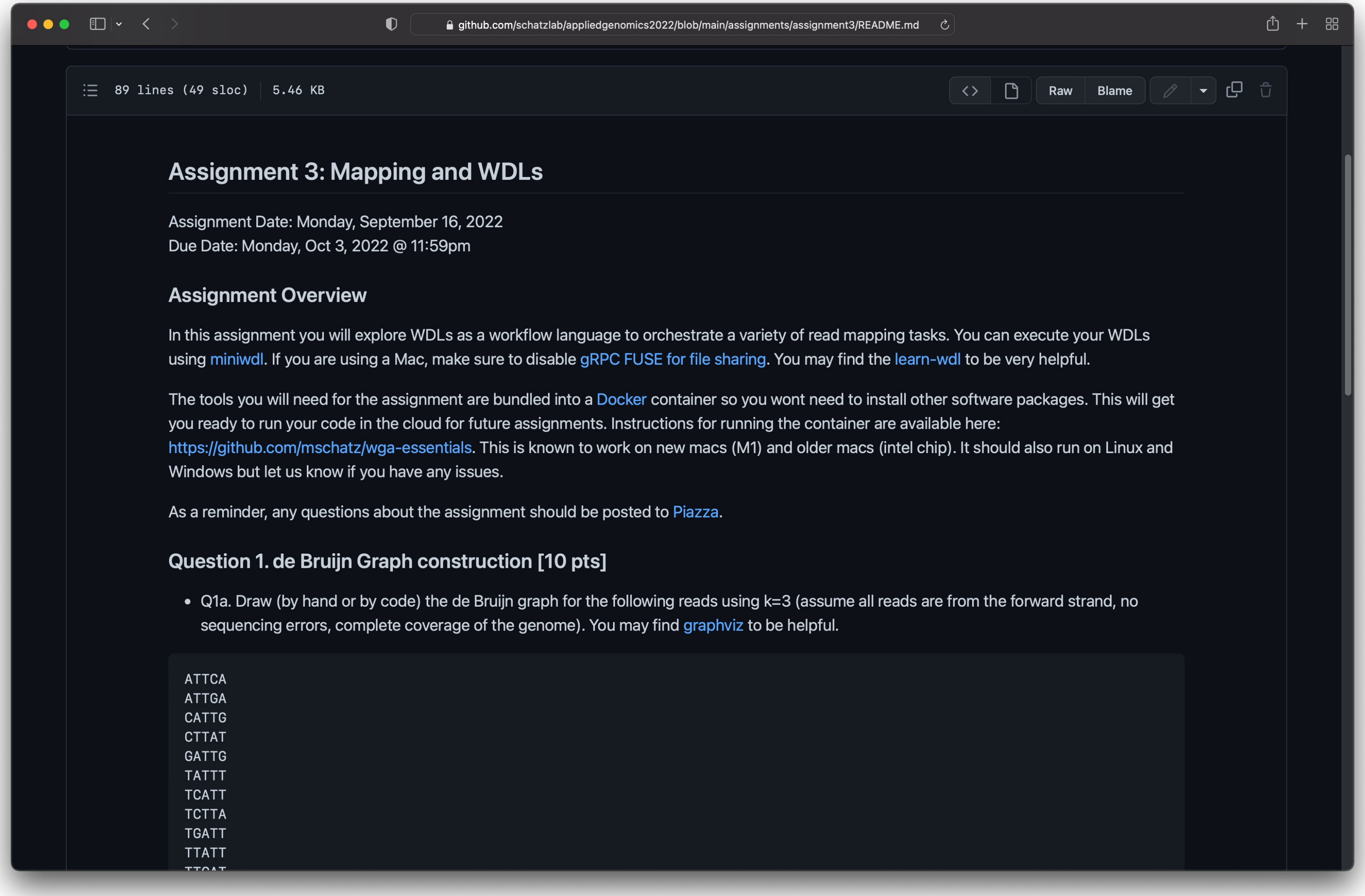
Read mapping and the bwt

Michael Schatz

Sept 26, 2022

Lecture 8: Applied Comparative Genomics

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


github.com/schatzlab/appliedgenomics2022/blob/main/assignments/assignment3/README.md
89 lines (49 sloc) | 5.46 KB
Raw
Blame

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
TTCAT
```

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

Parallel Algorithm Spectrum

Embarrassingly Parallel

Each item is Independent

Loosely Coupled

Independent-Sync-Independent

Tightly Coupled

Constant Sync

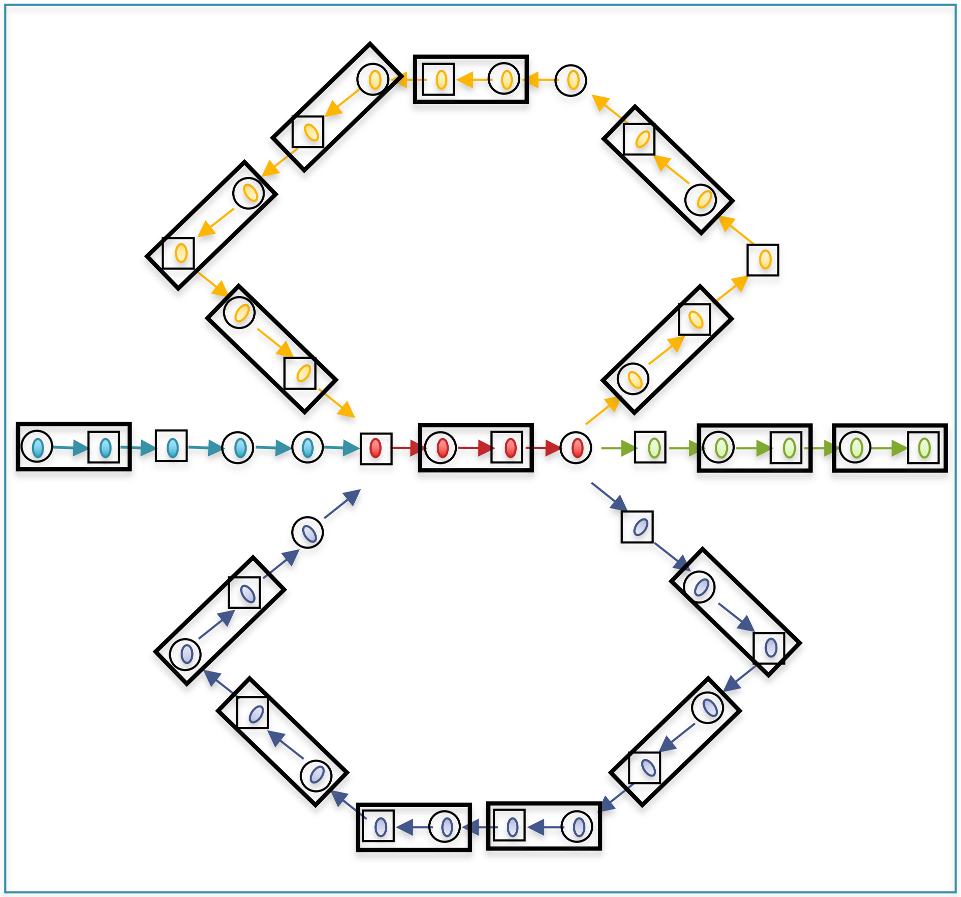
Fast Path Compression

Challenges

- Nodes stored on different computers
- Nodes can only access direct neighbors

Randomized List Ranking

- Randomly assign H/T to each compressible node
- Compress H→T links



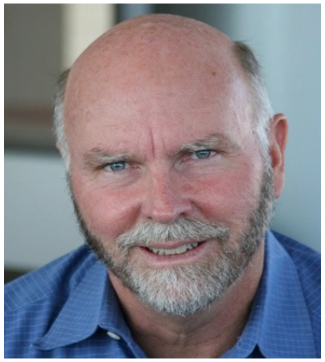
Initial Graph: 42 nodes

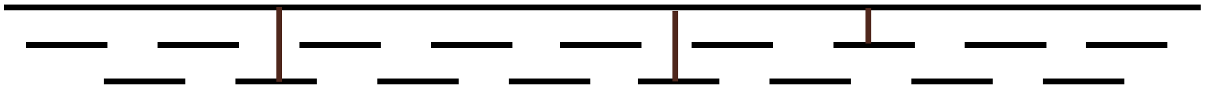
Randomized Speed-ups in Parallel Computation.
Vishkin U. (1984) *ACM Symposium on Theory of Computation. 230-239.*

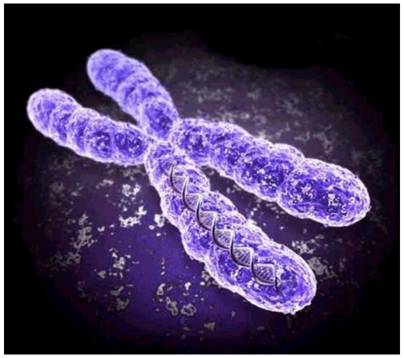
Read Mapping

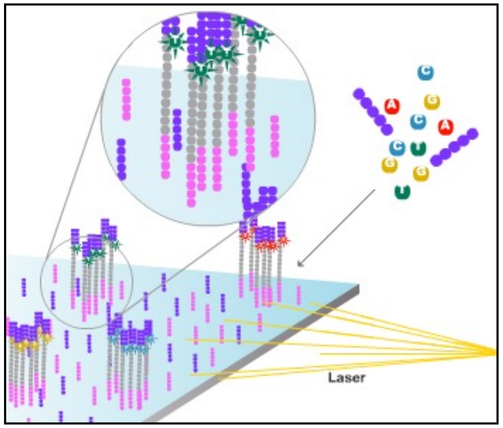
Personal Genomics

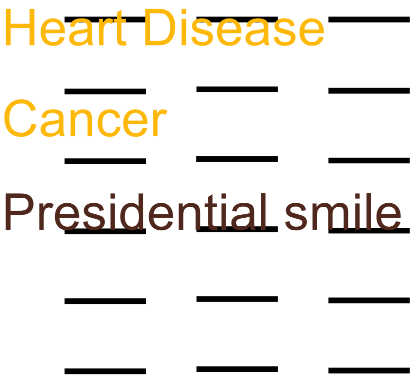
How does your genome compare to the reference?



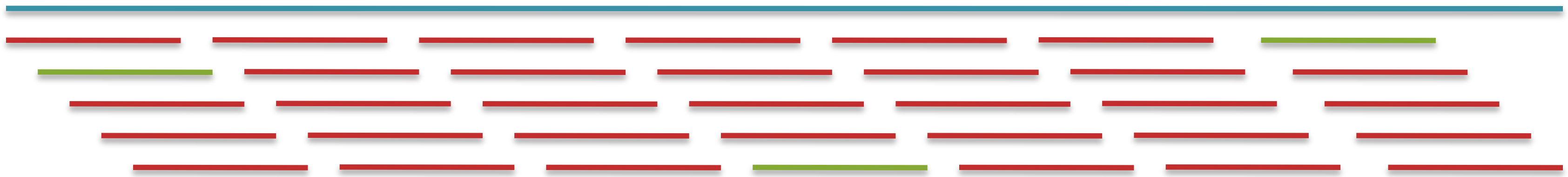








Brute Force Analysis



- 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?]

Brute Force Reflections

Why check every position?

– GATTACA can't possibly start at position 15 [WHY?]

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
T	G	A	T	T	A	C	A	G	A	T	T	A	C	C	…
								G	A	T	T	A	C	A	

– Improve runtime to O(n + m) [3B + 7]

- If we double both, it just takes twice as long
- Knuth-Morris-Pratt, 1977
- Boyer-Moyer, 1977, 1991

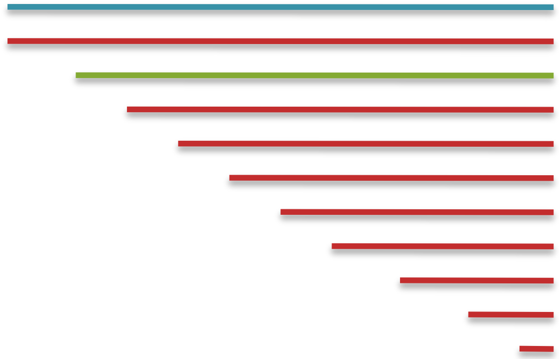
– For one-off scans, this is the best we can do (optimal performance)

- We have to read every character of the genome, and every character of the query
- For short queries, runtime is dominated by the length of the genome

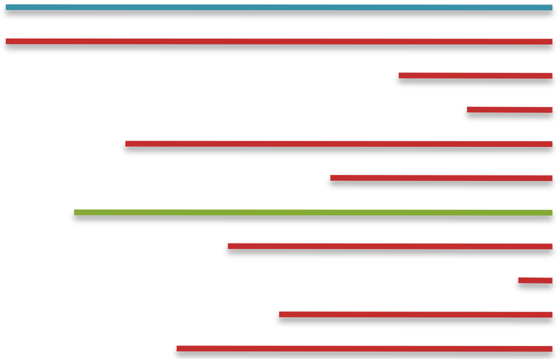
Suffix Arrays: Searching the Phone Book

- What if we need to check many queries?
  - We don't need to check every page of the phone book to find 'Schatz'
  - Sorting alphabetically lets us immediately skip 96% (25/26) of the book *without any loss in accuracy*

- Sorting the genome: Suffix Array (Manber & Myers, 1991)
  - Sort every suffix of the genome



Split into n suffixes



Sort suffixes alphabetically

[Challenge Question: How else could we split the genome?]

Searching the Index

- Strategy 2: Binary search
  - Compare to the middle, refine as higher or lower

- Searching for GATTACA
  - Lo = 1; Hi = 15;

Lo →

#	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

Hi →

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

Lo →

#	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

Hi →

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 →

#	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

Hi →

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;

	#	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
Lo →	9	GATTACAGATTACC…	2
	10	GATTACC…	9
	11	TACAGATTACC…	5
	12	TACC…	12
	13	TGATTACAGATTACC…	1
	14	TTACAGATTACC…	4
Hi →	15	TTACC…	11

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

#	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 →

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;

	#	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
Lo →	9	GATTACAGATTACC…	2
	10	GATTACC…	9
Hi →	11	TACAGATTACC…	5
	12	TACC…	12
	13	TGATTACAGATTACC…	1
	14	TTACAGATTACC…	4
	15	TTACC…	11

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

#	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 →

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;

Lo
Hi

#	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

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!

Lo
Hi

#	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

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?]

Binary Search Analysis

- Binary Search

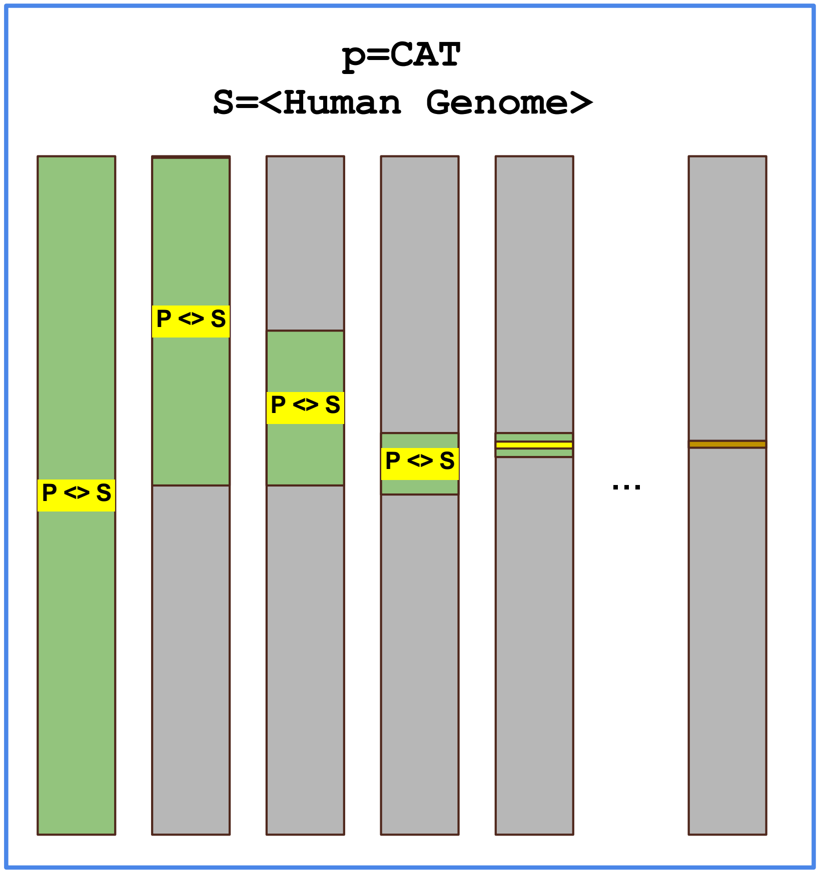
```
Initialize search range to entire list
    mid = (hi+lo)/2; middle = suffix[mid]
    if query matches middle: done
    else if query < middle: pick low range
    else if query > middle: pick hi range
Repeat until done or empty range                [WHEN?]
```

- 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: $n/(2^x) \le 1$; $x = \lg_2(n)$ [32]

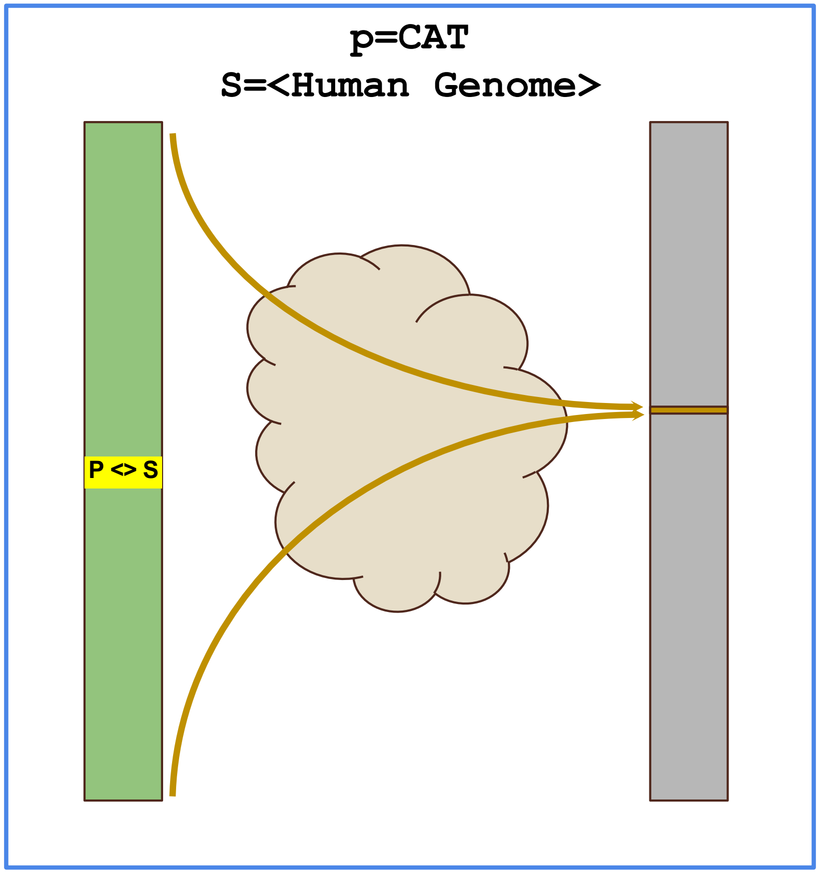
- Total Runtime: O(m lg n)
  - More complicated, but much faster!
  - Looking up a query loops 32 times instead of 3B

Can be reduced to O(m + lg n)
using an auxiliary data structure called the LCP array

Sapling: Accelerating Suffix Array Queries with Learned Data Models



p=CAT
S=<Human Genome>



p=CAT
S=<Human Genome>

What if instead of a slow algorithmic approach to find the correct rows, we could somehow quickly guess/predict the correct rows?

Kirsche, M, Das, A, Schatz, MC (2020) Bioinformatics
doi: https://doi.org/10.1093/bioinformatics/btaa911

Part 2: Burrows Wheeler Transform

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