TA Lab Week 9 – Strings

The goal of this lab is to use smart data structures, to speed up computations on strings. The first exercise focuses on substrings, with the goal to determine the longest substrings without repeating characters. In Exercises 2 and 3, you are asked to implement a well-known measure of the similarity between two strings, the edit cost distance.

Exercise 1 – Longest Substring Without Repeating Characters

Write an algorithm that, given an input string, determines the length of the longest substring without repeating characters. For example, the longest substring without repeating characters of "ABCADFECAB" is "DFECAB" (length=6). The answer for string "BBB" is 1.

Hint

The length *n* of the input string can be really high (e.g., >100,000).
Can you achieve $O(n)$ by storing additional information?

Exercise 2 – Equal Cost Edit Distance

In computational linguistics and computer science, edit distance is a way of quantifying how dissimilar two strings (e.g., words) are to one another by counting the minimum number of operations required to transform one string into the other.

Edit distance finds applications in natural language processing, where automatic spelling correction can determine candidate corrections for a misspelled word by selecting words from a dictionary that have a low distance to the word in question. In bioinformatics, it can be used to quantify the similarity of DNA sequences, which can be viewed as strings of the letters A, C, G and T.

Given two strings str1 and str2 and the following basic operations that can performed on str1:

- Insert
- Remove
- Replace

find the minimum number of edits (operations) required to convert 'str1' into 'str2'. All of the above operations are of equal cost. Demonstrate that your algorithm works with a few test DNA sequences (combinations of A, C, G, and T).

Hint

This problem may be solved very simply with recursion, however that solution has a complexity which is $O(3^m)$*. Consider this approach:*

If last characters of two strings are same, ignore last characters and get count for remaining strings. Recur for lengths m-1 and n-1.

Else, we consider all operations on str1*, consider all three operations on last character of first string, recursively compute minimum cost for all three operations and take minimum of three values.*

Insert: Recur for m and n-1
Remove: Recur for m-1 and n
Replace: Recur for m-1 and n-1

We can solve this problem with dynamic programming for a solution which is $O(n \times m)$ *(in space and time), where n is* $|\mathrm{str1}|$ *and m is* $|\mathrm{str2}|$*.*

Exercise 3 – Varied Cost Edit Distance

When looking at DNA sequences, in some circumstances we want to get a measure of similarity for two different sequences, which may come from two different species or specimens.

Naively, we may say that a replacement operation (resulting perhaps, from a mutation) is of “lower cost” than an insertion or deletion when comparing a strand of DNA from two different species or subjects to compare their similarity.

Modify your previous implementation of the Edit Distance to account for this, such that:

- Insert; cost = 5
- Remove; cost = 5
- Replace; cost = 1

Demonstrate that your algorithm works with a few test DNA sequences (combinations of A, C, G, and T). Compare the edit distances of pairs of strings which have passed through the equal cost (exercise 2) and varied cost (exercise 3) algorithm and see how the different costs affect the scores and the operations made in the process.