CMPT 409 Assignment 2

Heather Li, Ekjot Singh Billing, Manshant Singh Kohli

February 2, 2018

1 Hidden Password

(C++ 11) In this question, we first concatenate the string to itself and now we have a string of twice the length of the original string. We now create a suffix array for this new string by storing the indexes of the starting character of all sorted suffixes. The first element of this suffix array gives us the position of the first element which is equivalent to the number of rotations needed.

2 Where's Waldorf?

(C++ 11) For this question, we first create a suffix array for each row, column and diagnal from the grid. Then we binary search each word (and reverse word) in all the suffix arrays computed before and print the coordinates of first character.

3 Life Forms

(C++ 11) For this question, we first concatenate all the words together to form one long string while keeping track of the start position of each original string in this new long string. Then we create a suffix array for this string compute the LCP for each position. Then we use a TreeMap to sort all these suffix indexes based on their LCP values while confirming that the substring belongs to the same original string (ie., using TreeMap like a priority queue giving higher LCP values priority). Since we know that suffix array contains suffix positions in the sorted order, we know similar substring must be together in the suffix array. Now for each suffix (based on their LCP priority + sorted order), we check that its neighbours form enough substrings that are part of atleast half of original words. If this is true, we print the substring. We continue for all suffixes with the same LCP, if none of these have enough neighbours to be part of atleast half of original words, we move on to the next priority LCP value. If in the end we don't find any substrings that match from atleast half of original words, we print "?"

4 GATTACA

(C++ 11) For this question, we create a suffix array of the given string and compute the LCP for each position. Now we take the position with the highest LCP and binary search the lower and upper bound of this substring and this should give us the number of times this substring is repeated. If no LCP value is greater that 0, we print "No repetitions found!"