Homework 4

CS 181, Fall 2025

Out: Nov. 3
Due: Nov. 14, 11:59 PM

Please upload your solutions on Gradescope. You can use LaTeX or a word document to write up your answers, but we prefer you use LaTeX. You may scan hand-written work or images for parts of solutions **only if** they are extremely clean and legible. Please ensure that your name does not appear anywhere in your handin.

Problem 0: Readings

The following sections of the Barton textbook chapter, which can be found on the resources page of the course website under Chapter 3 Readings. They may be helpful for Problems 3 and 4.

(a) *For Distance Methods, Corrections Are Essential to Convert Measures of Similarity to Evolutionary Distances*, pgs. 27 - 29

(b) *UPGMA and Neighbor-Joining Methods*, pgs. 22 - 26

Problem 1: Building and using suffix trees

The CS 181 TAs are spending a day out on Sorin's farm! But before they leave at the end of day, Sorin wants to test what they have learned from working on the farm by giving them a special problem to solve. They must find all the suffixes of the word, "OINKOINO", and they're enlisting your help to do so!

(a) Construct the **expanded** suffix tree T_x for the word x = `OINKOINO`. Build the tree one suffix at a time, starting with suf_1. Show the first four partial trees in your construction as well as the complete suffix tree T_x.

(b) Show how to use your suffix tree to return the starting indices of all occurrences of the string `OIN` in the text x. List all of these indices. In general, what is the big-O time complexity of returning the starting indices of all occurrences of a pattern p in a text t? Explain.

(c) Show how to use your suffix tree to determine whether the string `OINKOS` occurs in x. Does it occur in x? In general, what is the big-O time complexity of determining whether a pattern p occurs in a text t? Explain.

(d) **Bonus:** Construct the **compacted** suffix tree and the **position** suffix tree for the word x = `OINKOINO`. Explain why **position** suffix trees can be stored in $O(n)$ space, whereas **compact** suffix trees require $O(n^2)$ space.

Hints for this section: (1) You can assume that each node stores their children in a dictionary, so checking for a given character in the direct children of a node is constant time. (2) When calculating Big-O time complexity, consider what strings would create the worst-case scenario tree and how many nodes this tree would have.

Problem 2: Longest shared substring

Consider the following problem:

LONGEST SHARED SUBSTRING PROBLEM	
Input:	Strings u and v.
Output:	The longest substring that occurs in both u and v.

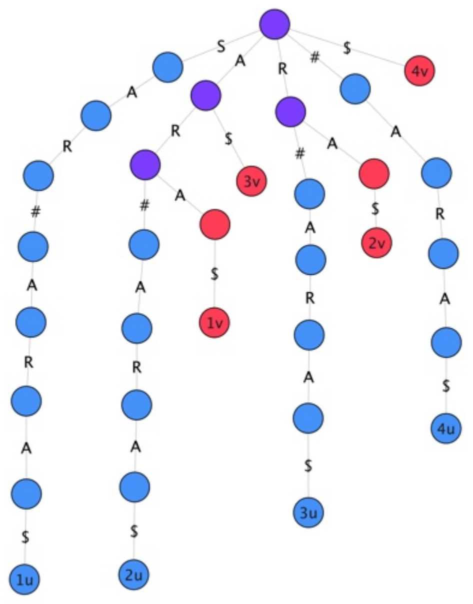
In theory, we could solve this problem by concatenating the two strings and constructing a joint suffix tree for the resulting string. In this problem, we expand upon this idea.

We begin by appending two different end-of-string characters to u and v, giving us $u\#$ and $v\$$. We then build the suffix tree for the concatenation of these two strings, $u\#v\$$. We shall color the leaves of this tree according to the following rule: if a leaf is labeled by the starting position of a suffix starting in u, color it blue; if a leaf is labeled by the starting position of a suffix starting in v, color it red. Note that all the leaves in the tree will be colored either red or blue according to this procedure.

Next we color the internal nodes of the tree according to the following rules:

- A node is colored blue (resp. red) if all the leaves in the subtree rooted at that node are blue (resp. red).
- A node is colored purple if the subtree rooted at that node contains both blue and red leaves.

The following is an example of joint suffix tree for $u\#v\$$ where u = SAR and v = ARA:



For <u>all</u> tasks in this problem, assume that we are using **expanded** suffix trees.

(a) Explain how every path ending in a purple node in the suffix tree of $u\#v\$$ spells out a substring shared by u and v.

(b) Explain how every path ending in a blue (resp. red) node in the suffix tree of $u\#v\$$ spells out a substring that appears in u but not in v (resp. v but not in u). **Note:** If the path contains #, then the substring spelled out by this path ends at #.

Given these two facts, it should be clear how we find the longest shared substring of u and v: We need only examine the strings spelled by paths that lead to purple nodes; the longest such string is precisely the solution to the Longest Shared Substring Problem.

The above approach makes use of a single suffix tree to solve the Longest Shared Substring Problem. It is also possible to solve this problem using two suffix trees, one for each of u and v.

(c) Describe an algorithm that solves the Longest Shared Substring Problem using the two suffix trees T_u and T_v.

(d) Compare (in as much detail as possible) the worst case space- and time-efficiency of the algorithm that uses a single suffix tree for $u\#v\$$ with the worst case space- and time-efficiency of the algorithm you designed in (c) above.

Problem 3: Jukes-Cantor Distance

(a) To get the feel of how to calculate the Jukes-Cantor distance, consider two biological sequences that have accumulated mutations over time. Below are snapshots of the two sequences at 4 different points in evolutionary time:

	$t = 1$	$t = 2$	$t = 3$	$t = 4$
Sequence 1	`AGGTCA`	`AGCTCA`	`AGCTCA`	`AACTCA`
Sequence 2	`AGGTCA`	`AGGTCA`	`TGGTCA`	`TGGTCC`

For each timepoint t, calculate the fraction of sites that are different between the two sequences, λ_t. Then use this value to calculate the Jukes-Cantor distance D_t between each sequence.

(b) Now consider a fifth timepoint:

	$t = 5$
Sequence 1	`AACTGA`
Sequence 2	`TGGTCC`

Compute λ as above. What happens when you try to use this value to calculate the Jukes-Cantor distance?

(c) Propose a correction to the Jukes-Cantor distance that avoids the issue above. What result do you get when you calculate your new distance at $t = 5$? Explain in two or three sentences the potential benefits and drawbacks of your new distance. It may be helpful to read up a little on the *Infinite Sites Model*.

Problem 4: Neighbor Joining

It's a special day at Sorin's farm! A baby cockatiel, Coco, recently hatched from its egg and is learning to fly. Despite being a hatchling, Coco is already flapping her tiny wings and visiting all of the animals on the farm. Inspired by the barn owl and the chickens she has seen so far, Coco wants to understand the evolution of the farm's birds over the decades and perhaps discover a distant cousin or two. Coco decides to build an evolutionary tree for the various birds and their relatives.

	Justinoavis	Alliavis	Sorinoavis	Chicken	Coco
Justinoavis					
Alliavis	135				
Sorinoavis	159	24			
Chicken	213	78	54		
Coco	177	42	18	36	

Use the distance matrix above and the neighbor joining algorithm to construct an evolutionary tree for the birds above. In a sentence or two, comment on Coco's place in the evolution of birds.