CSE 3500 **Homework 4** Released on October 12, 2021
Due on October 21, 2021

Suffix Trees, Hashing

Please complete this problem set by October 21, 2021 at 11:59PM.

Problem 0 – Suffix Trees on multiple strings (25%)

A suffix tree of m strings $(S_1, S_2, \ldots, S_m)$ is a suffix tree built from inserting the strings in order: $S_1\$_1$, $S_2\$_2$, $\ldots$, $S_m\$_m$ where $\$_1, \ldots, \$_m$ are distinct terminating strings.

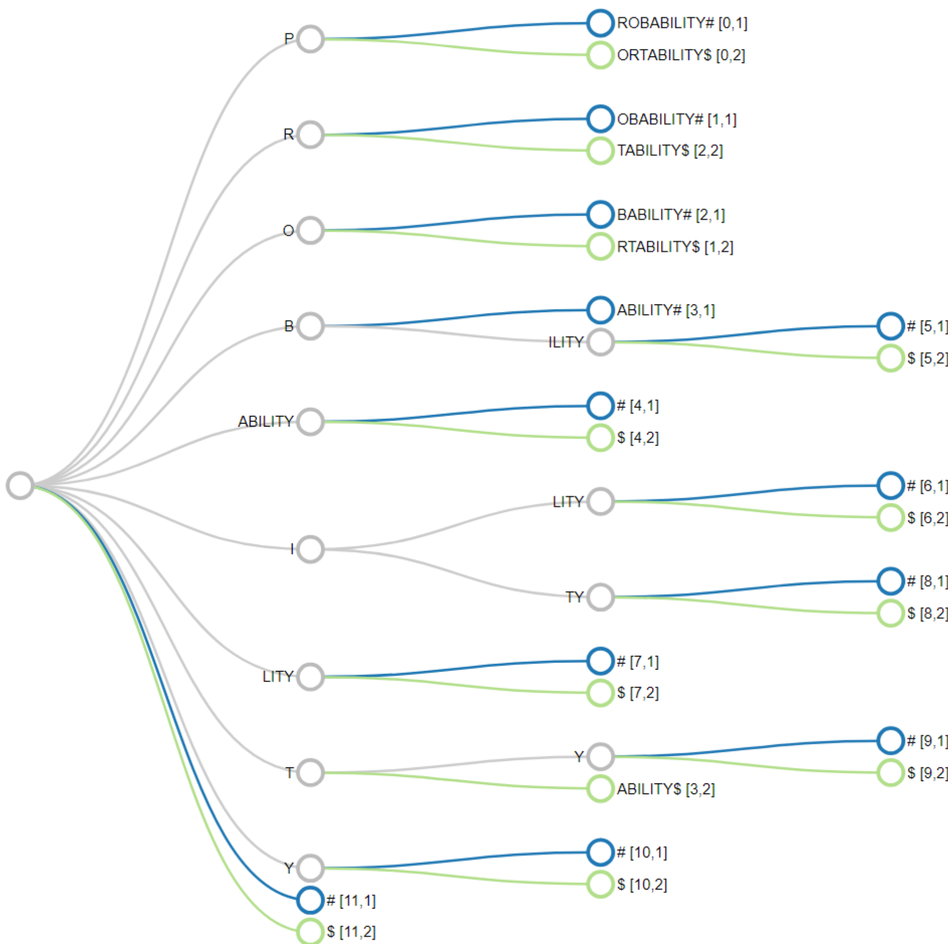

Figure 1: A suffix tree built for strings PROBABILITY# and PORTABILITY$ where # and $ are the terminating characters.

- Build the suffix tree for BANDANNA# and SAVANNAH$.

- Give an algorithm to find the longest common substring of m strings, $S_1, \ldots, S_m$ using suffix trees.
- What is your algorithms runtime? Show your work/give an argument.
- Using your algorithm (by tracing the steps in your suffix tree), what is the longest common substring of BANDANNA# and SAVANNAH$?

Problem 1 – Reversible substrings (25%)

Let S be a string of length m and a *substring* be a contiguous sequence of characters from S identified by a tuple $S_{(i,l)}$ for starting index i and length l. Let the reverse of a string $S[0]S[1]\ldots S[m-2]S[m-1]$ be $S[m-1]S[m-2]\ldots S[1]S[0]$ denoted S^-. A *reversible substring* is a substring where $S_{(i,l)} = S^-_{(m-(i+l),l)}$. Consider this example in python:

```
>>> S="PPABCDCBTTHMN"
>>> i=3
>>> l=5
>>> m=len(S)
>>>
>>> (S[::-1])[m-(i+l):m-(i+l)+l]
'BCDCB'
>>> S[i:(i+l)]
'BCDCB'
```

Note that in the previous example we convert the length l into an ending index for python. Intuitively, a reversible substring is a substring that is the same in the forward and reverse directions.

- Write an algorithm using suffix trees that computes the longest reversible substring of a string S. *Hint: Think about using a suffix tree for multiple strings.*
- What is your algorithms runtime? Show your work/give an argument.
- Use your algorithm to find the longest reversible substring of the string RACECARS (you need not draw a diagram).

Problem 2 – Suffix Arrays (25%)

- Build the suffix array for string "repetitive".
- Recall the longest common prefix array, an array of size $m-1$ that stores the length of the longest common prefix for each pair of adjacent strings. Build the LCP array for "repetitive".
- Describe an algorithm for finding all occurrences of a string P in a larger string S using suffix and LCP arrays built from S.
- Consider this algorithm and the case where the alphabet is large, say, $O(m)$. Would you prefer to use suffix trees or suffix arrays? Why?

Problem 3 – Hashing (25%)

Consider the following simple hash function h for mapping keys $z_j \in Z$ for $j = 1, \ldots, n$ to array indices $\{1, \ldots m\}$. For each key $z \in Z$, $h(z) = i$ with probability $1/m$ where $i = 1, \ldots, m$. What is the expected number of keys such that $h(a) = h(b)$ over all keys $a \neq b$? *Hint: you want to define a similar random variable representing collisions as we discussed in class or Chapter 1.5 of the Algorithms textbook.*