**Aditya Pimplaskar**

**UID: 804900470**

**Project 4: Report**

1. **BUGS IN CLASSES/ISSUES:**

My classes worked generally according to the expected behavior. They built on both Xcode and g32.

1. **ISSUES WITH BIG-O REQUIREMENTS**

I should meet the Big-O requirements. Though I may have some extra loops, they are still not going to influence the Big-O as they will not be part of the term of greatest power.

1. **METHODS:**

*Trie::Find*

* This function calls the findRequireFirstMatch function in my TrieNode class on the root node
* *TrieNode::find*
  + Base case:
    - If the sequence size is 0 (we have reached the end of the sequence we want to find): return values
  + If the child’s key is the same as the current character in the sequence:
    - We will call find recursively on the child, maintaining exactMatchOnly
  + Else if exactMatchOnly is false
    - We will call find recursively on child, setting the exactMatchOnly parameter to true
  + Return result
* *TrieNode::findRequireFirstMatch*
  + Allocate a vector to hold results
  + Recursively searches through nodes downstream of the Node that is passed in
  + Base case:
    - If the sequence size is 0 (we have reached the end of the sequence we want to find): return values
  + Loop over the vector of children that each node has
    - If the key of the node matches the character in the sequence we are searching for: recursively call *find* on that child with a substring of the sequence
      * Keep track of exactMatchOnly
      * Store the results of that recursive call in a temporary vector
    - Loop over the temporary vector and push those results into the outer results vector
  + Return results.
* The idea with this function was to keep recursively finding a subsequence as we progress further down the Trie. The nature of the exactMatchOnly parameter allows us flexibility with checking for mismatches and capping mismatches at one. The find method did not maintain that the first elements had to match. In turn, findRequireFirst adds this provision but uses the functionality of the find function.

*GenomeMatcher::findGenomesWithThisDNA*

* This function finds a substring of length m\_searchLength to find an initial group of matches that will be a starting point where we will scan the rest of the sequence.
* For each fragment substring that we found, I scanned through the characters, comparing them with sequences within the m\_genomes hash table
  + This was done in the *scan* function (listed below)
  + The scan function returns an int that reflects the match length
* The returned match (from scan) values and position (from trie find returning values)
* Had to handle case if a match had already been registered for that genome
  + If so: compared match length and if the match length of the new scan was longer, that genome’s match updated
  + If not, add a new match to the matches vector
* Returns true if the matches vector is not empty, false if it is emtpy
* Before this function runs, have to check:
  + If fragment.length() < minimumLength: false
  + If minimumLength < minimumSearchLength() of matcher: false
* *Scan* function
  + Takes in a reference to a DNA match, a startIndex, a reference to a fragment string, the Boolean if exactMatchOnly, and an integer for minimumLength
  + We get the genome we are looking for by using the hash table
    - Finding the element at the key: genome name
  + We will extract the sequence from the startIndex to the end of the sequence
    - If the extract fails: we return -1 which is a flag to say that scan failed
  + Comparing the sequence that is passed in to the extracted sequence from the genome:
    - There is a counter that counts matches between the extracted sequence and the parameter sequence
    - If exactMatchOnly is false, we are allowed one error: I handle this by using a Boolean
    - If exactMatchOnly is false or one error has already been allowed, then the comparison loop breaks and returns the counter
      * This counter is the length of the match
  + All that is left is ensuring that the length of the match exceeds the minimum match length
    - If so, we return this match length
    - If not, we return -1 as a flag for failure

**DESIGN DECISIONS:**

The STL container that I used in my GenomeMatcher class was a hash table (as the spec allowed for me to use any STL container). This made finding/accessing genomes more efficient.