WRITE UP
AKWARANDU UGO NWACHUKU
COMP 15 - Project 1

What data structures will you be using for this project, and why were those choices made? If using more than one data structure, how will they interact? If you are modifying a known data structure, what changes will you be making and why?

Tries give us a way to cut short the distance of search by defining a path on each inserted sequence. I employed the use of a Trie because I believe it is easier to visualize than say using a dynamic array or even a binary tree. I initially thought to use a binary tree, splitting the inserted node by which character comes first in the alphabet, but then I realized that didn't make too much sense and it'd also create a very unbalanced tree. I believe a Trie counteracts the potential unbalanceness that may arise in our database. In handling operations, such as search, the queried string traverses a path with similar characters until the similarity is broken or until the end of the sequence is reached. This saves the trouble of using a "for loop" to check the similarity of the queried string and the stored sequence. The "for loop" runs at a worst case time of $O(n^{(\text{length}(\text{sequence}))})$, whereas a tree may run at O(n.log (n)), where n is the length of the sequence.

How do you plan on solving insert, query and remove requests?

Insert

The insert operation inserts a sequence into the root node of the trie. Each node represents a character of the sequence at each level of the trie. In the case a sequence similar to another sequence stored in the trie is inserted, the insertion does not create a repeat node for each character of the inserted sequence, but instead traverses the same stored sequence until a dissimilarity in the next character to be inserted appears, the inserted sequence then branches off at the root of difference to form another bst. The new bst then continues until the last character in the sequence. The last character in any sequence is represented by a boolean representation, claiming true to notify the end of a sequence. When inserting question marks we create four nodes representing all the possible characters that may be inserted.

Query

The query function should work in a similar fashion to the node count function from homework 4.
We recursively search each child of the root, for a similar character with the inserted sequence.
If we find the character, we increment an integer variable "match" and recursively search the next node: comparing the next prefix in our inserted sequence and the character at that node. We also keep track of how deep in the tree we are by adding finding the tree height at each node level. We do this recursive call on all nodes in the tree that match. Once we meet a node that is dissimilar, we call a function that finds the tree height of the sequence with the closest match. We then find the max between the length of the sequence with the closest match and the length of the query word. In the case of a tie breaker, when we have two sequences with the same prefix in common and the same length, we use the converter_function to find the int value of the next unmatched character, and pick the sequence that is lowest in value.

Remove

To remove we search our tree for the inserted sequence to be removed, if found; we change the eow last character in the sequence to false. In that singular case, the sequence to be removed is a subset of another longer sequence. In the case we are removing a sequence that has no branches, we delete all the nodes passed into the remove function. We have to be careful when we are removing a sequence ACGT from a trie that contains ACGT and ACT. We have to make sure only to delete GT, ensuring we keep the sequence ACT. So there has to be a way to cut off between ACGT and ACT. We have to keep removing till we hit a node with more than just one child.

Why do you think your solution is a good solution to this project? How are space and time concerns addressed?

I believe my solution is a good solution to the problem, best case for searching is O(logn) worst case is O(n). IN terms of space complexity I believe a trie saves more space than a dynamic array for example, because it uses pointers.

In a general sense, how will your code be structured? What classes will you write? What are some of the most important functions you must implement?

I intend to implement a trie class and a struct for a node. Each node has with it four children each representing: A, C, T, G. In each node also, there is an end of word(EOW) marker which is a boolean. True represents if the character is the end of a sequence and false represents otherwise. My code should be structured with a Trie.cpp, Trie.h, and a main.cpp; the background material would be the makefile and the test_main.cpp i intend to make for testing. Another function that may come in handy is the print Trie function, that'd help visualize how the sequence is getting stored. And finally, some important functions that may come in handy are helper functions to shorten and clean up the code (e.g converter_func).

How much code have you written so far?

So far, I've written my makefile, a bit of my header file and my cpp file as well.