Integrated Activity 1 reflection

Eugenio Guzmán Vargas

A01621084

This project involved the analysis of simulated data transmission files to identify security threats and patterns. The core task was divided into three distinct challenges, which consisted on detecting specific sequences of malicious code, locating the longest "mirrored" or palindromic code segment, and measuring the similarity between two transmission files by finding their longest common substring. To solve these problems efficiently, it was crucial to move beyond brute force or “naive” methods and apply specialized, high-performance string algorithms, to ensure that our program could handle edge cases in a reasonable time.

For each part of the problem, a specific algorithm was chosen to provide an optimal balance of performance and correctness:

The first task required searching for several small, fixed patterns within two much larger text sequences. A naive approach would involve checking for the pattern at every possible starting position in the text, leading to a worst-case complexity of O(n * m), where 'n' is the length of the text and 'm' is the length of the pattern. However, this is highly inefficient. Hence, the Knuth-Morris-Pratt (KMP) algorithm was selected to optimize this process. KMP's efficiency lies in its preprocessing step. It creates a "Longest Prefix Suffix" (lps) table for the pattern, which allows the algorithm to avoid redundant comparisons after a mismatch. If a mismatch occurs, the lps table tells us the length of the longest proper prefix of the pattern that is also a suffix. This information allows us to shift the pattern forward intelligently, without ever needing to move the text pointer backward.
<u>Computational Complexity:</u>

- Preprocessing (lps table): O(m)
- Searching: O(n)
- Total Complexity: O(n + m).

This linear time complexity represents a massive improvement over the naive approach, especially for long transmission files.

The second challenge was to find the longest mirrored code, or palindrome. The brute-force method of checking every possible substring would be $O(n^3)$, while a more optimized approach of expanding from every possible center would be $O(n^2)$. Both are too slow for large datasets.
Manacher's Algorithm was the ideal choice as it solves this problem in linear time. Its strategy involves two key innovations. It iterates through the transformed (added special characters to handle odd and even lengths) string, maintaining the center and rightmost boundary of the palindrome found so far that extends the furthest. By using the symmetrical properties of palindromes relative to this center, it can make an educated guess for a minimum palindrome length at new positions, drastically reducing the number of character comparisons needed.

Computational Complexity:

- The algorithm processes each character of the transformed string a constant number of times. Its total complexity is O(n), making it the most efficient solution possible for this problem.

Finally, to quantify the similarity between the two transmission files, we needed to find the longest substring they shared. For this task, a dynamic programming approach was implemented. This method constructs a 2D table of size (n+1) x (m+1), where 'n' and 'm' are the lengths of the two strings.
Each cell dp[i][j] in the table stores the length of the common substring ending at position i-1 in the first string and j-1 in the second. The table is filled based on a simple rule: if the characters string1[i-1] and string2[j-1] are the same, dp[i][j] = dp[i-1][j-1] + 1. If they are different, dp[i][j] = 0, as a contiguous substring has been broken. While filling the table, we keep track of the maximum value encountered, which corresponds to the length of the longest common substring.

Computational Complexity:

- Time Complexity: Since every cell in the (n x m) grid must be computed, the time complexity is O(n * m).
- Space Complexity: The 2D table requires space proportional to the product of the string lengths, resulting in a space complexity of O(n * m).

While this is less efficient than the linear-time algorithms for the other parts, it is a standard and reliable method for this problem. For the size of the test files, this specific approach was the most adequate that we could find.