Report

Xiaoshu Luo

904971914

1. Potential problems:

My program compiles with no known bugs or problems, but it can be very slow when using the test main provided by the website.

1. Time complexity issue:

**(1)For Trie:**

Trie() satisfies the big O requirement.

~Trie() satisfies the big O requirement.

void reset() satisfies the big O requirement.

void insert(const string& key, const ValueType& value) satisfies the big O requirement.

std::vector<ValueType> find(const std::string& key, bool exactMatchOnly) const satisfies the big O requirement.

**(2)For Genome:**

Genome(const std::string& nm, const std::string& sequence) satisfies the big O requirement.

~Genome() satisfies the big O requirement.

Static bool load(std::istream& genomeSource, std::vector<Genome>& genomes)

int length() const satisfies the big O requirement.

std::stringname() const satisfies the big O requirement.

bool extract(intposition,int length, std::string& fragment) const satisfies the big O requirement.

**(3) For GenomeMatcher:**

GenomeMatcher(int minSearchLength) satisfies the big O requirement.

~GenomeMatcher() satisfies the big O requirement.

void addGenome(const Genome& genome) satisfies the big O requirement.

int minimumSearchLength() const satisfies the big O requirement.

bool findGenomesWithThisDNA(const std::string& fragment, int minimumLength, bool exactMatchOnly, std::vector<DNAMatch>& matches) const does not satisfy the big O requirement. Its big O is H\*(F+V), in which V is the size of the “matches ” vector.

bool findRelatedGenomes (const Genome& query, int fragmentMatchLength, bool exactMatchOnly, double matchPercentThreshold, std::vector<GenomeMatch>& results) const does not satisfy the big O requirement. Its big O is N+(Q\*(X+M)), in which N is the number of genomes in the genomematcher and M is the average number of matches for every fragment.

1. Pseudocode:

*my node structure:*

*each node contains a vector of template values and 26 child pointers that correspond to the 26 upper letters.*

Find\_helper()

*Takes in a node pointer (start from the root), a key to be checked, the size of the original key, exactMatch indicator, and the vector container.*

*Base condition: If the key is empty string (initial condition)*

*Fill the container with the values pointed by the pointer;*

*Stop;*

*Base condition: if the first letter of the key doesn’t match and key’s length is equal to the original value, meaning that it is also the first letter of the complete key:*

*Stop;*

*Base condition: if the size of the key is one:*

*If the corresponding child pointer is not null,*

*fill the container with the values pointed by that child.*

*If exact match is still not required:*

*For all the other child pointers that are not null:*

*Fill the container with the values;*

*Stop;*

*If the first letter is found to exist in current node’s child:*

*Recursively:*

*Try find if the next letter is found in the next layer by following that corresponding child pointer and passing in the substring of the key that excludes the first letter until the key only has one letter. .*

*If the first letter is not found:*

*If the exactMatch is not required:*

*For each existing child letter in the current node:*

*Recursively:*

*Try find if the next letter of the key is found in the next layer by following all existing child pointers and passing in the substring of the key that excludes the first letter until the key only has one letter. Since a mismatch like this happens for once, then exactMatch is true for the following part of the key.*

Find()

Create an empty container;

Get the size of the input key;

*pass in the key to the helper function which starts from the root to fill the container.*

*Return the container;*

GenomeMatcher's findGenomesWithThisDNA()

*each node contains a vector of pairs of two integers, in which the first one is the index of the genome in the genome and the second is the position in that genome where the match appears.*

*If the trivial conditions are met:*

*Use a container to hold all pairs of potential matches by applying find method to the first “minimumsearchlength” letters of the fragment.*

*Repeatedly for each potential pair:*

*Extract the corresponding part of the genome indicated by the pair;*

*Repeatedly:*

*Extract the corresponding part with one more letter on the end;*

*See if this extended part still matches with the first “minimumsearchlength+1” letters of the fragment.*

*If no matches, break;*

*If the extended part reaches the maximum value of the fragment, break;*

*If the resultant extended part is longer than miminumLength passed in:*

*If trivial conditions are met for this part,*

*Add a corresponding DNA struct into the “matches” container.*

*If at least one struct is added, return true;*

*Else return false;*