MARMARA UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

CSE 4077 Advanced Data Structures - Project 1

Group Members:

Emre Cihan Varlı - 150119711

Ayşe Sena Aydemir - 150119735

1. **Direct Comparison Method**

Description: For each query $LCE_{A,B}(i, j)$, the method directly compares the characters in the two texts A and B.

Advantages:

- Simplicity: The implementation is straightforward and does not require any preprocessing of the data.
- No Additional Data Structures: Directly works on the input texts without constructing suffix arrays or LCP arrays.

Disadvantages:

- Time Complexity: Each query may take O(n) in the worst case, where n is the length of the shorter suffix starting at positions i and j. For multiple queries, this results in O(q.n), where q is the number of queries.
- Inefficient for Multiple Queries: Since there is no preprocessing, each query is handled independently, leading to repeated comparisons and poor scalability for large datasets or multiple queries.

2. **Generalized Suffix Array with RMQ**

Description: This method preprocesses the input texts to build a generalized suffix array (SA) and a longest common prefix array (LCP). It then uses a Range Minimum Query (RMQ) data structure (e.g., Fischer-Heun structure) to efficiently find the minimum value in the LCP array for the range determined by the suffix array indices of the query.

Advantages:

- Efficiency: Using Fischer-Heun, RMQ queries on the LCP array are answered in O(1), making this method highly efficient for multiple queries.
- Scalability: Well-suited for scenarios with a large number of queries due to its efficient query time.

Disadvantages:

- Preprocessing Overhead: Requires preprocessing time depending on the RMQ structure, which may be significant for very short input texts or single queries.
- Complexity of Implementation: Requires additional data structures and algorithms, making it more complex to implement compared to the direct comparison method.

3. **Summary**

The Direct Comparison Method is preferable for small inputs or when the number of queries is small, as it avoids the overhead of preprocessing. The Generalized Suffix Array with RMQ Method is ideal for scenarios with large inputs or multiple queries.

4. Details of Our RMQ Structure

The Range Minimum Query structure in our implementation uses the Fischer-Heun algorithm, which achieves O(n) preprocessing time and O(1) query time. This structure is applied to the Longest Common Prefix array, enabling efficient computation of minimum LCP values for a given range.

Preprocessing:

- Block Partitioning: The LCP array is divided into blocks of $B = \lfloor \log_2 n \rfloor$, where n is the length of the array. This ensures that there are $\lceil n/B \rceil$ blocks, each containing up to B elements.
- Block Minimums: For each block, the minimum value is computed and stored in the block_min array. These block minimums represent the global minima for each block and are used to build a Sparse Table for inter-block queries.
- Intra-block Lookup Tables: For each block, a lookup table is built to store the minimum value for all possible ranges within that block. This enables efficient O(1) intra-block queries.
- Sparse Table Construction: The block_min array is used to build a Sparse Table, which facilitates O(1) inter-block range queries.

Query:

- Identify Blocks: For a range [l, r] in the LCP array, determine the blocks that contain the indices l and r.
- Intra-block Queries: Query the lookup tables of the first and last blocks to find the minimum values for the partial ranges within those blocks.
- Inter-block Query: If the range spans multiple blocks, use the Sparse Table to find the minimum value for the blocks fully contained in the range.
- Combine Results: Return the minimum value of the results from intra-block and inter-block queries.

5. Results

The Fischer-Heun RMQ structure efficiently preprocesses the LCP array in O(n) time, where n is the array size, and handles each query in O(1), independent of the array size or query range. It ensures accurate computation of minimum values for any range within the LCP array and is particularly suited for scenarios with many queries, as the preprocessing cost is distributed over the queries.