# CS293 Project
# Plagiarism Checker

**Team Gotcha!**
**Ramachandran S** - 23b1052
**Harith S** - 23b1085
**Adarsh J** - 23b0960

November, 2024

# PHASE 1

## Perfect Matchings

The function `findExactMatchLength()` calculates the total length of all perfect matches between two tokenized submission files, where each match meets a specified minimum length of 10.

To avoid duplicates, two boolean arrays, `matched1` and `matched2`, track elements already part of a match. The function iterates through all possible starting indices, finds the longest matching substrings, and adds their lengths to the total if they meet the threshold. Matched elements are marked, and the loop advances by the match length to avoid overlaps.

The function returns the total length of all perfect matches.

## Approximation Match

### 1. Levenshtein Distance Function

The `levenshteinDist` function calculates the minimum number of edits (insertions, deletions, or substitutions) required to transform `submission1` into `submission2`. It uses a dynamic programming matrix `dp`, where:

- `dp[i][j]` holds the edit distance for the first `i` elements of `submission1` and the first `j` elements of `submission2`.
- The recurrence relation is:

$$\texttt{dp[i][j]} = \min(\texttt{dp[i-1][j-1]} + \texttt{cost}, \texttt{dp[i][j-1]} + 1, \texttt{dp[i-1][j]} + 1),$$

  where `cost` is 1 if elements differ, otherwise 0.

### 2. Approximate Match Function

The `findApproxMatch` function efficiently identifies similar substrings in two vectors by extending the edit distance approach:

- The first row of `dp` is initialized to allow substring matching.
- Matches are identified if the normalized distance is below a threshold (e.g., 96% similarity) for substrings of at least 15 elements.

### 3. Longest Common Subvector

The `longestCommonSubvector` function calculates the length of the Longest Common Subsequence (LCS) between two vectors. Using dynamic programming:

- `common[i][j]` stores the LCS length for the first `i` and `j` elements.
- The recurrence relation is:

$$\texttt{common[i][j]} = \begin{cases} \texttt{common[i-1][j-1]} + 1 & \text{if elements match,} \\ \max(\texttt{common[i][j-1]}, \texttt{common[i-1][j]}) & \text{otherwise.} \end{cases}$$

# PHASE 2

## Bloom Filter

The core data structure we used for checking plagiarism is a Bloom filter. We used to insert code tokens into a bit array and check for the existence of a token in the array. The system uses two Bloom filters: one for 15-token sequences and another for 75-token sequences.

- **Hash Function**: A Rolling hash function that is used to map sequences of tokens to indices in the Bloom filter's bit array.
- **Insert and Check Operations**: Code tokens are inserted into the Bloom filter for future comparisons, and the system checks whether specific token sequences exist in the filter.

## Multithreading

- A single worker thread is used to process tasks asynchronously, while the add submissions proceeds to deal with the next submission, improving responsiveness.
- Tasks such as tokenization, plagiarism detection, and Bloom Filter updates are enqueued to the worker thread to make sure add submission doesn't take significant time to return.
- A thread-safe task queue is implemented using `std::queue`, `std::mutex`, and `std::condition_variable` to keep track of all pending processing of submissions.
- The worker thread is notified when tasks are added using `notify_one`.
- Both constructors (default and parameterized) utilize threading for efficient initialization.

## Plagiarism Detection

The system compares token sequences in both Bloom filters (15-token and 75-token sequences) to detect potential plagiarism. If matches are found:

- **Perfect Matches:** Flag of there is a exact match of length 75 or if there are atleast 10 exact matches of length 15 or more .(Direct Plagiarism)
- **Time-Based Flagging:** If the time difference between submissions is more than 1 second, only the later submission is flagged; otherwise, both are flagged.
- **Additional Matches:** Flag if there are 20 exact matches of 15-token sequences, even if they aren't continuous .(Patchwork Plagiarism)

## Memory Management

Once a submission is processed, Bloom filters that are no longer required are deleted to free memory, ensuring efficient memory usage during the operation.

# Workflow Summary

1. **Tokenization**: Code submissions are tokenized into sequences of integers.
2. **Bloom Filter Insertion**: Token sequences are inserted into Bloom filters for later comparison.
3. **Plagiarism Detection**: Token sequences in new submissions are compared against existing submissions using the Bloom filters.
4. **Asynchronous Processing**: Tasks are handled by a worker thread to ensure non-blocking execution.
5. **Flagging**: Submissions suspected of plagiarism are flagged for review.