Project 07 - Genome Analysis Tool

This project introduces a console-based C++ genome analysis tool designed for computer science students and bioinformatics developers. It showcases the practical application of Data Structures and Algorithms (DSA) in processing and analyzing DNA sequences, addressing challenges in medical research, diagnostics, and broader bioinformatics fields.

Modular System Architecture

Our tool is built upon a modular design, ensuring clear separation of concerns, ease of extension, and maintainability—a crucial aspect for complex bioinformatics pipelines.

SequenceStore

Manages DNA sequences and their associated metadata.

SuffixTrie

Powers exact pattern searching for rapid sequence identification.

RollingHash

Facilitates efficient substring hashing for motif discovery.

MotifFinder

Identifies frequently occurring patterns in the genome.

QueryEngine

Orchestrates both exact and approximate search operations.

Core Objectives: Bridging DSA with Bioinformatics

Efficient DNA Management

Develop robust mechanisms to store, retrieve, and manage complex DNA sequence data with unique identifiers.

Advanced Pattern Matching

Implement algorithms for fast exact pattern matching and sophisticated motif detection within genomic sequences.

Mutation Handling

Incorporate approximate matching capabilities to account for real-world DNA variations and mutations.

Real-World Application

Translate theoretical DSA knowledge into tangible solutions for biological problems, fostering a deeper understanding of computational biology.

SequenceStore: The Genomic Repository

The SequenceStore module is the foundational component, responsible for the persistent storage and retrieval of DNA sequences.

Key Functionality:

- Stores DNA sequences and their descriptive names.
- Assigns unique numerical IDs to each sequence for efficient referencing.
- Enables rapid retrieval and clearing of stored genomic data.

```
Sample Output: Total sequences stored: 3
ID 0:
ACGTGTCAGTACGATCGTACGTTAGCTAGCTGACGTAGCTAGCTG
ATCGTAC
ID 1: TTAGGCACGT
ID 2: CCGATTAAGC
```

Real-World Applications:

- Centralized genome databases (e.g., NCBI GenBank).
- Secure storage for hospital patient DNA records.
- Forensic DNA evidence management.
- Repositories for viral genome sequences.

Suffix Trie: Precision in Exact Matching

The SuffixTrie data structure provides an optimal solution for locating exact patterns within DNA sequences, crucial for gene identification.

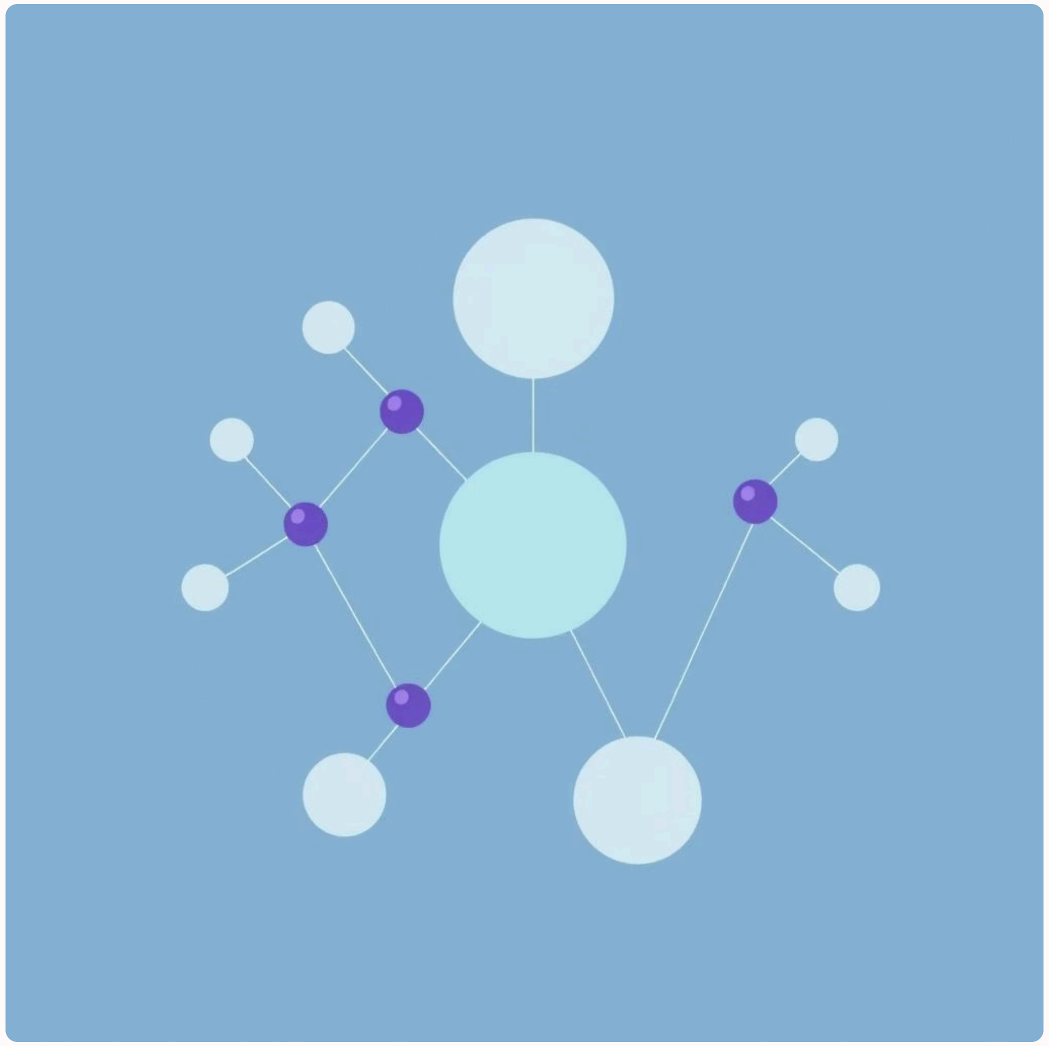
Mechanism:

- All suffixes of a DNA sequence are inserted into the trie.
- Enables highly efficient searches for exact patterns.
- Returns all starting positions where the pattern is found.

```
Sample Output: Searching pattern: ACG
 Pattern found at positions: 0, 10, 18, 32
```

Impactful Applications:

- Rapid gene identification in large genomes.
- Detection of specific disease markers.
- Forensic DNA matching for identification.
- Comparative analysis of bio-sequences.



Rolling Hash: Efficient Substring Processing

The RollingHash algorithm drastically improves the performance of substring comparisons by leveraging a sliding window approach.

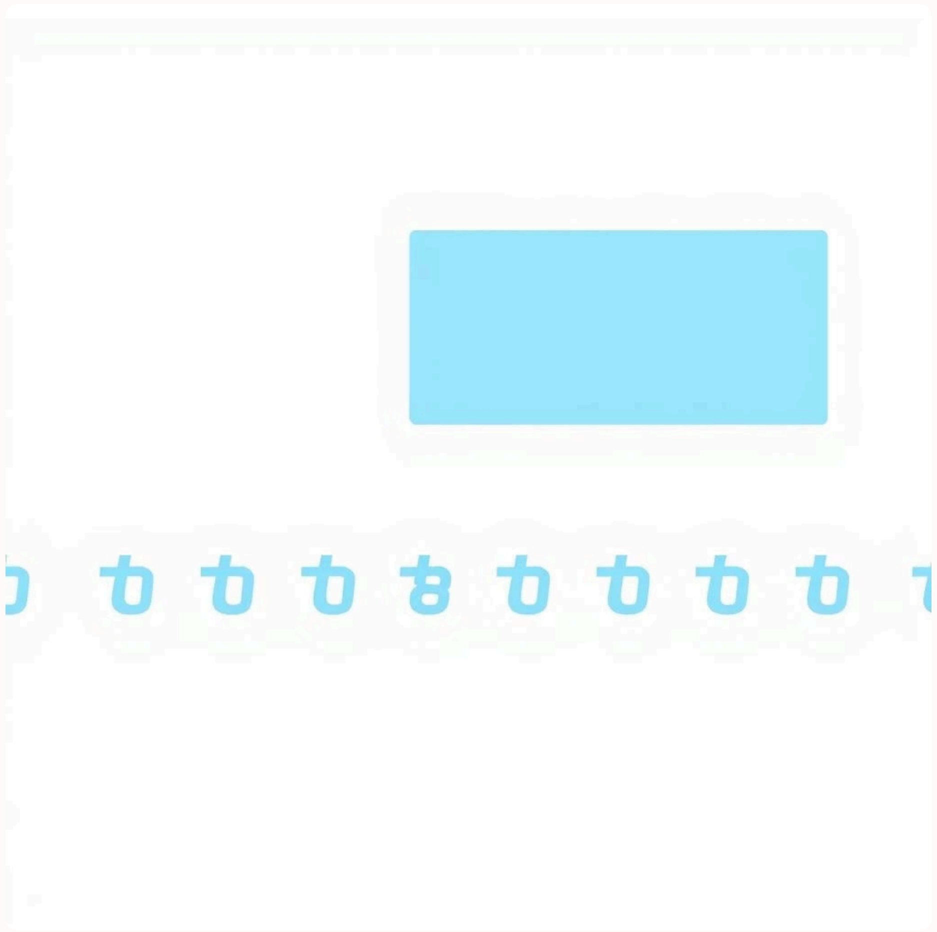
Core Principle:

- Computes the hash of a substring in constant time, given the previous substring's hash.
- Avoids redundant recomputation of hashes for overlapping windows.
- Achieves a time complexity of O(n), making it highly scalable for long sequences.

Sample Output: Window: ACGTG | Hash: 110 Window: CGTGT | Hash: 443 ... (continues for other windows)

Diverse Applications:

- Accelerating genome scanning and sequence alignment tools.
- Detecting plagiarism in large text corpuses.
- Analyzing network packets for anomaly detection.
- Identifying malware signatures in cybersecurity.



Made with GAMMA

Motif Finder: Uncovering Biological Patterns

The MotifFinder module utilizes the power of rolling hash and frequency counting to pinpoint frequently occurring substrings (motifs) within DNA.

Motif Identification

Scans sequences to identify short, recurring patterns that often have biological significance.

Frequency Analysis

Calculates the absolute count and percentage frequency of each detected motif.

Biological Insight

Helps in understanding regulatory elements and functional regions of the genome.

Sample Output:

```
CGT | Count: 5 | Frequency: 10%
 GCT | Count: 4 | Frequency: 8%
```

Significance in Research:

- Analysis of gene regulation and transcription factor binding sites.
- Detection of protein binding domains.
- Advancing cancer research by identifying oncogenic motifs.
- Facilitating drug discovery and vaccine development.

QueryEngine: Navigating Genomic Data

The QueryEngine acts as the central interface for users, enabling both exact and approximate searches across all stored DNA sequences.

Exact Search

Locates precise matches of a given pattern across the entire dataset.

Example: `SeqID: 0 | Name: Sample DNA 1 | Pos: 4 | Match: GTC`

Approximate Search

Identifies patterns allowing for a specified number of mismatches (edit distance).

Example: `Match: GAT | Distance: 1`

Approximate Search: Real-World Imperfections

Approximate search is critical because real-world DNA contains natural variations, mutations, and sequencing errors. This feature simulates these imperfections, providing a more realistic and robust analysis.

- **Mutation Detection:** Crucial for identifying genetic diseases.
- **Virus Strain Comparison:** Tracing the evolution and spread of pathogens.
- **Evolutionary Studies:** Understanding genetic divergence across species.
- **Personalized Medicine:** Tailoring treatments based on individual genomic profiles.
- **DNA Error Correction:** Identifying and correcting sequencing artifacts.

Performance

Our design choices prioritize efficiency, scalability, and maintainability, aligning with the demands of real-world bioinformatics pipelines.

Optimized Algorithms

Suffix Trie for fast exact searches, Rolling Hash for efficient substring processing, and Hash Maps for rapid frequency counting.

Modular Design

A structured approach ensures the system is scalable, easy to maintain, and adaptable to future enhancements.

Time & Space Complexity Analysis

Sequence Storage

- Insert sequence: **O(n)**
- Space: **O(n)** per sequence

Suffix Trie

- Build Trie: **O(n²)** (inserting all suffixes)
- Exact search: **O(m + k)**
  - *m* = pattern length
  - *k* = number of occurrences
- Space: **O(n²)** (acceptable for educational dataset sizes)

Rolling Hash

- Initial hash: **O(k)**
- Rolling update per window: **O(1)**
- Total motif scan: **O(n)**

Query Engine

- Exact search: **O(m)** per query
- Approximate search (edit distance ≤ 1):
  - **O(n·m)** (acceptable due to small pattern size)
- Space: **O(r)** where *r* = number of matches

Motif Finder

- Sliding window scan: **O(n)**
- Hash collision handling: **O(1)** average
- Overall: **O(n)** time, **O(n)** space

Code-Level DSA Integration

- `SequenceStore` handles **data validation and storage**
- `SuffixTrie` and `TrieNode` enable **fast exact substring search with position tracking**
- `RollingHash` supports **efficient sliding-window hashing**
- `MotifFinder` applies hashing for **frequency-based pattern discovery**
- `QueryEngine` integrates these structures to perform **exact and approximate searches**

Algorithmic Trade-offs Made Explicit

- A **suffix trie** was used instead of repeated string scans to reduce search time
- **Rolling hash** was preferred to avoid recomputing hashes for every window
- Approximate search was implemented with controlled edit distance to balance **simplicity and correctness**

Robustness Through Edge-Case Handling

The implementation correctly handles:

- Empty DNA sequences
- Invalid characters in input
- Motif lengths greater than sequence length
- Safe skipping of sequences during approximate matching.