# Advanced Data Structures

**CSE 4094**

**Project 1 Report**

**Ahmet Bozbay 150119861**

**Anıl Batuhan Aslan 150119656**

**Çağan Kurt 150119677**

## What is the definition of LCE(Longest extension Common)?

The LCE (Largest Common Extension) is a string matching algorithm used to find the longest common prefix of a set of strings. It is often used in applications such as text editors, where it can be used to quickly search for and highlight matching strings in a document.

The LCE algorithm works by comparing the characters of the strings in the set, starting from the beginning of the strings and working towards the end. If a character in one string does not match the corresponding character in the other strings, the algorithm stops and returns the common prefix up to that point.

For example, consider the following set of strings:

{"abcdef", "abcefg", "abcfgh"}

The LCE algorithm would start by comparing the first character of each string, which is "a" in all three cases. It would then move on to the second character, which is also "b" in all three strings. The algorithm would continue in this manner until it reaches the fourth character, at which point the strings diverge. The common prefix for these strings is therefore "abc", so the LCE algorithm would return "abc" as the result.

There are various ways to implement the LCE algorithm, and the specific details of the implementation will depend on the specific application and requirements. However, the basic principle of comparing characters in the strings and returning the common prefix remains the same.

## Comparison of the LCE and RMQ Structures

RMQ (Range Minimum Query) and LCE (Largest Common Extension) are two different algorithms that are used to solve different problems.

RMQ is an algorithm used to find the minimum value in a given range of a sequence of numbers. It can be implemented using a table, where the table stores the minimum values for different ranges of the sequence. For example, if the sequence is [4, 3, 7, 5, 2, 1], the table might store the minimum values for the following ranges:

| Range | Minimum Value |
|---|---|
| **[4]** | **4** |
| **[3,4]** | **3** |
| **[7]** | **7** |
| **[5,7]** | **5** |
| **[2,5]** | **2** |
| **[1,2]** | **1** |

To perform a RMQ query, the algorithm looks up the minimum value for the specified range in the table. For example, to find the minimum value in the range [2, 5], the algorithm would look up the value in the table and return 2.

LCE, on the other hand, is an algorithm used to find the longest common prefix of a set of strings. It can also be implemented using a table, where the table stores the length of the LCE for different pairs of strings. For example, consider the following set of strings:
{"abcdef", "abcefg", "abcfgh"}

The LCE table for these strings might look like this:

| Strings | LCE Lenghts |
|---|---|
| **("abcdef", "abcefg")** | **3** |
| **("abcdef", "abcdgh")** | **4** |
| **("abcefg", "abdfgh")** | **2** |

To find the LCE for a given pair of strings, the algorithm looks up the LCE length in the table and returns the common prefix of the specified length. For example, to find the LCE for the strings "abcdef" and "abcfgh", the algorithm would look up the value in the table and return "abc" as the result.

In summary, RMQ is an algorithm used to find the minimum value in a range of a sequence, while LCE is an algorithm used to find the longest common prefix of a set of strings. Both algorithms can be implemented using a table to store precomputed values, but they are used to solve different problems.

## Explaining of the Our RMQ Structure Algorithm How it works.

The function LCE2(A,B,i,j) is an implementation of the LCE (Largest Common Extension) algorithm that uses a suffix array and a longest common prefix (LCP) array to find the LCE of the substrings of A and B starting at index i and j respectively.

The function begins by checking if either index i or j is larger than the length of the respective string, in which case it returns 0. This is done to prevent index out of bounds errors.

The function then looks up the indices x and y of the substrings in the suffix array using the idx.index() function. The suffix array idx is a list of strings that store the indices of the suffixes in the original strings, with the suffixes sorted in lexicographic order. The indices are stored as strings in the format "<index>a" or "<index>b", where "a" or "b" indicates the string the suffix belongs to.

Once the indices x and y have been found, the function determines which index is smaller and returns the minimum value in the range of the LCP array that corresponds to the substrings. If x is smaller, it returns the minimum value in the range LCP[x:y]. If y is smaller, it returns the minimum value in the range LCP[y:x]. If x and y are equal, it returns the value at index x in the LCP array.

The LCE of the substrings is then given by the minimum value in the LCP array range, as the minimum value represents the length of the longest common prefix of the substrings.

I hope this helps clarify the purpose and operation of the code. Let me know if you have any further questions.

_

## Explaining of code :

The first function, LCE(A,B,i,j), is an implementation of the LCE algorithm that takes two strings A and B, and two indices i and j, and returns the length of the LCE of the substrings of A and B starting at index i and j respectively.

The algorithm works by comparing the characters of the two substrings, starting from the beginning and working towards the end, until it finds a character that is not the same in both substrings. It then returns the length of the common prefix up to that point.

The second function, lcp(a,b), is a helper function that finds the longest common prefix of two strings a and b. It does this by comparing the characters of the two strings, starting from the beginning and working towards the end, until it finds a character that is not the same in both strings. It then returns the length of the common prefix up to that point.

The third function, LCE2(A,B,i,j), is an implementation of the LCE algorithm that uses a suffix array and a longest common prefix (LCP) array to find the LCE of the substrings of A and B starting at index i and j respectively.

To find the LCE, the function first looks up the indices of the substrings in the suffix array and uses these indices to find the range of the LCP array that corresponds to the substrings. It then returns the minimum value in this range as the LCE.

Both implementations of the LCE algorithm are valid, but they have different trade-offs in terms of time and space complexity. The first implementation has a time complexity of O(n), where n is the length of the longer of the two substrings, but it requires O(1) space. The second implementation has a time complexity of O(log n) when using a suitable data structure to find the minimum value in the LCP array, but it requires O(n) space to store the suffix array and LCP array.