

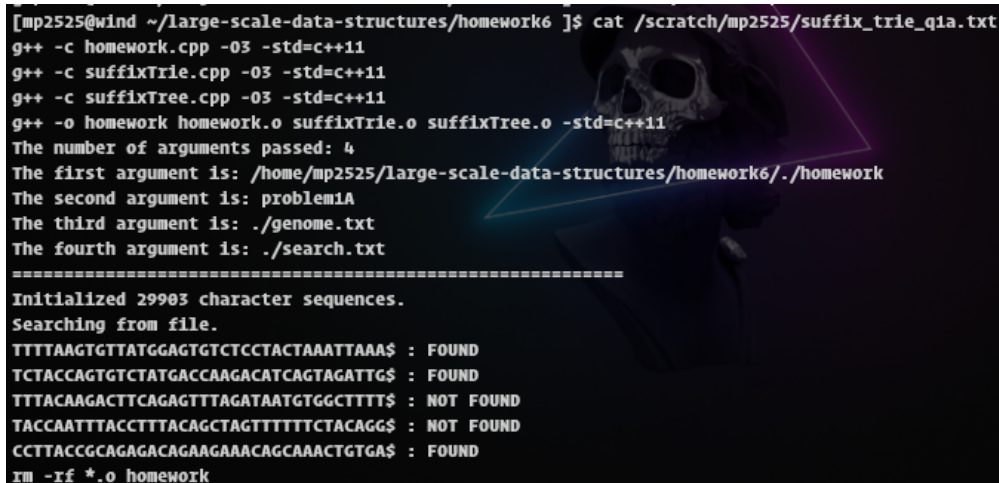
## INF503\_Assignment\_6\_mp2525 - Solutions

### Problem #1 (of 2): Suffix trie

#### A. Implementing a basic (perfect match only) search function

- Read in the SARS-COV2 genome sequence (Appendix A) and store it in the suffix trie.
- Implement a perfect match search function, which would take in a sequence fragment, search for presence of that fragment in the suffix trie and report presence/absence of the fragment in the genome (finding a single hit is ok – no need to find all possible hits).
- Use read fragments in appendix B to test your search function.

Ans:



```
[mp2525@wind ~/large-scale-data-structures/homework6]$ cat /scratch/mp2525/suffix_trie_q1a.txt
g++ -c homework.cpp -O3 -std=c++11
g++ -c suffixTrie.cpp -O3 -std=c++11
g++ -c suffixTree.cpp -O3 -std=c++11
g++ -o homework homework.o suffixTrie.o suffixTree.o -std=c++11
The number of arguments passed: 4
The first argument is: /home/mp2525/large-scale-data-structures/homework6/./homework
The second argument is: problem1A
The third argument is: ./genome.txt
The fourth argument is: ./search.txt
=====
Initialized 29903 character sequences.
Searching from file.
TTTTAAGTGTATGGAGTGTCTCCTACTAAATTAAS$ : FOUND
TCTACCAGTGTCTATGACCAAGACATCAGTAGATTG$ : FOUND
TTTACAAGACTTCAGAGTTTAGATAATGTGGCTTTT$ : NOT FOUND
TACCAATTTACCTTTACAGCTAGTTTTTCTACAGG$ : NOT FOUND
CCTTACCGCAGAGACAGAAGAAACAGCAAACTGTGA$ : FOUND
rm -rf *.o homework
```

The method ***readSequenceFromFile*** reads the SARS-COV2 genome sequence (Appendix A) and stores it in the suffix trie. Also, It is being used in the constructor.

The method ***searchSequence*** takes the sequence and sequence length to search the perfect match on the suffix trie. It returns the bool value where true means the presence of the sequence in the suffix trie.

The method ***searchSequencesFromFile*** takes the file path that contains the fragments in appendix B. The search performed on these fragments has the following results.

Fragment #	Fragment	Search Result
1	TTTTAAGTGTTATGGAGTGTCTCCTACTAAATTAAA	FOUND
2	TCTACCAGTGTCTATGACCAAGACATCAGTAGATTG	FOUND
3	TTTACAAGACTTCAGAGTTTAGATAATGTGGCTTTT	NOT FOUND
4	TACCAATTTACCTTTACAGCTAGTTTTTTTCTACAGG	NOT FOUND
5	CCTTACCGCAGAGACAGAAGAAACAGCAAAGTGTGA	FOUND

#### B. Stress testing your search function

- What is the size of the trie (# of nodes)?
- Generate 5K, 50K, and 100K random 36-mers from the SARS-COV2 genome sequence and use them to search the suffix tree.
  - How many of your 36-mers had a match? Does it make sense? Explain why.
  - How long did it take (big O notation estimate).

Ans:

```
[mp2525@wind ~/large-scale-data-structures/homework6] $ cat /scratch/mp2525/suffix_trie_q1b.txt
g++ -o homework homework.o suffixTrie.o suffixTree.o -std=c++11
The number of arguments passed: 3
The first argument is: /home/mp2525/large-scale-data-structures/homework6/./homework
The second argument is: problem1B
The third argument is: ./genome.txt
=====
Initialized 29903 character sequences.
Suffix Trie Size: 446931004
5000 random sequences: 5000 sequences found
Time taken: 0.007617 seconds.
=====
50000 random sequences: 50000 sequences found
Time taken: 0.052136 seconds.
=====
100000 random sequences: 100000 sequences found
Time taken: 0.091108 seconds.
=====
rm -rf *.o homework
```

The size of the trie (# of nodes) is 446901100 (without DS) and 446931004 (including DS).

# of Random 36 mers	Total Match	Time taken
5000	5000	0.007617 seconds
50000	50000	0.052136 seconds
100000	100000	0.091108 seconds

It does make sense to me because the random sequences used for the search are generated from the subject (SARS-COV2 genome sequence) which yield the perfect match for all the sequences.

The big O notation for the search is:  $O(M \times N) \sim O(M \times 36) \sim O(M)$   
where M is the number of random sequences and N is the sequence length which is constant.

## Problem #2 (of 2): Suffix tree – extra credit (50 points)

### A. Implementing a basic (perfect match only) search function

- Read in the SARS-COV2 genome sequence (Appendix A) and store it in the suffix tree.
- Implement a perfect match search function, which would take in a sequence fragment, search for presence of that fragment in the suffix tree and report presence/absence of the fragment in the genome (finding a single hit is ok – no need to find all possible hits).
- Use read fragments in appendix B to test your search function.

Ans:

```
[mp2525@wind ~/large-scale-data-structures/homework6]$ cat /scratch/mp2525/suffix_tree_q2a.txt
g++ -c homework.cpp -O3 -std=c++11
g++ -c suffixTrie.cpp -O3 -std=c++11
g++ -c suffixTree.cpp -O3 -std=c++11
g++ -o homework homework.o suffixTrie.o suffixTree.o -std=c++11
The number of arguments passed: 4
The first argument is: /home/mp2525/large-scale-data-structures/homework6/./homework
The second argument is: problem2A
The third argument is: ./genome.txt
The fourth argument is: ./search.txt
=====
Initialized 29903 character sequences.
Searching from file.
TTTTAAGTGTTATGGAGTGTCTCCTACTAAATTAAS : FOUND
TCTACCAGTGTCTATGACCAAGACATCAGTAGATTG$ : FOUND
TTTACAAGACTTCAGAGTTTAGATAATGTGGCTTTT$ : NOT FOUND
TACCAATTTACCTTTACAGCTAGTTTTTCTACAGG$ : NOT FOUND
CCTTACCGCAGAGACAGAAGAAACAGCAAAGTGTGA$ : FOUND
rm -rf *.o homework
```

The method ***readSequenceFromFile*** reads the SARS-COV2 genome sequence (Appendix A) and stores it in the suffix trie. Also, It is being used in the constructor.

The method ***searchSequence*** takes the sequence and sequence length to search the perfect match on the suffix trie. It returns the bool value where true means the presence of the sequence in the suffix trie.

The method ***searchSequencesFromFile*** takes the file path that contains the fragments in appendix B. The search performed on these fragments has the following results.

Fragment #	Fragment	Search Result
1	TTTTAAGTGTTATGGAGTGTCTCCTACTAAATTA	FOUND
2	TCTACCAAGTGTCTATGACCAAGACATCAGTAGATTG	FOUND
3	TTTACAAGACTTCAGAGTTTAGATAATGTGGCTTTT	NOT FOUND
4	TACCAATTTACCTTTACAGCTAGTTTTTTTCTACAGG	NOT FOUND
5	CCTTACCGCAGAGACAGAAGAAACAGCAAAGTGTGA	FOUND

The result matches with problem 1A that uses suffix trie.

## B. Stress testing your search function

- What is the size of the tree (# of nodes)?
- Generate 5K, 50K, and 100K random 36-mers from the SARS-COV2 genome sequence and use them to search the suffix tree.
  - How many of your 36-mers had a match? Does it make sense? Explain why.
  - How long did it take (big O notation estimate). Compare with estimates in Problem #1

Ans:

```
[mp2525@wind ~/large-scale-data-structures/homework6]$ cat /scratch/mp2525/suffix_tree_q2b.txt
g++ -c homework.cpp -O3 -std=c++11
g++ -c suffixTrie.cpp -O3 -std=c++11
g++ -c suffixTree.cpp -O3 -std=c++11
g++ -o homework homework.o suffixTrie.o suffixTree.o -std=c++11
The number of arguments passed: 3
The first argument is: /home/mp2525/large-scale-data-structures/homework6/./homework
The second argument is: problem2B
The third argument is: ./genome.txt
=====
Initialized 29903 character sequences.
Suffix Tree Size: 88912
5000 random sequences: 5000 sequences found
Time taken: 0.001297 seconds.
=====
50000 random sequences: 50000 sequences found
Time taken: 0.012924 seconds.
=====
100000 random sequences: 100000 sequences found
Time taken: 0.02648 seconds.
=====
rm -rf *.o homework
[mp2525@wind ~/large-scale-data-structures/homework6]$
```

The size of the trie (# of nodes) is 88879 (without DS) and 88912 (including DS). The size is smaller than the problem 1B that uses suffix trie. It is because the suffix tree compresses the sequences in a smaller number of nodes.

# of Random 36 mers	Total Match	Time taken
5000	5000	0.001297 seconds
50000	50000	0.012924 seconds
100000	100000	0.02648 seconds

It does make sense to me because the random sequences used for the search are generated from the subject (SARS-COV2 genome sequence) which yield the perfect match for all the sequences.

The big O notation for the search is:  $O(M \times N) \sim O(M \times 36) \sim O(M)$   
where M is the number of random sequences and N is the sequence length which is constant.

The big O notation is the same for suffix trie and suffix tree when neglecting the constant traverse of 36 or less nodes in the tree. However, the execution time is small in the suffix tree because it doesn't have to traverse N (36) nodes in the tree unlike suffix trie. It traverses less than 36 nodes. The suffix tree with the compression of the sequence saves the memory and yields the fast search operation.