String Algorithms: Introduction

- This chapter focuses on efficient algorithms for string processing.
- Many string problems can be solved in $O(n^2)$ time, but the challenge is to find algorithms that work in $O(n)$ or $O(n \log n)$ time.
- A fundamental string processing problem is the pattern matching problem: finding occurrences of a pattern in a string.
- We'll explore algorithms beyond the brute force approach, aiming for better time complexities.

String Terminology

- **Zerobased indexing:** Strings are indexed starting from 0 (e.g., s[0] is the first character).
- **Alphabet:**The set of characters that can appear in a string (e.g., {A, B, ..., Z}).
- **Substring:**A sequence of consecutive characters within a string (e.g., 'AB' is a substring of 'ABC').
- **Subsequence:**A sequence of characters in a string, not necessarily consecutive, but maintaining their original order (e.g., 'AC' is a subsequence of 'ABC').
- **Prefix:**A substring starting at the beginning of a string (e.g., 'AB' is a prefix of 'ABC').
- **Suffix:**A substring ending at the end of a string (e.g., 'BC' is a suffix of 'ABC').

Pattern Matching Problem

- **Goal:**Find all occurrences of a pattern (length m) within a string (length n).
- **Example:**Pattern 'ABC' occurs twice in the string 'ABABCBABC'.
- **Brute Force Approach:** Tests all possible positions for the pattern in the string, taking O(nm) time.
- **Challenge:**Find algorithms with better time complexities, ideally O(n + m).

Block Decomposition for Efficient Range Queries

- **Problem:**Given an array, efficiently calculate the sum of elements within a specified range.
- **Block Decomposition Approach:**
- Divide the array into blocks of size $\sqrt{n}$.
- Store the sum of elements in each block.
- To calculate the sum of a range, divide it into single elements and blocks, allowing for $O(\sqrt{n})$ time complexity.

Case Processing: Finding Closest Cells with Same Letter

- **Problem:**Given a grid of cells with letters, find two cells with the same letter that are closest to each other.
- **Approach:**
- Consider each letter separately.
- For each letter, find the minimum distance between two cells containing that letter.
- **Algorithms:**
- **Algorithm 1 (Brute Force):** Check all pairs of cells with the same letter, taking O(k^2) time (k is the number of cells with that letter).
- **Algorithm 2 (BreadthFirst Search):** Simultaneously start a breadthfirst search from each cell with the same letter, finding the minimum distance.