

Edit Distance

Michael Schatz

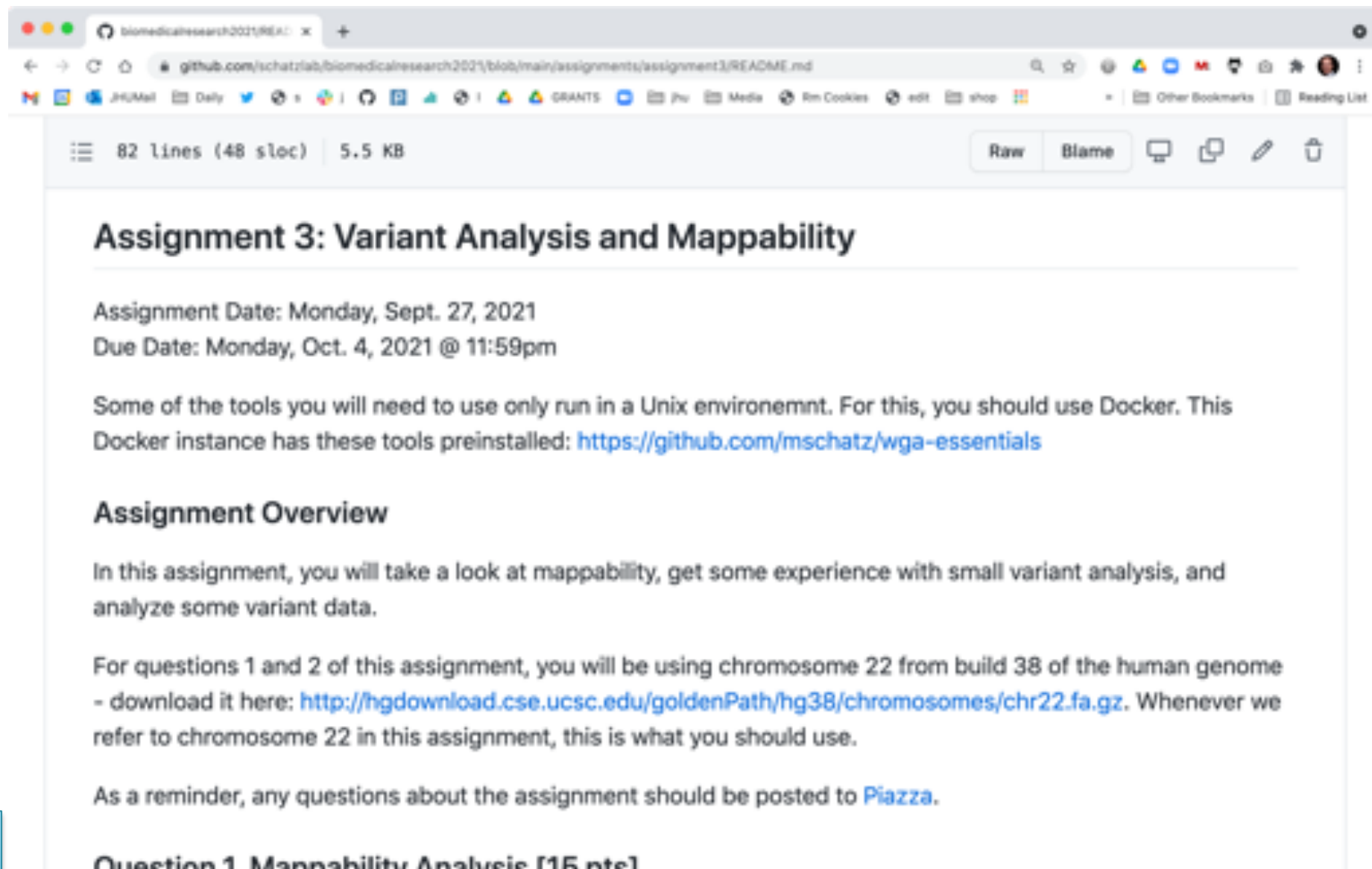
Sept 27, 2021

Lecture 8: Computational Biomedical Research



Assignment 3: Variant Analysis & Mappability

Due Oct 4 @ 11:59pm



The screenshot shows a web browser displaying a GitHub README file. The browser's address bar shows the URL: github.com/schatzlab/biomedicalresearch2021/blob/main/assignments/assignment3/README.md. The page header indicates the file is 82 lines (48 sloc) and 5.5 KB. The main content of the README is as follows:

Assignment 3: Variant Analysis and Mappability

Assignment Date: Monday, Sept. 27, 2021
Due Date: Monday, Oct. 4, 2021 @ 11:59pm

Some of the tools you will need to use only run in a Unix environment. For this, you should use Docker. This Docker instance has these tools preinstalled: <https://github.com/mschatz/wga-essentials>

Assignment Overview

In this assignment, you will take a look at mappability, get some experience with small variant analysis, and analyze some variant data.

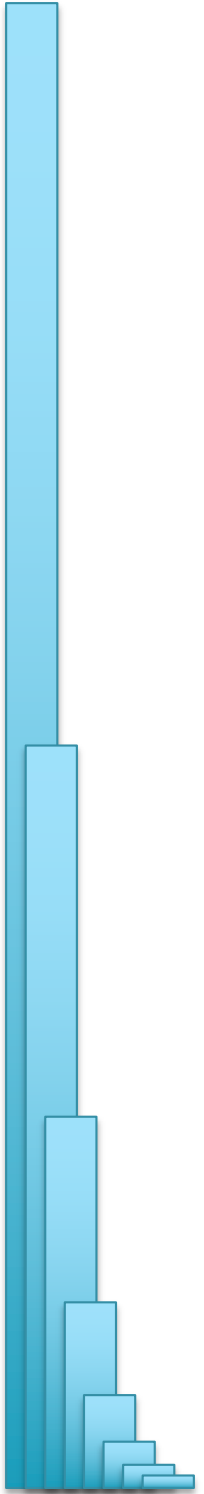
For questions 1 and 2 of this assignment, you will be using chromosome 22 from build 38 of the human genome - download it here: <http://hgdownload.cse.ucsc.edu/goldenPath/hg38/chromosomes/chr22.fa.gz>. Whenever we refer to chromosome 22 in this assignment, this is what you should use.

As a reminder, any questions about the assignment should be posted to [Piazza](#).

Question 1: Mappability Analysis [15 pts]

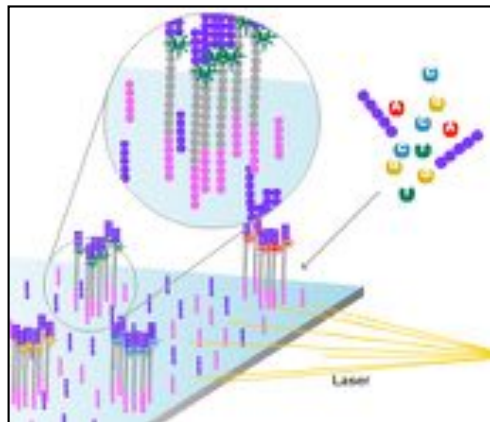
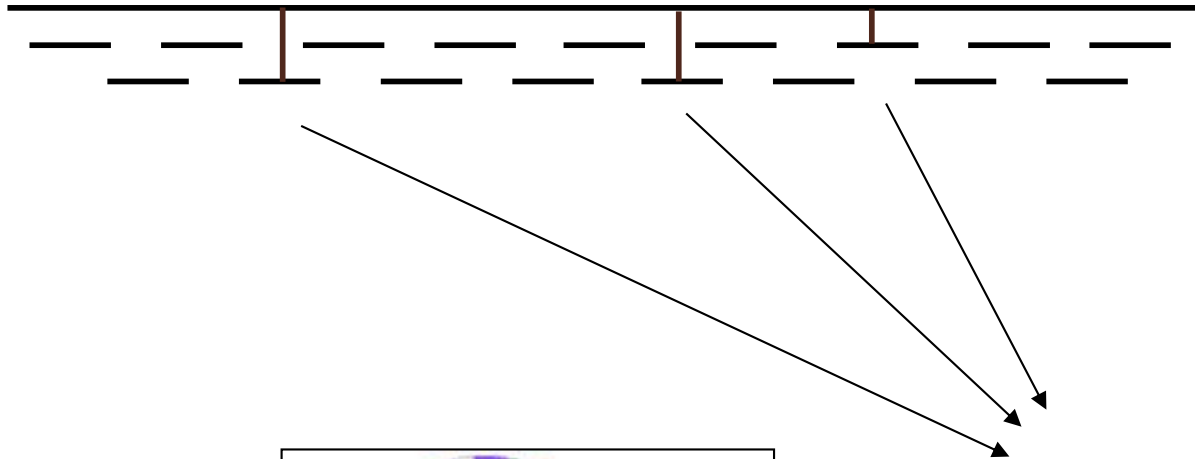
<https://github.com/schatzlab/biomedicalresearch2021>

Read Mapping



Personal Genomics

How does your genome compare to the reference?



Heart Disease

Cancer

Presidential smile

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?]

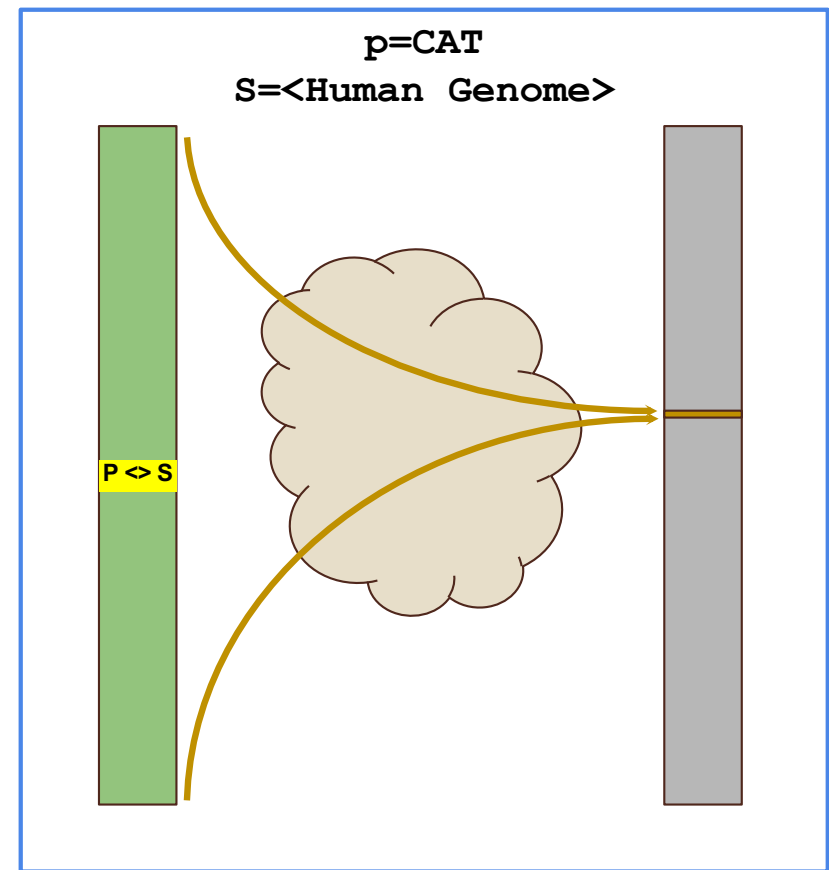
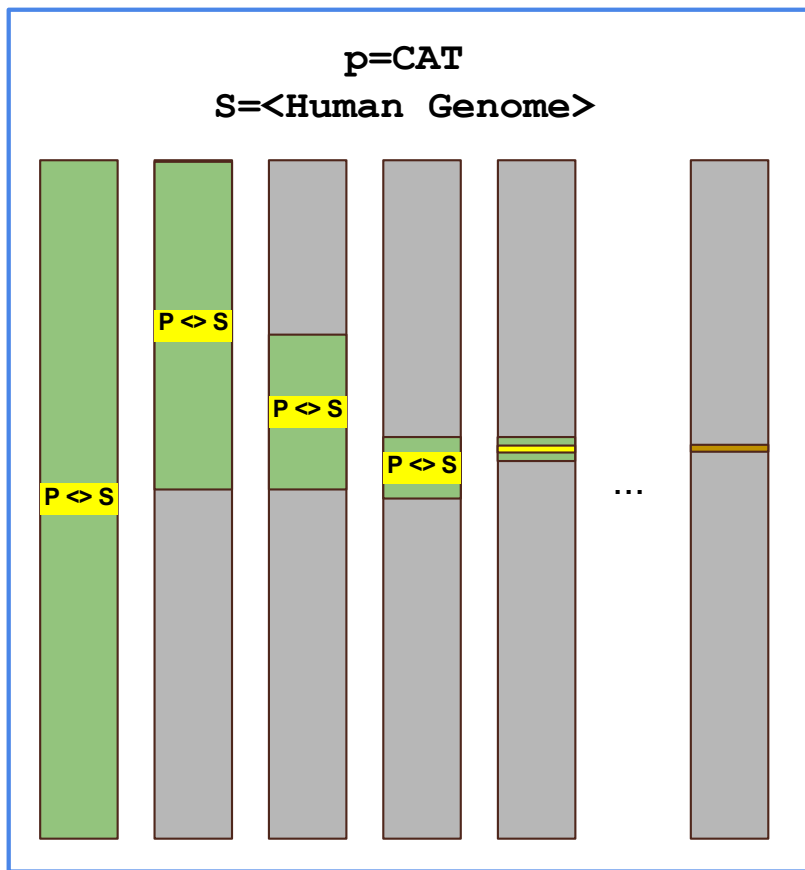
Searching the Index

- 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
→

Sapling: Accelerating Suffix Array Queries with Learned Data Models



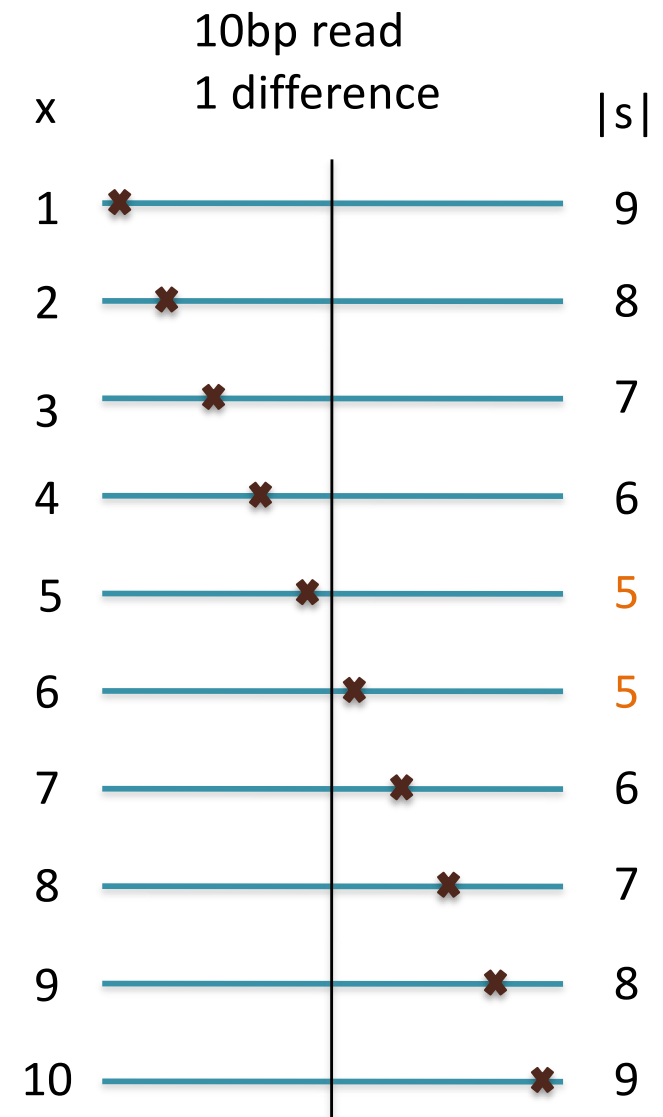
What if instead of a slow algorithmic approach to find the correct rows, we could somehow quickly guess/predict the correct rows?

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

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

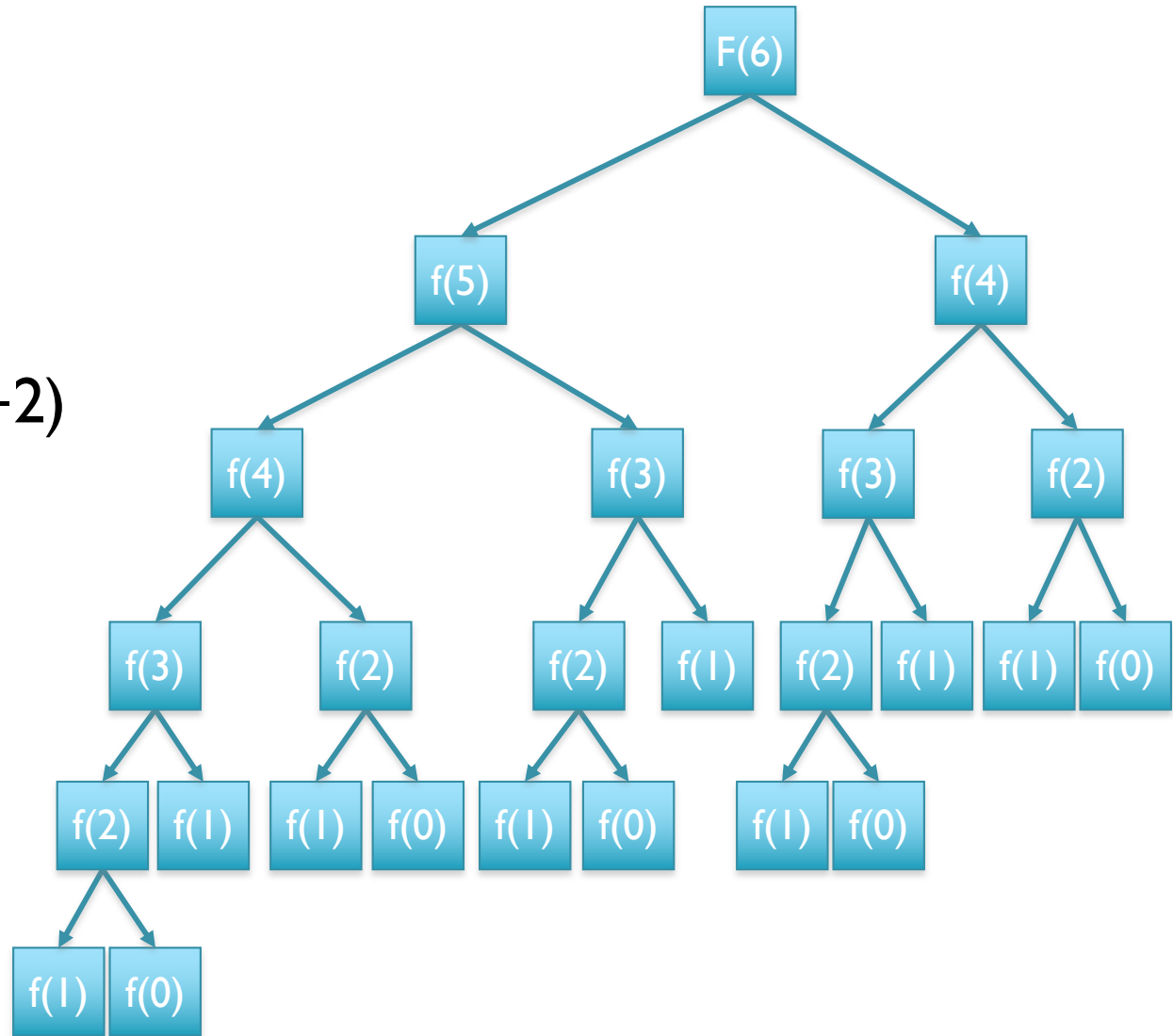
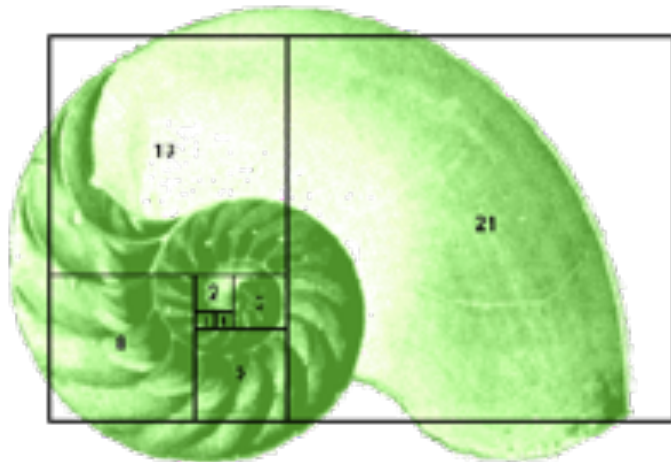




Part 2: Edit Distance (and Dynamic Programming)

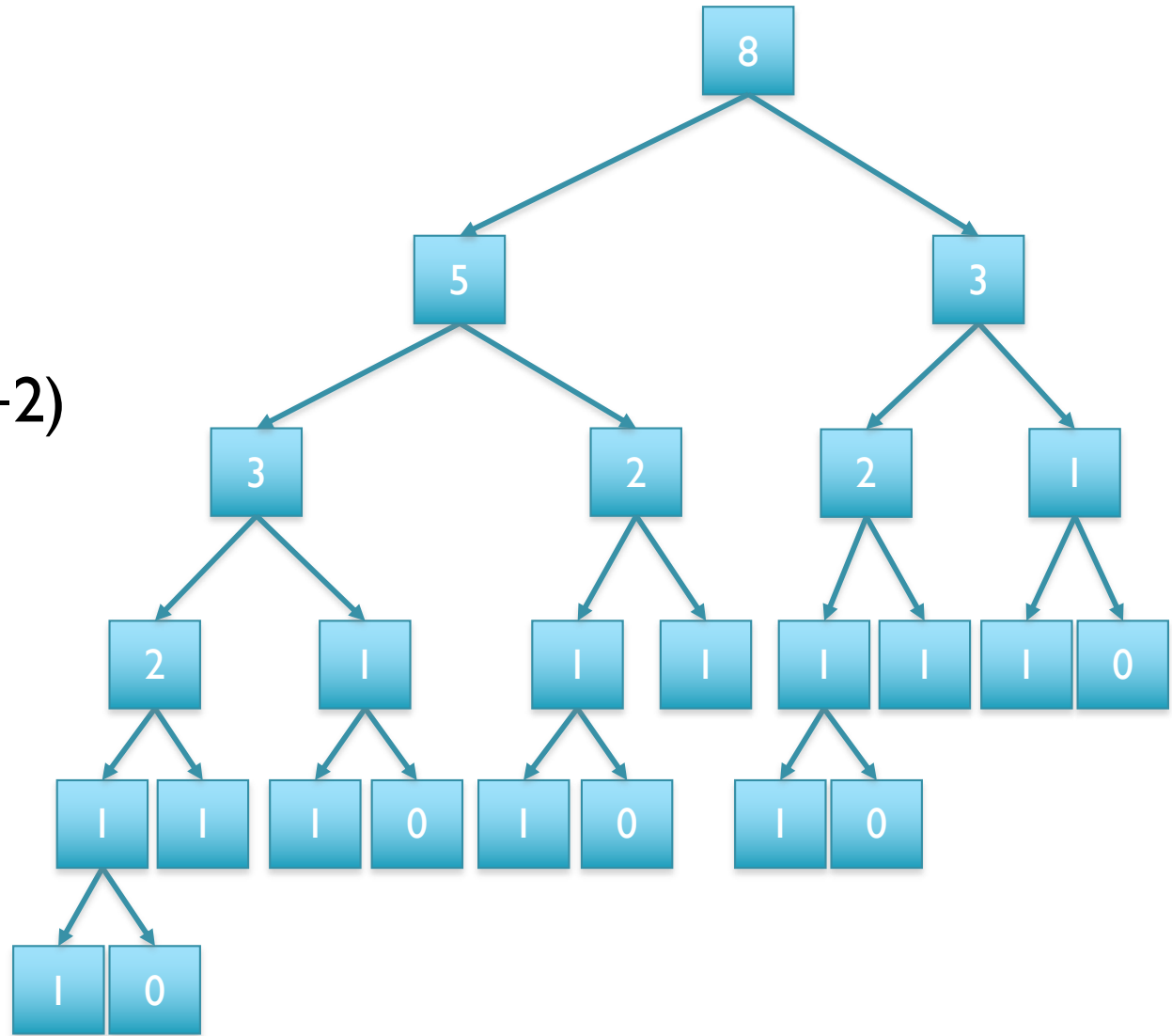
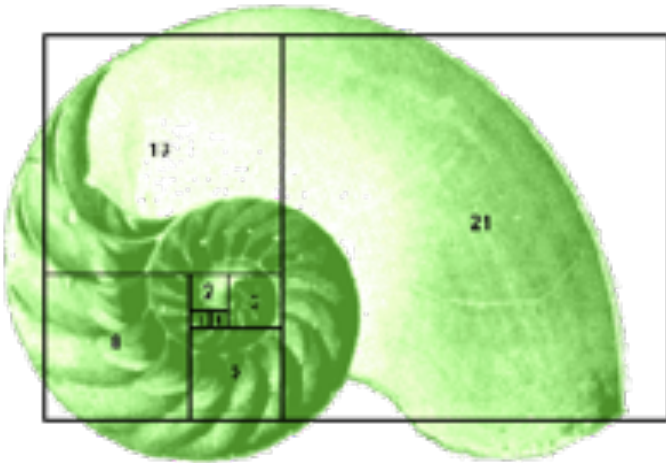
Fibonacci Sequence

```
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)
```



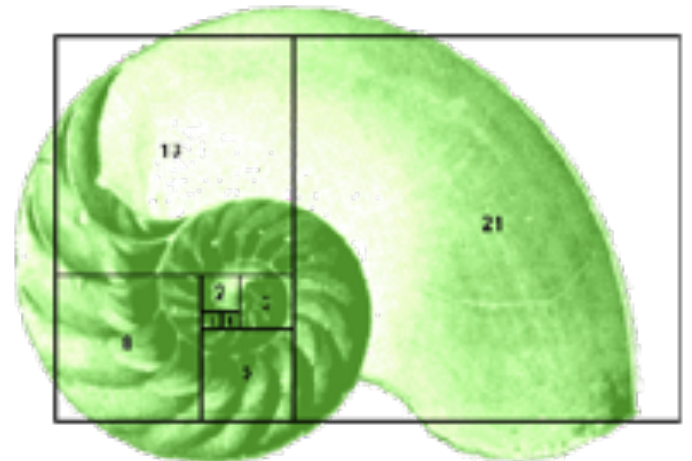
[What is the running time?]

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]
```

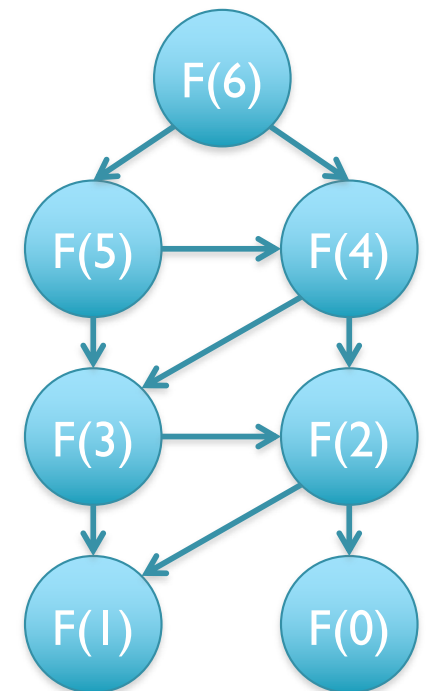
0	1	2	3	4	5	6
0	1	1	2	3	5	8

[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 (Bonus Slides!)
 - 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

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

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...

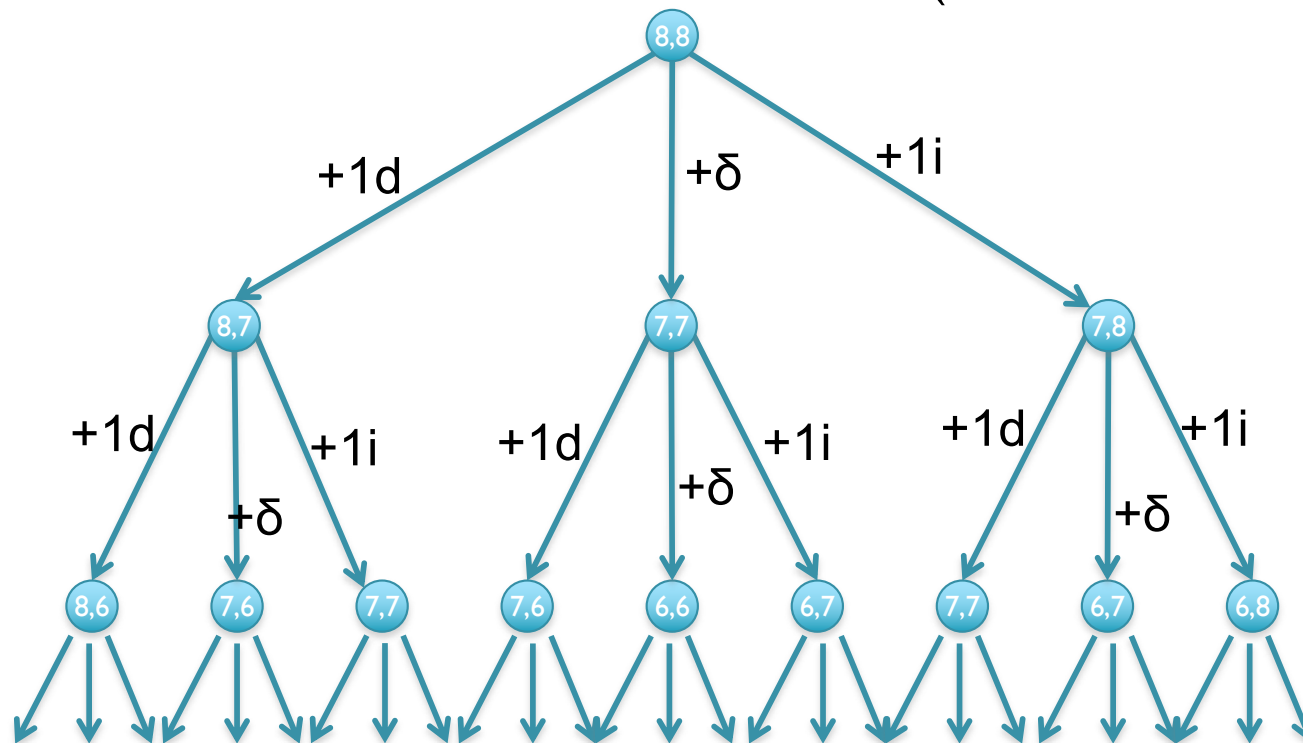
...M...	...I...	...D...
...X...	...-...	...X...
...Y...	...Y...	...-...

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 $i' \leq i$ 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?]