## Thant Zin Oo (Andy) | 705-172-144

Project 4 Report

# Known bugs and problems:

The only known problem I have is when I try to run the program on the SEASNET server. For some reason, I can’t load the genome files using the provided test mainframe. I suspect it’s because the path I’ve specified within the main function is wrong. Z:/Desktop/proj4 seas/data

The files load properly when I run the program under Visual Studio however. Also, I can manually input genomes when running on the SEASNET server, which is how I tested my program under g++.

# Big-O Satisfaction

All my functions (as far as I know) satisfy the spec’s complexity requirements.

class Trie

Trie();

//O(1)

~Trie();

//O(N) where N is the number of total nodes

void reset();

//O(N) because it destructs the trie

void insert(const std::string& key, const ValueType& value);

//O(LC) where L is the length of the key and C is the average number of children

std::vector<ValueType> find(const std::string& key, bool exactMatchOnly) const;

//O(V + LC) if exactMatchOnly == true, O(V + L2C) = O(V + LC) if exactMatchOnly == false

class GenomeImpl

GenomeImpl(const string& nm, const string& sequence);

//O(1) Because the constructor arguments are passed by reference and not copied.

static bool load(istream& genomeSource, vector<Genome>& genomes);

//O(2L) = O(L) where L is the char length of the input file

int length() const;

//O(1)

string name() const;

//O(1)

bool extract(int position, int length, string& fragment) const;

//O(S) where S is the length of the entire genome sequence

class GenomeMatcherImpl

GenomeMatcherImpl(int minSearchLength);

//O(1)

~GenomeMatcherImpl();

//O(N) where N is the total number of nodes in the genomeMatcher trie.

void addGenome(const Genome& genome);

//O(SM) where M is the minimum search length and S is the length of the entire genome. This is because I iterate S-M times. S is guaranteed to be > M, therefore I can generalize the term to S. For each iteration S, I call the extract method with complexity O(M).

int minimumSearchLength() const;

//O(1)

bool findGenomesWithThisDNA(const string& fragment, int minimumLength, bool exactMatchOnly, vector<DNAMatch>& matches) const;

//O(HF + HlogH) where H is the time complexity of finding each match O(V + LC) and F is the length of the fragment. The function calls my Trie’s find() function, with O(HF) complexity. It then iterates through all the matches for the length of the fragment, which is complexity (HF). Sorting the matches requires HlogH time, which in most cases will be < HF, but is not certain, hence the complexity O(HF + HlogH)

bool findRelatedGenomes(const Genome& query, int fragmentMatchLength, bool exactMatchOnly, double matchPercentThreshold, vector<GenomeMatch>& results) const;

//O(S/F \* H + RlogR) where S is the length of the query sequence, F is the length of the fragmentMatch, H is the general complexity of findGenomesWithThisDNA(), and R is the number of related genomes found. This is because my function calls findGenomes for each complete F length fragment of S. Sorting the related genomes requires O(RlogR), which in most cases should be < S/F \* H, but is not certain. Therefore my time complexity is O(S/F \* H + RlogR).

# Pseudocode for find() and findGenomesWithThisDNA()

find()

{  
 create empty match vector

determine first character

for each of root node’s children

check if path denoted by character exists

if so, call findHelper() only with that path, moving down a node and char

return match vector

}

findHelper()

{

if there is more than one mismatch, return

if we’ve reached the end of the key string, append all values to the match vector

otherwise, check the current character in the key string (which is always the next character from the previous call of findHelper())

for each of the children in the current node

if a path denoted by the current character exists

call findHelper() with that path, moving down a node and char

otherwise, if looking for SNiPs

call findHelper() with that path, moving down a node and char, and increasing the number of mismatches

}

findGenomesWithThisDNA()

{  
 if the length of the fragment length is less than minimum length, or minimum length is less than minimum search length, return false

clear match vector

format search parameter to be of length minSearchLength

call find(), searching for the formatted search parameter and store it in a separate vector of matches

for each found match

denote how many characters it matches with the fragment, either exactly or allowing for SNiPs

sort the found matches by ascending order of name and descending order of match lengths. This ensures that the match vector will be left in a state wherein all matches of the same genome name will be in a continuous chunk, with the first match of a unique genome being the largest match

call the unique algorithm to remove consecutive duplicates

for each unique match

check if the match length is at least as big as the minimum length allowed

if so, add the match to original match vector

return true if the match vector contains matches, false otherwise

}