Advanced Array and String Operations with Complexity Analysis

1. Introduction

This report provides an analysis of the time and space complexity of various C++ algorithms and data structures implemented for multi-dimensional arrays and string operations. The focus is on understanding their performance characteristics and practical implications.

2. Implemented Algorithms and Structures

2.1 Two-Dimensional Array Operations

File: TwoDimensionalArray.cpp

Overview: This implementation includes operations such as initialization, row/column insertion, and element access within a two-dimensional array.

Complexity Analysis:

- **Access Operation**:
  - **Time Complexity**: O(1)
  - **Space Complexity**: O(n * m), where *n* and *m* are the array dimensions.
- **Row/Column Insertion**:
  - **Time Complexity**: O(n) for inserting a row, O(m) for inserting a column.
  - **Space Complexity**: O(n * m), due to the array size remaining unchanged.

2.2 KMP String Matching Algorithm

File: StringAlgorithms.cpp

Overview: The Knuth-Morris-Pratt (KMP) algorithm is implemented for efficient pattern matching in strings. This algorithm preprocesses the pattern to create a longest proper prefix-suffix array to avoid redundant comparisons.

Complexity Analysis:

- **Preprocessing**:
  - **Time Complexity**: O(m), where *m* is the length of the pattern.
  - **Space Complexity**: O(m), for the prefix-suffix array.
- **Search Operation**:
  - **Time Complexity**: O(n), where *n* is the length of the text.
  - **Space Complexity**: O(1), apart from the prefix-suffix array.

2.3 Run Length Encoding (RLE)

File: RunLengthEncoding.cpp

Overview: Run Length Encoding is a simple form of data compression where consecutive occurrences of the same character are replaced with a single instance followed by the count.

Complexity Analysis:

- **Time Complexity**: O(n), where *n* is the length of the input string.
- **Space Complexity**: O(n), as the output size can be proportional to the input.

3. Test Cases

Each algorithm is accompanied by test cases located in the tests/ directory:

- **Two-Dimensional Array Operations** (test_TwoDimensionalArray.cpp): Tests for element access, row/column insertion, and boundary cases.
- **KMP Algorithm** (test_StringAlgorithms.cpp): Tests with varying text and pattern lengths, including edge cases like empty patterns.
- **Run Length Encoding** (test_RunLengthEncoding.cpp): Tests for strings with repetitive and non-repetitive sequences.

4. Summary

The implemented algorithms demonstrate efficient handling of their respective operations:

- The **KMP algorithm** is ideal for searching patterns in large texts, outperforming naive methods.
- **Two-dimensional array operations** provide a basis for advanced data structure implementations.
- **Run Length Encoding** showcases a simple yet effective compression technique for strings with repetitive characters.