Tecnológico de Monterrey

Integrated Activity #1

Student

Diego Alberto Carrillo Castro - A01645484

Reflection:

In order to complete the Integrated Activity 1, I decided to select some algorithms to succeed in the task of finding malicious codes in the files of transmission, including sub-strings that may appear in the transmission files.
The Algorithms I selected for this task were: KMP, Manacher and LCSubString

KMP:

The KMP algorithm is ideal for text pattern searching, such as for malicious code. I implemented it to find the location of patterns m1, m2, and m3 within the transmissions t1 and t2.

- **Complexity:** The complexity of KMP is O(n+m), where n is the length of the text (transmission) and m is the length of the pattern (malicious code).

- **Analysis:** Unlike a brute-force search, which could be O(n×m) in the worst case, KMP is linear. This is achieved through a pre-processing phase that builds an auxiliary table (lpsTable). This table, which has a complexity of O(m), allows the algorithm to know how far to "shift" without re-examining already-compared characters in the text, ensuring a very efficient search.

Manacher:

I used Manacher's algorithm to find the longest palindromic substring in the transmission texts. This function not only identifies the longest substring but also counts the total number of palindromes, providing an additional metric for analyzing data structure.

- **Complexity:** Manacher's algorithm has a linear complexity, O(n), where n is the length of the input string.

- **Analysis:** While it may appear to have a nested loop, the algorithm is linear because it leverages palindrome symmetry. It doesn't re-evaluate the length of already-found palindromes. Instead, it uses information from previous palindromes to "jump" and avoid redundant work. To handle both even and odd-length palindromes uniformly, the string is pre-processed by inserting a special character (#) between each character.

LCSubString:

To find the longest common substring between the two transmissions (t1 and t2), I implemented a dynamic programming approach. This algorithm builds a two-dimensional table to store the length of common suffixes for each pair of substrings, which guarantees that the solution is optimal.

- **Complexity:** The complexity of this algorithm is O(n×m), where n and m are the lengths of the two strings.

- **Analysis:** Although it is quadratic, this complexity is expected since the algorithm must compare every character of the first string with every character of the second to build the dynamic programming table. This method is robust and ensures the precise identification of the longest common substring, which in this context could indicate repeated fragments of information or structure in both files.