Program 3: Recursion (60 Points)

**Objectives:**

The focus of this assignment is to create and use a recursive method given a moderately difficult problem.

**Program Background:**

**DNA**, or deoxyribonucleic acid, is a molecule that contains genetic instructions for living organisms. Each is composed of a sequence of four bases, adenine (A), cytosine (C), guanine (G), and thymine (T). These sequences can be represented by an ordered string of the symbols for the bases. When comparing two strands of DNA we can look at their **longest common subsequence**. The longer this string the closer the DNA strands are related.

**Subsequences** do not require the characters be adjacent in the string, only the order matters. **Substrings** are a subset of **subsequences**, as characters in substrings must be consecutive. The **common substring** of two strings is a string which appears in both strings. The **longest common substring** is the string out of these that has the most characters. For this problem we will only be concerned with the more restrictive **substring**.

**Program Description:**

The program user will enter a DNA character sequence as the master sequence to be compared. They will then enter any number of additional candidate DNA sequences. For each of these candidate sequences entered it will check if its **longest common substring** with the master sequence is longer than any seen previously. If so, the master sequence will be updated with the value of its best match and the **common substring**. Once all candidates have been entered the program will report its findings.

The following classes are required:

* InvalidDNAStrandException
* DNA
* DNADriver

**UML DIAGRAM FOR AND DISCUSSION FOR InvalidDNAStrandException**

|  |
| --- |
| InvalidDNAStrandException extends RuntimeException |
|  |
| <<constructor>> InvalidDNAStrandException( )  <<constructor>> InvalidDNAStrandException(molecule : char) |

There will be two reasons that cause a DNA strand to be invalid. If the strand is longer than 15 characters (this is due to size limitation of an int) and if there are characters in the strand other than ‘A’, ‘C’, ‘G’, or ‘T’. There are two constructors, one for each problem, giving a different String message for the Exception to reflect the problem.

Methods:

InvalidDNAStrandException( ) – Class constructor. Calls upon super class constructor with the String “Strand too long”

InvalidDNAStrandException(molecule : char) – Class constructor. Calls upon super class constructor with the String, “Invalid molecule: “, followed by the character that is passed.

**UML DIAGRAM FOR AND DISCUSSION FOR DNA**

|  |
| --- |
| DNA |
| - strandString : String  - bestMatch : DNA  - substringMatch : String |
| <<constructor>> DNA(strand : String)  + getStrand( ) : String  + setStrand(strand : String)  + toString( ) : String  + checkMatch(matchCandidate : DNA)  **- RabinKarp(find : String) : int**  **- stringHash(s : String) : int**  **- charNumericValue(c : char) : int**  **- RabinKarpHashes(hashes : int[], pos : int, length : int) : int**  **- linearSearchRecursive(data : int[], key : int, pos : int) : int** |

An instance of DNA will primarily store the String sequence of that DNA strand. It also has the ability to store a reference to another instance of DNA as well as its **longest common substring** with that DNA. Methods are also included to update this information given a new DNA candidate.

Data Members:

strandString – String representation of the DNA strand

bestMatch – Reference to another instance of DNA that is the best match found

substringMatch – The **longest common substring** between this strand and the DNA referenced by bestMatch

Methods:

DNA(strand : String) – Class constructor. Sets the passes String to strandString by calling upon setStrand. Sets bestMatch to null and substringMatch to the empty String, “”.

getStrand( ) : String – returns strandString

setStrand(strand : String) – Attempts to set strandString data member to the String passed. If the passed String is invalid due to it being more than 15 characters or there are invalid characters in the String it will create and throw an InvalidDNAStrandException, calling upon the appropriate constructor.

toString( ) : String – If there is a valid match, will return a String of the following form:

DNA Strand: GATCG

Best Match Strand: AATCT

Matching Substring: ATC

If there is no valid match it will output the strand and the message, “No match found”, as show below:

DNA Strand: GATCG

No match found

checkMatch(matchCandidate : DNA) – Checks if the passed DNA candidate has a longer substring than is currently stored. Uses the Rabin-Karp algorithm for substring matching. In order to do this, you will need to break the candidate strand into all possible substrings and send them to the RabinKarp method. This is a bit, “Brute Force” (there are more efficient but complex ways of doing this), however we can mitigate the work by eliminating a lot of possible substrings. A substring from the candidate that is longer than strandString can be skipped. All substrings whose length is shorter than the current substringMatch can also be skipped. Additionally, if we process the longest substrings first, we can stop once a match is found and not bother with the shorter substrings.

RabinKarp(find : String) : int - Does the preprocessing of finding the hash for the substring find using the stringHash method and the hashes of substrings in the search string using RabinKarpHashes method. Calls upon linearSearchRecursive to determine if the substring hash is in the collection of hashes and returns the result.

stringHash(s : String) : int - Given a string, return the hash value of the entire String. Use a base 4 number system to create the hash as described in class. This will be needed only to find the hash of the substring that is being searched for and the base case for finding all substring hashes in the search string.

charNumericValue(char c) : int - Given a character, returns the numeric value of the character in our base 4 number system. Assign the possible characters, ‘A’, ‘C’, ‘G’, and ‘T’ values 0 – 3.

RabinKarpHashes(hashes : int[], pos : int, length : int) - Finds the hash values of all substrings of size length in the strandString String, starting at index pos and working down. These values are stored in the hashes array that is passed. **This method must be recursive, using the technique as described in the Rabin-Karp lecture.**

linearSearchRecursive(data : int[], key : int, pos : int) : int - This is a recursive linear search. Return the position of key in the data array, or -1 if it is not present. **This method must be recursive.**

**Note:** These last five methods are declared private, as they are only for internal use within the DNA class.

**DISCUSSION FOR DNADriver**

See expected output below discussion, if your output is different in any way you are subject to losing points.

The user will be asked to enter a String for the master DNA strand and attempt to create an instance of DNA. If an invalid strand is entered an InvalidDNAStrandException should occur, notify the user of their error and reprompt until a valid strand is input.

User is then prompted to enter any number of candidate strands. If an invalid strand is entered the user is notified and it is not processed. Valid strands will be sent to the master’s checkMatch method as they are input. At any time the user may enter the String, “EXIT”, to stop inputting candidates. The DNA master strand is then output, reporting the results.

**Helpful methods:**

You may find the following methods in the java API helpful

String class method:

public String substring(int beginIndex, int endIndex) - Returns a new string that is a substring of this string. The substring begins at the specified beginIndex and extends to the character at index endIndex - 1. Thus the length of the substring is endIndex-beginIndex.

**Example Runs (User input underlined)**

Example 1:

Enter molecule string for master DNA strand: ACTGGTXGA

Invalid molecule: X, try again.

Enter molecule string for master DNA strand: GTACCTAGTACCTTGACTA

Strand too long, try again.

Enter molecule string for master DNA strand: GTACCAGTCCGGTAC

Enter molecule string for comparison or EXIT to quit: ACGTATCCGGTAGTC

Enter molecule string for comparison or EXIT to quit: GTCTCATGCCAOC

Invalid molecule: O, strand ignored.

Enter molecule string for comparison or EXIT to quit: GGACTCCAGCTTC

Enter molecule string for comparison or EXIT to quit: CCATGGACATCAATCGACTTGAC

Strand too long, strand ignored.

Enter molecule string for comparison or EXIT to quit: ACCGTTGATTACCAA

Enter molecule string for comparison or EXIT to quit: EXIT

DNA Strand: GTACCAGTCCGGTAC

Best Match Strand: ACGTATCCGGTAGTC

Matching Substring: TCCGGTA

Example 2:

Enter molecule string for master DNA strand: agctca

Invalid molecule: a, try again.

Enter molecule string for master DNA strand: AGCTCA

Enter molecule string for comparison or EXIT to quit: exit

Invalid molecule: e, strand ignored.

Enter molecule string for comparison or EXIT to quit: EXIT

DNA Strand: AGCTCA

No match found

Example 3:

Enter molecule string for master DNA strand: TGCCTGCTCG

Enter molecule string for comparison or EXIT to quit: AAAAAAA

Enter molecule string for comparison or EXIT to quit: EXIT

DNA Strand: TGCCTGCTCG

No match found

**Assignment Questions**

Provide the answers to these questions in your submission directory within a file called Assignment3Questions.

1. The assignment mentions there is an efficiency issue with the way we are dissecting the candidate DNA strand into substrings. There is another efficiency issue with this implementation, having to do with redundant calculations. What is that issue and describe how we may resolve it.

2. What is the purpose of the “Rolling Hash”, in the Rabin-Karp algorithm?

3. Create six of your own test case DNA master and candidate strands, some valid, some invalid. What were your test cases and their results?