Phase 2: Algorithm Description & Analysis

Pattern-Detecting Functions & Algorithms

Main Comparator Flow

- The class `plagiarism_checker_t` is responsible for detecting exact and patchwork plagiarism.
- The core function `process_submissions()` acts as a driver that runs in a background thread, handling new submissions asynchronously.
- The comparison logic is delegated to:
  - `check_plagiarism()` for exact match detection.
  - `is_patchwork_plagiarism()` for patchwork detection.

check_plagiarism Function

- Accepts two tokenized streams (new and old submissions).
- Scans all pairs of positions i, j, where $i \in A$, $j \in B$:
  - Returns `true` immediately if:
    a) A contiguous match of ≥ 75 tokens is found.
    b) ≥ 10 matches of length ≥ 15 are found.

is_patchwork_plagiarism Function

- Extracts all 15-token windows from the new submission.
- Checks if the same token window appears in any existing submission.
- If ≥ 20 such unique windows match, the submission is flagged.

Threaded Architecture & Asynchronous Execution

- The plagiarism detection framework uses multi-threading to maintain responsiveness and process efficiency.
- **Worker Thread (Background)**
  - Spawned by the constructor using `std::thread`.

  - Runs the `process_submissions` function, looping infinitely until termination is requested.
  - Waits on a `std::condition_variable` and safely dequeues a submission.
  - Tokenizes and compares the submission against the internal database.
  - Appends to database after analysis.
- **Main Thread (Submission Interface)**
  - Receives calls to `add_submission` externally.
  - Locks shared resources with `std::mutex` and pushes the submission to the queue.
  - Triggers `notify_one` to signal the background worker.
- **Thread Lifecycle Management**
  - Controlled via the `stop_processing` boolean flag.
  - On destructor call, this flag is flipped.
  - The worker thread then terminates gracefully and is joined to avoid resource leaks.
- This threading design ensures that:
  - The main program never blocks while checking submissions.
  - Processing is parallelized, allowing continuous acceptance of new submissions.

Time & Space Complexity

- **Worst-case time complexity:**

  $$O(n \cdot m + n^2 \cdot m \log m)$$

  where $n = |A|$ and $m = |B|$.
  - $O(n \cdot m)$: For comparing each pair of tokens in the naive match logic.
  - $O(n^2 \cdot m \log m)$: Involves nested loops during patchwork and fuzzy matching, especially when hashing token substrings.
- **Worst-case space complexity:**

  $$O(n^2 \cdot m)$$

  - Accounts for simultaneous storage of all token streams, match results, and pattern windows.
  - Token hash strings and rolling buffer structures used in `join_tokens` and patchwork comparison further add to usage.
- **Average-case performance:**
  - On realistic datasets with short to medium-length matches and well-distributed tokens, time reduces to roughly $O(n \cdot m)$ or below.
  - Pattern matching short-circuits early upon reaching thresholds (75 tokens or 10 short matches), making it efficient.

Helper Functions

- `join_tokens(tokens, start, end)` – Converts a slice of integer tokens into a space-separated string.
- `are_timestamps_close(s1, s2)` – Checks if two submissions were made within 100 ms.
- `is_later(s1, s2)` – Resolves tie by checking timestamp difference or submission ID.
- `flag_plagiarized(plag, source)` – Flags students and professors involved in plagiarism.

Control Flow for Plagiarism Detection

1. **Initialization**:
    - The constructor sets up initial timestamps and launches the worker thread.
2. **Task Submission**:
    - `add_submission()` pushes the new submission onto a thread-safe queue.
3. **Tokenisation**:
    - Inside the worker, `safe_tokenize` extracts integer tokens.
4. **Comparison**:
    - For every submission in the database:
        a) `check_plagiarism` is called.
        b) If not flagged, `is_patchwork_plagiarism` is evaluated.
5. **Flagging**:
    - `flag_plagiarized` is called with appropriate IDs and references.
6. **Database Update**:
    - Newly processed submissions are appended to the internal database.
7. **Termination**:
    - Destructor sets the stop flag, waits on condition variable, and joins the background thread.