CS293
Data Structures and Algorithms

Project Report
Plagiarism Checker (Phase 1 & 2)

Aditi Singh (23B1053)

Aakash Gupta (23B0953)

Sabil Ahmad (23B1057)

Indian Institute of Technology, Bombay

1 Phase 1

1.1 The algorithm and function signatures for Net Accurate Matches and Longest Approximate Matches

1.1.1 Accurate Match

Function signature:

```
void match ( std::vector<int> &submission1 , std::vector<int> &submission2
, std::vector<bool> &match_at_index_i, std::vector<bool> &match_at_index_j
, std::unordered_map<int, int> &match_length_at_i, int &min_length , int
&total_match_length );
```

- Iterate through both the submission and compare every possible starting index of one submission with that of another. If a consecutive match is found, it updates total match length ie sum of lengths of all valid matches, matched indices in both lists are maintained and skipped when getting repeated to avoid unnecessary double matching.

1.1.2 Approximate Longest Match Length

Function signature:

```
void fuzzy_approx (std::array<int, 5>& result_accurate, std::vector<int>&
submission1, std::vector<int>& submission2, int max_mismatches);
```

- We have implemented Levenshtein in a one way fashion. For deletion, we have allowed 20% mismatches between the two. For insertion, we have iterated the algorithm in reverse direction in which deletion for submission 2 act as insertion for submission 1 and thus efficiently we account for both insertion and deletion in the files.

1.2 Helpers and function signatures

- **Helper function:**

  ```
  void update( int &curr_match ,int &total_length_match, int
  &min_length ,int &i ,int &j ,std::unordered_map<int ,int>
  match_length ,std::vector<bool> &match_at_index_i ,std::vector<bool>
  &match_at_index_j);
  ```

  This function will update match length at particular index of submission 1 to prevent double counting and also updates flag of indices as true if a match is found.

- **Data structure used:** Apart from vector<int> of tokens and array<int, 5> for result, we have used **unordered_map<int,int>** for mapping match length to indices of the tokens.

- **Overall workflow:** match_submission calls accurate_match and fuzzy_match and give them reference for result array which update their respective entries.

1.3 Time and space complexity

Assuming n, m are length of submission 1 and submission 2.

- **Time complexity:** O(nm)
- **Space Complexity:** O($n + m$)

2 Phase 2

2.1 Data structures, helper function and flow of algorithm

2.1.1 Data Structure

We are using a **queue** which stores pair of timestamps and the pointer to submissions of unchecked files for processing and **set<pair<index, matchlength»** for storing patches from different files for patchwork plagiarism.

2.1.2 Helper functions:

Function **find_matches** calls **match** which has been used from phase 1.
Apart from this, **queue_processing** takes the unprocessed files in the queue and processes them in the thread of our class and calls **check_plagiarism**.

2.1.3 Workflow

- **Constructor**: Initializes *finished* to false, starts *processing_thread* for asynchronous processing, and records the timestamp of the moment. If submissions are provided, tokenizes their files and stores in the submissions vector for accessing it later.
- **Adding Submissions:** *add_submission* computes the timestamp for a new submission relative to *startTime*, pushes it to the unchecked queue, and notifies the *processing_thread* to begin processing if idle.
- **Processing Queue:** The *processing_thread* waits for new submissions or the finished flag. When notified, it processes submissions from the unchecked queue by calling *check_plagiarism*.
- **Destructor:** Sets finished to true, notifies the *processing_thread* to terminate, waits for it to join, and clears the submissions vector for a clean shutdown.

2.2 The algorithm and function signatures for long, short and patched length matches

```
void plagiarism_checker_t::check_plagiarism(std::pair<double, std::shared_ptr
<submission_t» sub1);
```

- Submission is pushed by constructor to the queue and then pop the element getting checked for plagiarism. *check_plagiarism* iterate through previous submissions which is **vector<** **<timestamp, type, submission, tokens»** where type is 0 if original, 1 if unplagged and plagged if 2.
- Based on appropriate conditions, *find_matches* function will return vector<pair<int,int» that will store indices and corresponding length of matches >= 15. For long matches, if any match in the vector has length >= 75, we will flag the file as plagged. However, for short matches, we will check if size of vector >= 10.
- For **patched** plagiarism, we have maintained a set of pair which stores pair of match at an index and length of that match from that index, and then take the **non interleaved disjoint unions** for total matches.

Time complexity: $O(mn^2)$
Space complexity: O(mn) where m is number of files checked by and n is number of tokens per file.

2.3 Threading and concurrency features

- A mutex (mtx) ensures that only one thread at a time can access or modify the unchecked queue or the finished flag. This prevents race conditions and ensures thread-safe operations.
- The condition variable (cv) allows the worker thread to sleep when queue is empty. When a new submission is added, the cv.notify_one() method wakes up the worker thread efficiently.
- The main thread can call add_submission without waiting, as the submission is simply added to the queue and handled asynchronously.
- By locking and unlocking explicitly, the program avoids deadlocks and race conditions. Further, no undefined behavior as the queue is always accessed under a mutex lock.

2.4 Checking submission timestamp

We set a chrono time clock in the constructor which is called when an object is instantiated, and when added a submission we see the program time and subtract it from the time stored earlier. Since, the whole processing part takes place in a different thread there is no blocking between two add submissions, thus noting correct submission timestamp. Now further while checking any file we could se this corect timestamp to corectly identify if it was less than 1s or not.

References

[1] https://www.geeksforgeeks.org/std-mutex-in-cpp/

[2] https://www.javatpoint.com/multithreading-in-cpp-with-examples