CSC 334: Topics in Computational Biology

Midterm: Fall 2015

Due: Tuesday, November 3 at 1pm

- This is a take-home exam with unlimited time from when it is out to when it is due.
- It is open-notes, so you may use any course materials. If you use any online resources that haven't been part of this class, please cite them.
- No communication about the exam with anyone in the class (or outside the class). However, you can email me if you need clarification.
- If there is a clarification I think should be made to the entire class, I'll post it on Piazza.
- I will still have office hours on Monday 4-6pm and Wednesday 5:30-7pm, but I might not say much!
- Turn in your exam to me in person or put it under the door of my office if I am not there (not in the bin outside).
- If you are unable to make it to my office to turn in the exam before the deadline, email me to make other arrangements such as scanning your exam.
- If you are unable to make progress on any part of the exam, tell me what you tried; describe your thought process.

Name	Solution Set (sketches)

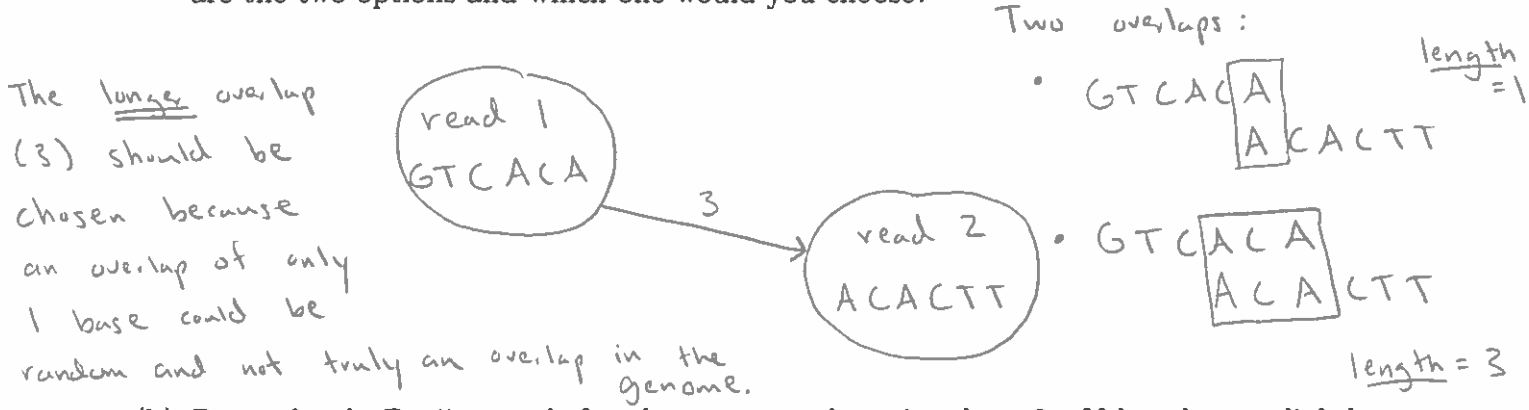
Part 1	/10	✓
Part 2	/25	✓
Part 3	/10	✓
Part 4	/20	✓
Part 5	/10	✓
Part 6	/25	✓
Total	/100	

Part 1: Genome Assembly

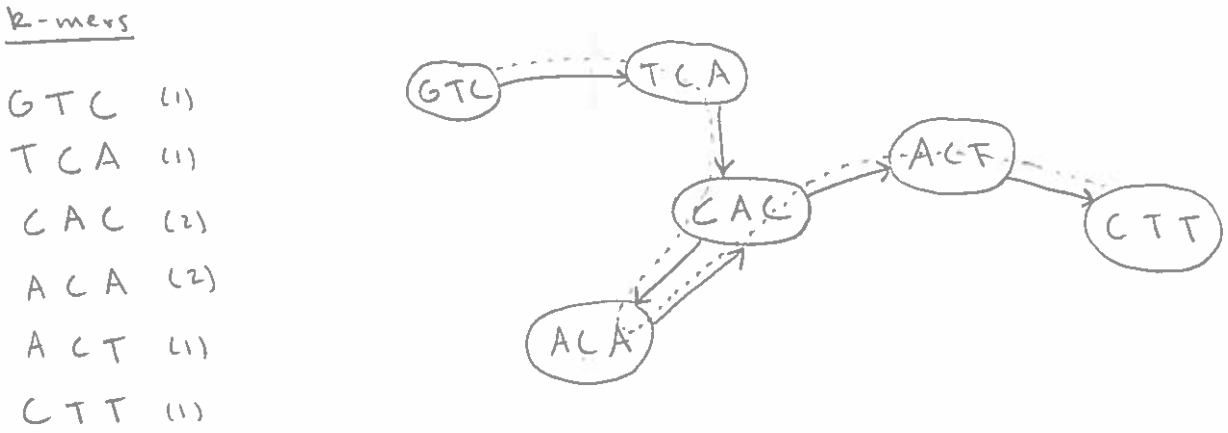
For this part, you are given two reads and tasked with assembling them into a genome (10pts):

read 1: GTCACA
read 2: ACACTT

(a) Draw the traditional read overlap graph for these two reads. For the overlap length, what are the two options and which one would you choose?



(b) Draw the de Brujin graph for these two reads, using $k = 3$. Velvet has a slightly more compact way of drawing the graph, but here use k-mers as nodes and directed edges for overlaps of $k - 1$ that occur in the read. (You don't need to do the reverse compliment.)



(c) Trace out the path through the de Brujin graph and write out the corresponding "genome".

dotted line above produces:

GTCACACTT

Part 2: String Alignment

In class we learned an algorithm for *global* string alignment (Needleman-Wunsch), which returns the optimal alignment between two strings over their entire lengths. There are many variations of Needleman-Wunsch for different tasks. One variation is used for *local* alignment (Smith-Waterman), which returns substrings of the original strings that are very similar. For example, if we wanted to find very similar (highly conserved) gene sequences shared between two very different (highly diverged) species, we might use local alignment. (25pts)

Let our two strings be S and T, with length$(S) = n$ and length$(T) = m$. Let match(a, b) be the match score between characters a and b, and let g be the gap penalty. The modifications below turn a global recursion into a local one:

Initialization:

$$\text{score}(i, 0) = 0, \quad \text{for} \quad i = 0, 1, \cdots, n$$
$$\text{score}(0, j) = 0, \quad \text{for} \quad j = 0, 1, \cdots, m$$

Recursion:

$$\text{score}(i, j) = \max \begin{cases} 0 \\ \text{score}(i-1, j-1) + \text{match}(S[i], T[j]) \\ \text{score}(i-1, j) + g \\ \text{score}(i, j-1) + g \end{cases}$$

Termination:

The highest score(s) in the DP table represent the end points of the best local alignment(s). Backtrace from these cells as before until a zero is reached to find the local alignments.

(a) Explain the relationship between these modifications and and the goal of *local* alignment.

These modification disallow negative scores (i.e. bad alignment scores), by allowing the alignments to "restart" whenever a 0 is encountered.

→ • Initializing the first row and column to 0 allows the alignment to start anywhere along either string.

→ • Adding 0 to the max operator allows the alignment to be restarted at any position within the table.

→ • Backtracing from the highest scoring cells allows the alignment to [illegible] position & returns the best one(s)

(b) For the overlap graph in Part 1, it might be helpful to use string alignment to compute the overlaps. To start off this idea, fill in the DP table below for the two reads from Part 1, using the *local* alignment algorithm above. For the cell $(i, j) = (4, 4)$, show your work clearly using the recursion above.

i ↓ \ j →	0	1	2	3	4	5	6
	-	A	C	A	C	T	T
0 -	0	0	0	0	0	0	0
1 G	0	0	0	0	0	0	0
2 T	0	0	0	0	0	1	1
3 C	0	0	1	0	1	0	0
4 A	0	1	0	2	1 ★	0	0
5 C	0	0	2	1	3	2	1
6 A	0	1	1	3	2	2	1

★ : $score(4,4) = \max \begin{cases} 0 \\ score(3,3) + match(A,C) \\ score(3,4) + g \\ score(4,3) + g \end{cases}$

$= \max\{0,\ 0-1,\ 1-1,\ 2-1\} = \boxed{1}$ ✓

(c) What is the best local alignment score? Traceback to find the best local alignment(s) that this score corresponds to. Was this procedure useful for finding read overlaps?

best local alignment score = $\boxed{3}$ (shaded cells)

tracebacks produce: (bold arrows)

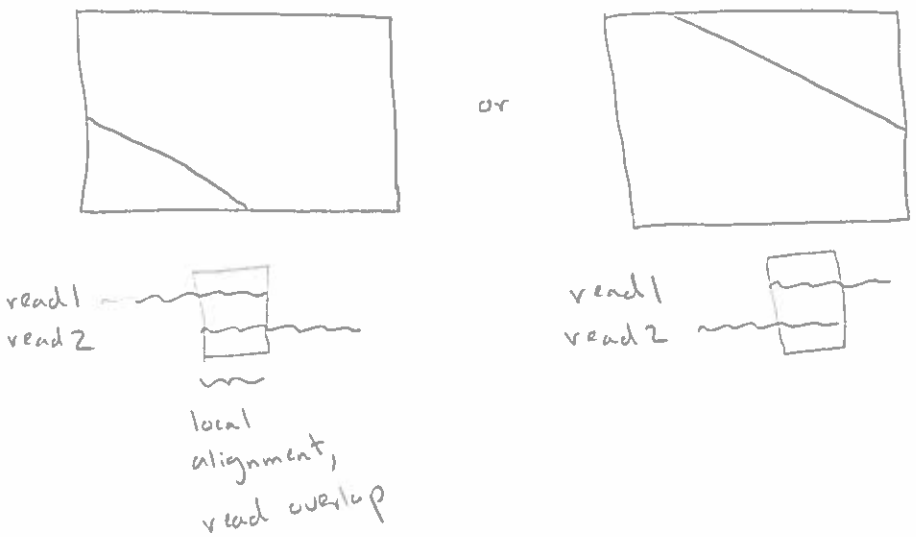
- cell at (6,3) => $\boxed{\begin{matrix} A\ C\ A \\ A\ C\ A \end{matrix}}$ ← only this local alignment was a read overlap.
- cell at (5,4) => $\boxed{\begin{matrix} C\ A\ C \\ C\ A\ C \end{matrix}}$

So the algorithm was useful, but it returned other alignments too.

(d) Make a modification to the termination procedure that would ensure that only read overlap alignments are returned.

- To return only read overlap alignments, the max score must lie along the last row or column, and the backtrace must terminate at the first row (if max score occurred on the last column) or first column (if max score was on the last row).

- Graphically:



Part 3: Burrows Wheeler Transform

Implement the BWT in python (your own original code), using the skeleton `midterm.py` online. Submit your code on Moodle. (10pts)

(a) Implement the helper function `cyclic_perms(S)`, which should return a list of all cyclic permutations of S, i.e. $\pi(S)$.

(b) Sort the cyclic permutations to obtain π^{sorted} (you can use the built-in python function `sorted(lst)`).

(c) Return the BWT (last column). Use it to obtain the BWT for read 1: GTCACA.

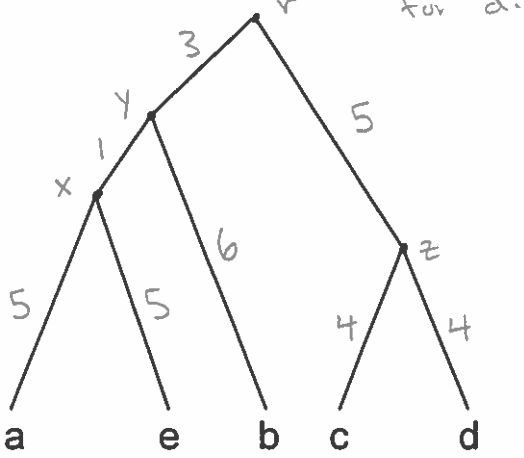
(d) Bonus (+2): Implement `reverse_bwt(L)` that takes the BWT as input and returns the original string S (it should not use π^{sorted}).

Part 4: Tree Building

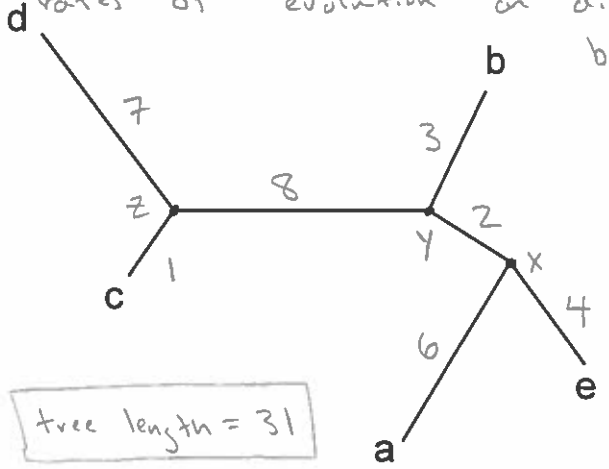
Comparing the trees and distance matrices created by UPGMA and NJ. (20pts)

The following two trees and *induced* distance matrices were created by UPGMA and NJ. Use these distance matrix results and tree topologies to label *all* the branch lengths of both trees. Without knowing the original distance matrix, which induced distance matrix do you think is "closer" to the truth?

NJ: total tree length is smaller (minimize evolution necessary for explaining the data) & allows for different rates of evolution on different branches.



tree length = 33



tree length = 31

d_u

UPGMA	a	b	c	d	e
a	0	12	18	18	10
b		0	18	18	12
c			0	8	18
d				0	18
e					0

UPGMA

$d_u(a,e) = 10 \Rightarrow d_u(a,x) = d_u(e,x) = 5$

$d_u(c,d) = 8 \Rightarrow d_u(c,z) = d_u(d,z) = 4$

$d_u(a,b) = 12 \Rightarrow d_u(b,y) = 6$

$\Rightarrow d_u(x,y) = 1$

$d_u(a,d) = 18 \Rightarrow d_u(d,r) = 9$

$\Rightarrow d_u(z,r) = 5$

$\Rightarrow d_u(y,r) = 3$

d_n

NJ	a	b	c	d	e
a	0	11	17	23	10
b		0	12	18	9
c			0	8	15
d				0	21
e					0

NJ

First use a to look at the subtree with c & d.

$d_n(a,d) - d_n(a,c)$

$= d_n(d,z) - d_n(c,z)$

$\Rightarrow 23 - 17 = d_n(d,z) - d_n(c,z)$

and also: (+) $8 = d_n(d,z) + d_n(c,z)$

$14 = 2 \cdot d_n(d,z)$

$\Rightarrow \boxed{d_n(d,z) = 7}$

$\Rightarrow \boxed{d_n(c,z) = 1}$

Similarly, use b to look at the subtree with a & e:

$$d_n(a,b) - d_n(e,b) = d_n(a,x) - d_n(e,x)$$

$$11 - 9 = d_n(a,x) - d_n(e,x)$$

and also: (+) $10 = d_n(a,x) + d_n(e,x)$

$$12 = 2 \cdot d_n(a,x)$$

$$\Rightarrow \boxed{d_n(a,x) = 6}$$

$$\Rightarrow \boxed{d_n(e,x) = 4}$$

Use c to look at the subtree with b & x:

$$d_n(c,b) - d_n(c,x) = d_n(b,y) - d_n(x,y)$$

$$12 - (15-4) = d_n(b,y) - d_n(x,y)$$

and also: (+) $(9-4) = d_n(b,y) + d_n(x,y)$

$$12 - 11 + 5 = 2 \cdot d_n(b,y)$$

$$\Rightarrow \boxed{d_n(b,y) = 3}$$

$$\Rightarrow \boxed{d_n(x,y) = 2}$$

Finally, use any pair that crosses (y,z) to find its length:

$$d_n(y,z) = d_n(d,b) - d_n(d,z) - d(y,b)$$

$$= 18 - 7 - 3$$

$$\Rightarrow \boxed{d_n(y,z) = 8}$$

Part 5: Perfect Phylogeny

Use Gusfield's algorithm to reconstruct a phylogeny for the data below (6 sequences, 4 sites). Show your steps, but you don't need to use radix sort. Based on your results, does a perfect phylogeny exist for this dataset? Explain why or why not. (10pts)

sequence	site 1	site 2	site 3	site 4
a	0	1	1	1
b	0	0	0	0
c	1	0	1	0
d	0	1	0	0
e	1	0	1	0
f	0	0	1	1

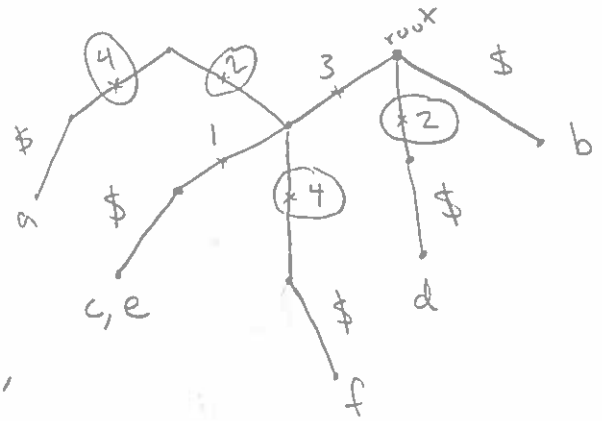
① sort the columns (descending):

	3	2	4	1
a	1	1	1	0
b	0	0	0	0
c	1	0	0	1
d	0	1	0	0
e	1	0	0	1
f	1	0	1	0

② write out mutation paths:

	path			
a	3	2	4	$
b	$			
c	3	1	$	
d	2	$		
e	3	1	$	
f	3	4	$	

③ build the phylogeny:

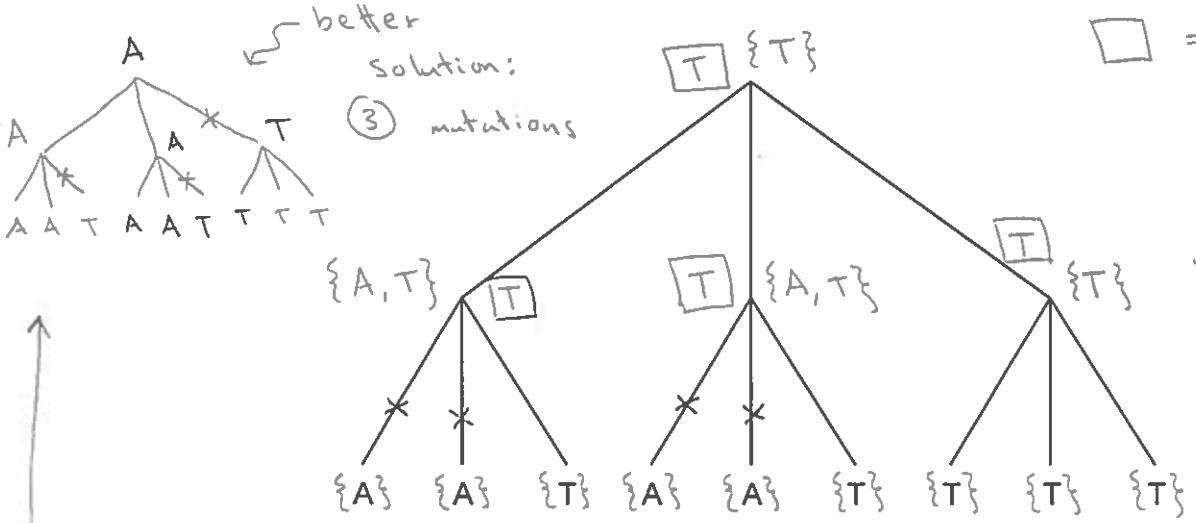


NOT a perfect phylogeny, two mutation events at site 2, & site 4.

Part 6: Ancestral Reconstruction

Fitch's algorithm for non-binary trees. (25pts)

(a) Run Fitch's algorithm on the ternary tree below to determine the base at every ancestral node. Given the result, how many mutations would be required to produce the data observed at the leaves? Is there a better solution? What is it and how many mutations does it involve?



with this solution, (4) mutations would be required to produce the data at the leaves. "A" would be a better choice for the root, giving us (3) mutations.

(b) Create a new algorithm (with the same goal as Fitch's algorithm) that works for general non-binary trees. Explain the steps clearly, using a notation similar to Fitch in class (see notes for class meeting 15).

Instead of using the union & intersection operators to determine the set of possible bases S_v at node v, use the base(s) with the maximum count over all the children.

indicator function $= \begin{cases} 1 & \text{if } b \in S_c \\ 0 & \text{if } b \notin S_c \end{cases}$

$$S_v = \left\{ \underset{b \in \{A,C,G,T\}}{\operatorname{argmax}} \left(\sum_{\substack{c \text{ in} \\ \text{children} \\ \text{of } v}} \mathbb{1}(b \in S_c) \right) \right\}$$

argmax returns the argument associated

# of times base b is seen in all the children of v

$S_\ell = \{x\}$ if x is the base assigned to leaf ℓ

example:

$$S_v = \left\{ \text{argmax} \left\{ \begin{array}{ll} 1, & \text{count for A} \\ 3, & \text{count for C} \\ 2, & \text{count for G} \\ 0, & \text{count for T} \end{array} \right\} \right\}$$

$S_v = \{C\}$

top-down step: same as before.

(c) Run your algorithm on the tree below to determine the base of every ancestral node (note that Fitch would be ambiguous in this case).

6 mutations total