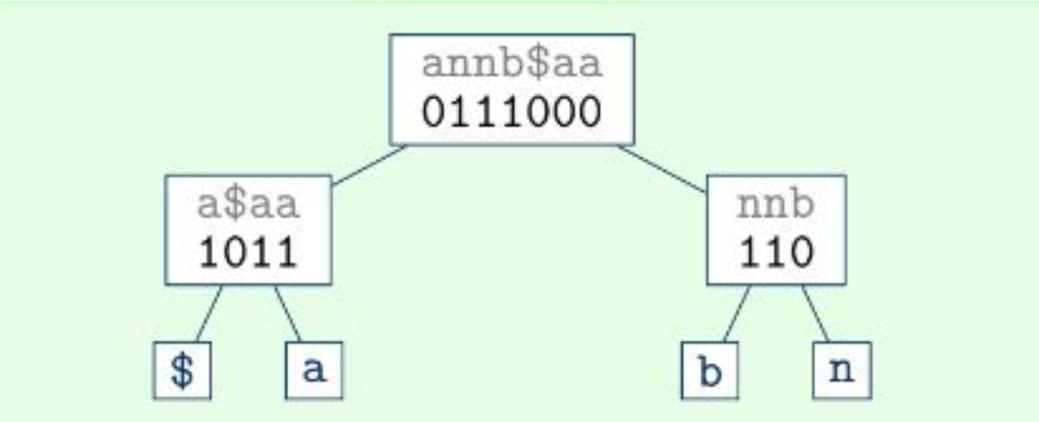
Implementation of Wavelet Trees

Computational Biology
Prof. Dominik Kempa

Introduction



- Supports rank queries for σ > 2
- Space: O(n log σ) bits and query: O(log σ) time.

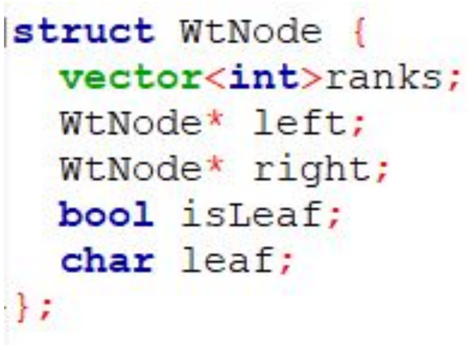
Latest Developments

Can be divided into 4 categories:

- Wavelet Matrix:
  - Stores bits in a matrix
  - Faster access and more efficient range queries
- Succinct Wavelet Trees:
  - Very high compression ratios
- Dynamic Wavelet Trees:
  - Suited for sequences that are updated in real time
- Wavelet Trees for Graphs:
  - Wavelet tree-based structures for representing and querying graph data which provides significant improvements in efficiency over traditional graph representations

Implementation details

- Implemented in C++
- At each level, instead of bit vectors, we store rank vectors. We do this to support 0(1) rank queries.
- The space complexity of the implementation will be O(nlogσ) words and the time complexity is O(nlogσ)
- Node level details



```
struct WtNode {
   vector<int>ranks;
   WtNode* left;
   WtNode* right;
   bool isLeaf;
   char leaf;
};
```

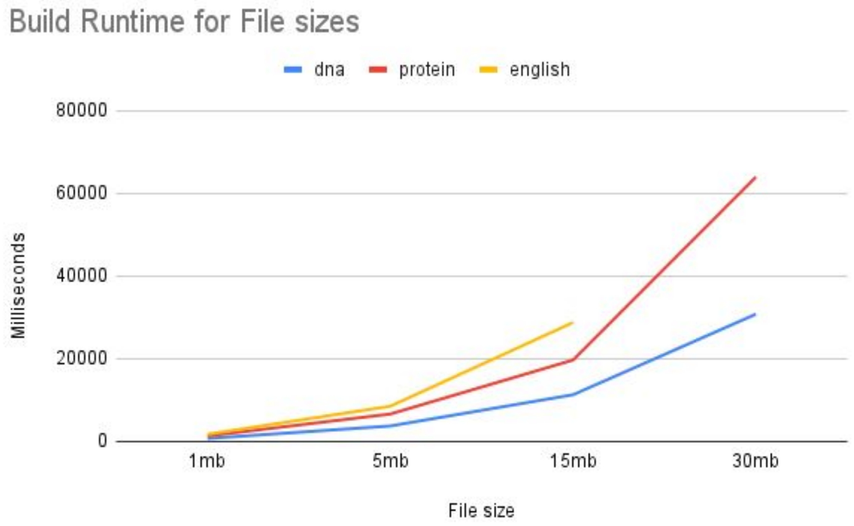
Experimental Setup

- Used DNA sequences, Protein Sequences and English text data from Pizza&Chilli repository.
- Each of these have different sigma values.
- The file sizes we used for the experiments are 1mb, 5mb, 15mb and 30mb.
- We measure
  - Build runtime for the wavelet tree
  - Access and Rank query times
  - RAM usage
  - CPU instructions
- Runtimes are measured using chrono library in C++
- RAM and CPU usage are measured using Valgrind tool in Linux.

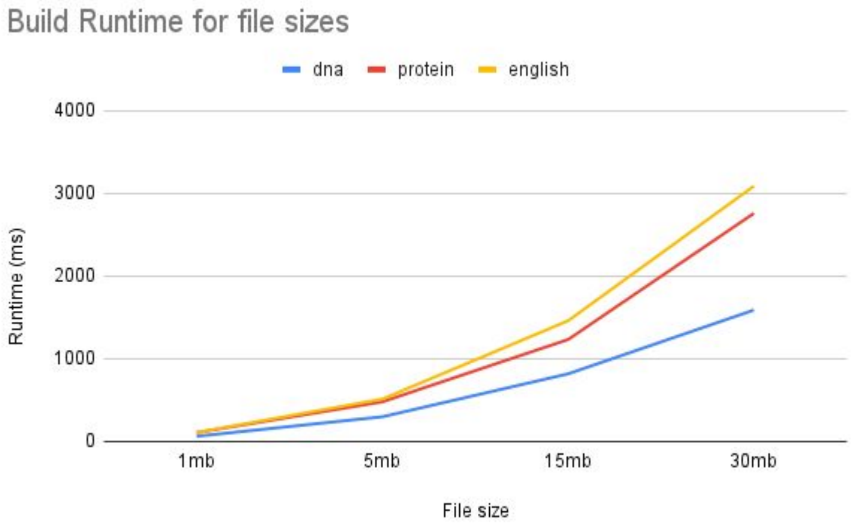
Machine Specifications

- 2 GB Ram
- 4 core processor Intel i7
- Run on Linux using a Virtual Machine.

Experimental Results : Build Runtime

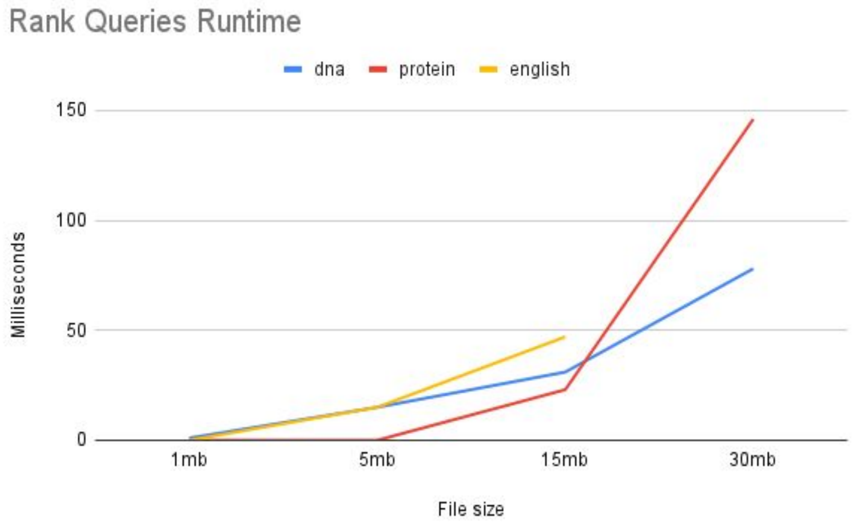


Our Implementation

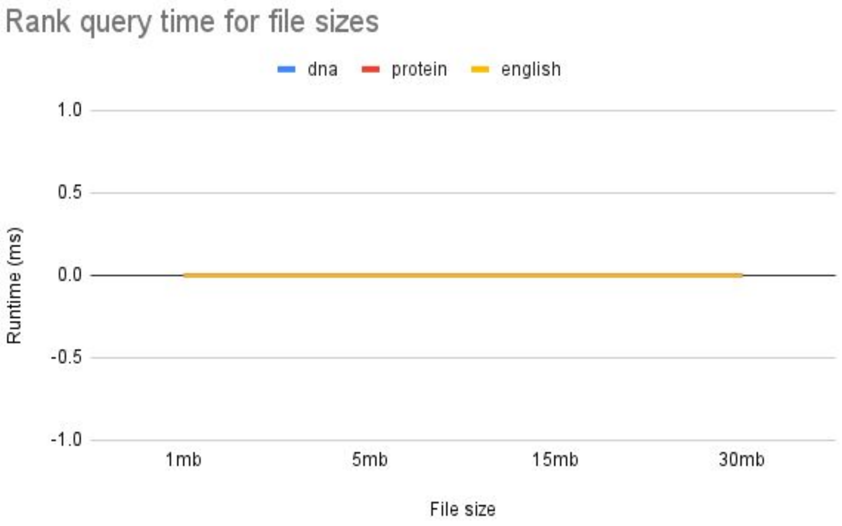


SDSL

Experimental Results : Rank Runtime

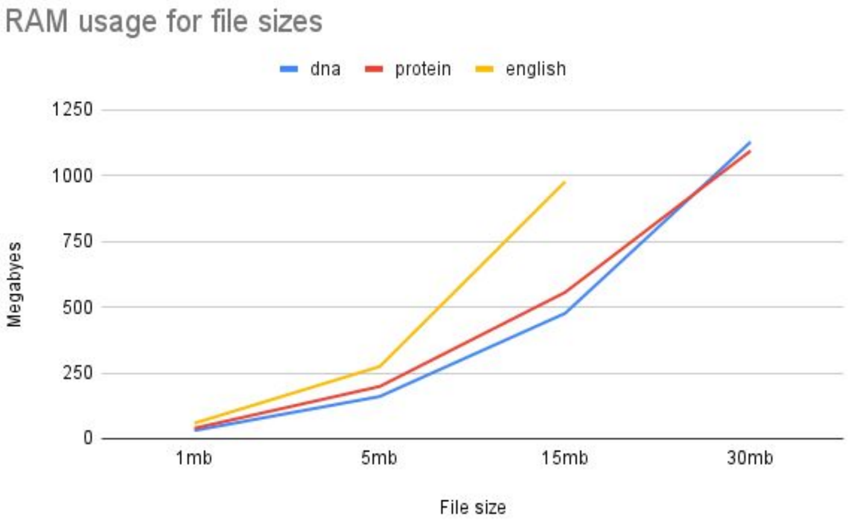


Our Implementation

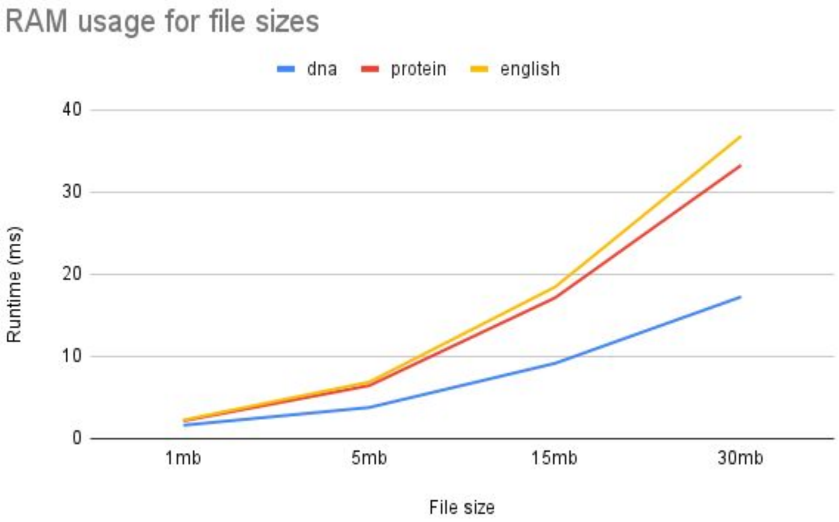


SDSL

Experimental Results : RAM Usage

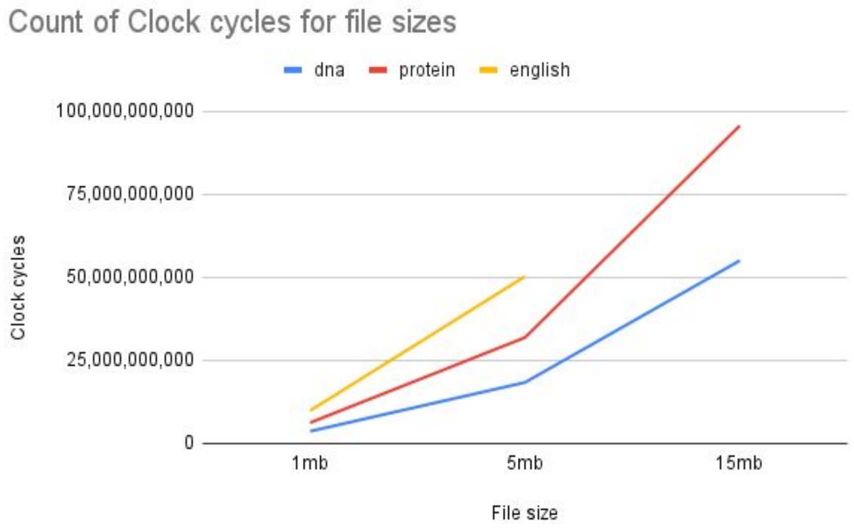


Our Implementation

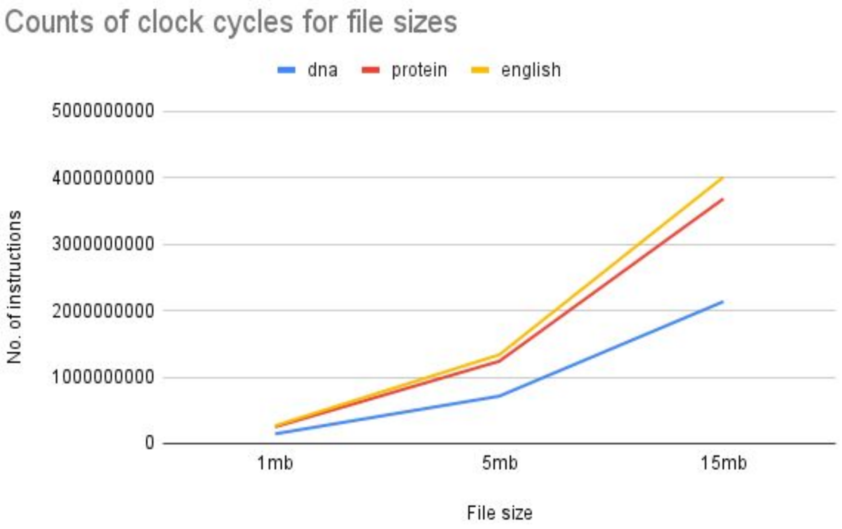


SDSL

Experimental Results : CPU Usage



Our Implementation



SDSL

Analysis of Results

- As sigma increases, the amount of RAM and CPU usage increases exponentially for our implementation.
- We were able to run a 50mb file for smaller sigma values (less than 50).
- The increase in RAM is mainly due to the in-memory Rank vectors we store at each node level.
- We still are able to support fast access and rank queries which take O(log σ) time.
- To improve on our current performance, we can implement Jacobson’s Rank algorithm which supports O(1) rank queries on bit vectors.