

Plagiarism Checker

Mohana Evuri
23B1017

Shaik Awez Mehtab
23B1080

Abhineet Majety
23B0923

Fall 2024

1 Phase 1

1.1 Net Length of Exact Matches Calculation

- **Algorithm:**
 - For each matching length “len” in the range $\{10, 11, \dots, 20\}$, the number of matching substrings is counted and added.
 - The substrings of each length are compared efficiently by `reinterpret_cast`, using which a vector of integers is hashed to a sequence of bytes.
- **Function Signatures:**
 - `int numExactMatches(const std::vector<int>&, const std::vector<int>&)`: Computes the number of exact matches.

1.2 Longest Approximate Match

- **Algorithm:** For each pair of start indices of the two vectors, the length of the longest approximate match is found. This is done by simply traversing the vectors.
- **Function Signatures:**
 - `std::array<int, 3> findLongestApproxMatch(std::vector<int>&, std::vector<int>&)`: Finds the longest approximate match and the respective start indices.

1.3 Helper Functions

- `SubstrMap allSubstrHashes(const std::vector<int>&, int)`: Hashes all substrings of a given length.
- `std::string getHash(const std::vector<int>&, int, int)`: Hashes a vector of integers.
- `bool is_already_checked(const std::vector<bool>&, int, int)`: Checks if a substring was already matched.
- `bool isPlagged(const int, const int, const int, const int)`: Given the lengths of submissions, number of matches, and maximum approximate match length, it determines if plagiarism has taken place.

1.4 Overall code flow

- **Data Structures:** We have used `vector` and `unordered multimap` in our implementation.
- **Flow:**
 - When `match_submissions` is called, it calls `numExactMatches`, `longestApproxMatch` and `isPlagged` in that order
 - `numExactMatches` uses `isAlreadyChecked` and `allSubstrHashes`, which uses `getHash`.

1.5 Complexity Analysis

Let n and m be the submission sizes.

- **Time Complexity:** $O(nm \cdot \min(n, m))$: Calculating the number of exact matches takes $O(nm)$ time. Finding the longest approximate matching substring takes $O(nm \cdot \min(n, m))$ time.
- **Space Complexity:** $O(n + m)$: The calculation of the number of exact matches takes $O(n + m)$ space. Finding the longest approximate matching substring takes $O(1)$ space.

2 Phase 2

2.1 Plagiarism Detection

- **Algorithm:** Dynamic programming is being used. The length of the matching substring that ends in the indices is recorded for each pair of indices i, j of the vectors.
 - **Short Pattern Match Detection:** Whenever the length of the matching substring equals 15, the number of matches is incremented.
 - **Long Pattern Match Detection:** For each pair of indices, the length of the longest match is updated if needed.
 - **Patchwork Plagiarism Detection:** The number of matches with submissions made in the last second is recorded for each submission. Flagging is done based on this measure.
- **Function Signatures:**
 - `void check_two_submissions(std::pair<int, std::shared_ptr<submission_t>>, std::pair<int, std::shared_ptr<submission_t>>, int, int, int&):` Checks if two submissions are similar and flags them if necessary.
 - `void plagiarism_checker_t::process_submissions(int):` Compares the current submission with all previous submissions and flags in case of patchwork plagiarism.

2.2 Complexity Analysis

Let the number of files be m and the average number of tokens per file be n .

- **Time Complexity:** $O(mn^2)$: Comparing two submissions takes $O(n^2)$ time, and there are $O(m)$ files to compare for each submission.
- **Space Complexity:** $O(mn)$: A cache of tokens of all previous submissions is maintained, which takes $O(mn)$ space. Vectors and queues take $O(m)$ space.

2.3 Concurrency Features

- **Threading:** Threading is used to make `add_submission` non-blocking and process submissions parallelly:
 - The function `add_submission` pushes the submission to a queue. Submissions are processed in the `processor` thread. Submission is made non-blocking in this way.
 - The `processor` thread maintains the metadata in `to_check` and initiates tasks in `pipe` queue. The thread `processor` performs the actual processing of comparing and flagging submissions.
- **Concurrency:**
 - Concurrency is maintained using `mutex` and locks to avoid data races when shared data is used.
 - Conditional variable `cv` sends signals so that thread waits passively.
 - The atomic variable `stop` is used to terminate the code properly in the presence of threading.

2.4 File Identification

When a submission is made, it is checked against each file submitted before it via the vector `to_check`. If a submission was made within one second before the submission, even that is flagged if it was not already flagged. The vector maintains every submission's pointer, timestamp, and status(flagged/not flagged).

2.5 Helpers and Data Structures

- **Helpers:**
 - `void plagiarism_checker_t::process_submissions():` Updates metadata of each new submission.
 - `int plagiarism_checker_t::curr_time_millis():` Gives the current time according to the `system_clock`.
- **Data Structures:** The data structures `queue` and `vector` are used.
- **Flow:**
 - In the constructor, threads are started, and the vector is initialized with initial submissions.
 - When `add_submission` is called, the submission is pushed to a queue.
 - Processing is done by the other thread, leaving the main thread free to add other submissions.
 - When the destructor is called, threads are notified that there will be no more submissions and joined to the main thread.