Assignment 0*

Abdel Rahman Alsabbagh
abdelrahman.sabbagh@kaust.edu.sa

Implementation Description

In this assignment's script, I implemented the Knuth-Morris-Pratt (KMP) [1] algorithm for both exact and approximate pattern matching in large genomic sequences, ensuring efficient memory usage by processing chromosomes one at a time. I read a DNA sequence pattern from a FASTA file and searched for occurrences of this pattern in a genome file, logging the positions of any matches. For approximate matching, I allowed up to one mismatch (which could be a substitution, insertion, or deletion). I achieved this through simplified character comparisons instead of generating all mismatch possibilities, which would have taken forever to finish. To optimize performance, I used KMP's prefix function and tracked both execution time and memory usage with `tracemalloc`. The script was designed to be run with command-line arguments, allowing me to choose between exact or approximate matching modes.

Important Notes

I developed two versions of the code: one that loads the entire genome into memory at once, and another that reads the genome line by line. In the second version, once a chromosome is fully processed, the code performs pattern matching on that chromosome and then deletes it from memory to optimize resource usage. In this report, I will only discuss the latter code, since it uses the least memory, although the former was much faster. Both codes are attached with the submission.

Additionally, to make sure my code is performing the task correctly, I created a bunch of small unit tests on a set of sample texts instead of directly running it on the genomes. I also attached this code for your reference.

You will find the instructions for running all codes in the last section. All codes were run on an Apple M3 Max chip with 36GB of RAM.

Results on AluY Counts

For context, both genome assemblies have 709 chromosomes.

In GRCh38 (hg38)

Chromosome	Start	End	Match Type	Time (s)	Peak Memory (KB)
CM000665.2	88629832	88630142	Exact match	1267.6564	36.47
CM000669.2	39808489	39808799	Exact match		
CM000679.2	18689763	18690073	Exact match		

Table 1: Exact matching on GRCh38 (hg38). Total number of matches found were 3.

*CS 249: Algorithms in Bioinformatics

Chromosome	Start	End	Match Type	Time (s)	Peak Memory (KB)
CM000663.2	85940122	85940432	Approximate match (S)	16766.2398	38.86
CM000663.2	144846234	144846544	Approximate match (S)		
CM000665.2	69313857	69314167	Approximate match (S)		
CM000665.2	88629832	88630142	Exact match		
CM000666.2	132898458	132898768	Approximate match (S)		
CM000667.2	125028156	125028466	Approximate match (S)		
CM000669.2	39808489	39808799	Exact match		
CM000670.2	100790760	100791070	Approximate match (S)		
CM000674.2	62838639	62838949	Approximate match (S)		
CM000675.2	63082060	63082370	Approximate match (S)		
CM000676.2	34836097	34836407	Approximate match (S)		
CM000677.2	53225105	53225415	Approximate match (S)		
CM000679.2	18689763	18690073	Exact match		
CM000679.2	19593503	19593813	Approximate match (S)		
CM000683.2	26516833	26517143	Approximate match (S)		
KI270778.1	54847	55157	Approximate match (S)		

Table 2: Approximate matching on GRCh38 (hg38). Total matches with at most 1 mismatch were 16. There are three types of mismatches: (S)ubstitution, (I)nsertion, and (D)eletion).

In T2T CHM13v2.0

Chromosome	Start	End	Match Type	Time (s)	Peak Memory (KB)
CP068276.2	186833277	186833587	Exact match	1199.3436	36.34
CP068271.2	39966041	39966351	Exact match		

Table 3: Exact matching on T2T CHM13v2.0. Total number of matches found were 2.

Chromosome	Start	End	Match Type	Time (s)	Peak Memory (KB)
CP068277.2	143946938	143947248	Approximate match (S)	16134.1556	36.80
CP068277.2	190858894	190859204	Approximate match (S)		
CP068276.2	186833277	186833587	Exact match		
CP068275.2	69350794	69351104	Approximate match (S)		
CP068274.2	136221657	136221967	Approximate match (S)		
CP068273.2	126911808	126912118	Approximate match (S)		
CP068272.2	51234801	51235111	Approximate match (S)		
CP068272.2	68212687	68212997	Approximate match (S)		
CP068271.2	39966041	39966351	Exact match		
CP068271.2	49189460	49189770	Approximate match (S)		
CP068270.2	101916771	101917081	Approximate match (S)		
CP068267.2	86916177	86916487	Approximate match (S)		
CP068266.2	62817411	62817721	Approximate match (S)		
CP068266.2	79100751	79101061	Approximate match (S)		
CP068265.2	66839102	66839412	Approximate match (S)		
CP068264.2	17947515	17947825	Approximate match (S)		
CP068261.2	2142369	2142679	Approximate match (S)		
CP068261.2	19541649	19541959	Approximate match (S)		
CP068260.2	66026304	66026614	Approximate match (S)		
CP068257.2	24874951	24875261	Approximate match (S)		

Table 4: Approximate matching on T2T CHM13v2.0. Total matches with at most 1 mismatch were 20. There are three types of mismatches: (S)ubstitution, (I)nsertion, and (D)eletion).

1 Performance Analysis

In my performance analysis, I observed significant differences in execution time and memory usage between the two genome assemblies, GRCh38 (hg38) and T2T CHM13v2.0, as well as between the two code implementations. For exact matching, GRCh38 took approximately 1267.66 seconds with a peak memory usage of 36.47 KB, while T2T CHM13v2.0 completed in 1199.34 seconds with a slightly lower peak memory usage of 36.34 KB. In approximate matching, GRCh38 identified 16 matches in 16766.24 seconds, whereas T2T CHM13v2.0 found 20 matches in 17992.76 seconds, both with similar memory footprints. Notably, all approximate matches I reported were of the (S)ubstitution type, meaning that the mismatches involved a single character substitution rather than (I)nsertions or (D)eletions.

I also found that the optimized code, which processes chromosomes line by line, demonstrated a remarkable reduction in peak memory usage. It maintained an extremely low peak of around 36 KB, compared to the around 9863 MiB required when loading the entire genome into memory at once. This highlighted the effectiveness of the line-by-line approach in minimizing memory consumption, making it suitable for environments with limited resources. However, this memory efficiency came at the cost of increased execution time, as the optimized code took significantly longer to complete both exact and approximate matching tasks compared to the normal code, which loaded the entire genome at once. Despite the slower performance, the peak memory usage remained almost constant across all experiments, indicating stable and efficient memory management. This trade-off between memory efficiency and execution time underscored the importance of choosing the appropriate approach based on available resources and performance requirements. Overall, T2T CHM13v2.0 exhibited better performance in terms of execution time, likely due to its more recent and refined assembly, while both assemblies maintained comparable memory efficiency.

Code Execution Instructions

I developed two versions of the code: one that loads the entire genome into memory at once (named `CS249_Assignment0_Normal_AbdelRahman_Alsabbagh.py`), and another that reads the genome line by line (`CS249_Assignment0_Optimized_AbdelRahman_Alsabbagh.py`). In the second version, once a chromosome is fully processed, the code performs pattern matching on that chromosome and then deletes it from memory to optimize resource usage.

Additionally, to ensure the correctness of my code, I created a set of small unit tests on sample texts rather than running the code directly on the full genomes. These unit tests are included in the file `CS249_Assignment0_Unit_AbdelRahman_Alsabbagh.py` for your reference.

To run the code, use the following commands:

- Read the `requirements.txt` file to load all dependencies:

```
pip install -r requirements.txt
```

- For exact match on the GRCh38 (hg38) / `hg38` genome:

```
python CS249_Assignment0_Optimized_AbdelRahman_Alsabbagh.py --genome <hg38_Fasta>
--pattern <Pattern_Fasta> --output results_exact_match_hg38.txt
```

- For Approximate match (S) on the GRCh38 (hg38) / `hg38` genome:

```
python CS249_Assignment0_Optimized_AbdelRahman_Alsabbagh.py --genome <hg38_Fasta>
--pattern <Pattern_Fasta> --output results_approximate_match_hg38.txt --approx
```

- For exact match on the T2T CHM13v2.0 / `t2t` genome:

```
python CS249_Assignment0_Optimized_AbdelRahman_Alsabbagh.py --genome <t2t_Fasta>
--pattern <Pattern_Fasta> --output results_exact_match_t2t.txt
```

- For Approximate match (S) on the T2T CHM13v2.0 / `t2t` genome:

```
python CS249_Assignment0_Optimized_AbdelRahman_Alsabbagh.py --genome <t2t_Fasta>
--pattern <Pattern_Fasta> --output  results_approximate_match_t2t.txt --approx
```

- For unit testing:

```
python CS249_Assignment0_Unit_AbdelRahman_Alsabbagh.py
```

Disclaimer

I utilized GenAI models to enhance the aesthetics of my code, particularly in writing the documentation and discussing the arguments and return values of each function, as well as in improving this very report. Aside from that, all other aspects of the work were completed independently.

References

[1] Donald E Knuth, James H Morris, Jr, and Vaughan R Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.