Lab 3-2: Suffix Trees for Repeat Search and Palindrome Discovery

Due date: May 10 (Thursday).

About the Lab

This is a joint lab with **CHEM 441**. The **CSC 448** teams remain the same as in prior labs. The pairing with the **CHEM 441** students remains the same, unless Dr. Goodman explicitely makes reassignments.

The goal of this lab is to (a) build suffix trees to use for various string-search- and string-matching-related tasks, (b) use suffix trees to discover repeated sequences in DNA fragments, (c) use suffix trees to discover palindromes in DNA fragments. The specifications for the first task are provided in this document. Tasks (b) and (c) are briefly addressed here; full specifications will come from your **CHEM 441** partners.

Time	CSC 448	CHEM 441
April 26 lab	Discuss assignment/map out solution	
May 1 lab	Work on implementation	
May 1 – May 8	Software development	
May 3 lab	Software development	
May 8 lab	Prototype delivery/discussion/use	
May 8 – May 10	Finishing software/ software use	
May 10 lab	Software delivery	Software use

Lab Assignment

As in **Lab 3-2**, your assignment consists of three parts:

1. Development of a library of tools for building, maintaining and using suffix trees.
2. Development of a software tool that uses suffix trees to search for a variety of repeated sequences in the DNA fragments.
3. Development of a software tool that uses suffix trees to search for a variety of palindromes in the DNA fragments.

The parts are briefly discussed below.

Suffix Tree Library

Create a library of tools for suffix tree processing. The library shall contain the following:

1. An implementation of an abstract data type for suffix trees. The core operations are:
   (a) create an empty tree
   (b) create a new leaf node, append it to an existing node in a tree (input: node to append to, edge label, leaf label).
   (c) split an edge by inserting an new internal node.
   (d) insert a single suffix into a given tree (input: suffix, position).
2. Suffix tree creation procedures given an input string (and/or a file in FASTA format).
3. Various *upgrades* to the core suffix tree data structure, that allow for fast repeat detection algorithms and fast palindrome detection algorithms. In particular, at the very least, for repeat detection:
   (a) Leaf nodes should store the left character of the suffix.
   (b) Internal nodes should store left character/left-diversity flag.
   (c) The procedure for labelling the suffix tree with left-diversity information shall be implemented (either as standalone functionality, or as part of the suffix tree construction process)

   For palindrome detection:

   (a) Suffix tree structure shall be extended to accommodate for generalized suffix tree storage (in particular, leaf node labels shall be updated to store a list of `<string, position>` pairs).

(b) `lca()` (lowest common ancestor) and `lce()` (longest common extension) procedures shall be implemented.

4. Implementation of string matching: given a suffix tree and a query string, determine if the query string is found in the tree and report all occurrences (this should work on **any** variant of the suffix tree. On generalized suffix trees the report will contain occurrences in all strings hosted in the suffix tree).

The size of the string input cannot be limited. The actual inputs used in the second part of the assignment will be on the order of tens of thousands of characters. Your code needs to be able to handle strings up to hunderds of thousands of characters long. If traditional ways of managing strings that long in your programming language of choice fail (e.g., if this goes beyond the limitation of, say, Java's `String` data type) you must implement your own abstract data type for management of long strings. You can assume that all data given to your program will fit in main memory, and that your suffix trees will fit in main memory, but beyond that, no other size assumption shall be made.

Write a simple `main` program verifying the functionality of your library.

Search for repeats

As usual, each team will receive the specific instructions for this part of the assignment from their **CHEM 441** partners. The instructions in this document are just a brief general overview.

CHEM 441 students will be working with the contig assemblies constructed using your **Lab 3-1** code (together with the GFF information on the coding regions). There is a number of reasons why *repeated strings* of certain sizes are important to them: repeats identify certain strings that might represent functional DNA elements; repeats in close proximity indicate reduced DNA string complexity, and so on.

Your goal is to build a program that:

1. takes as input a DNA string in FASTA format, and (if necessary) the coding regions information in the GFF format.
2. searches for maximal repeats (and for all repeats) in the string.
3. allows users to specify certain filtering conditions on the repeats.
4. reports all repeats/maximal repeats that match the specified filtering conditions.
5. ouputs (at user option) reported results in a convenient for **CHEM 441** students format.

Repeat search. Searching for repeats and maximal repeats shall be performed using the suffix tree functionality from the library you are creating for this lab assignment.

Filtering. The users of this program may be interested in looking for specific types of repeats. The information that can limit which repeats to report may include the following criteria:

1. Length of repeated sequences. Typical length of repeated sequences may be 6-12 nucleotides. Users may desire repeats of specific length, repeats whose length falls in specific range, and/or repeats whose length exceeds specific length.

2. Proximity of repeated sequences. Users may desire to only see repeated sequences that are relatively close to each other in the DNA. (e.g., seeing a repeat within 100 nucleotides may be desired, but seeing one within 10000 nucleotides may be not interesting).

3. Specific locations of repeated sequences. Users may desire to only see repeated sequences that occur in specific locations of the DNA string (a range of positions, coding/non-coding regions, etc).

4. Specific sequences. Users may desire to only see repeated sequences that contain a specific subsequence (e.g., all repeats that contain `ATG`). Users may also desire to check if a specific sequence is a repeat.

5. Sequence type. Users may desire to see either maximal repeats only, or any repeats.

Note, that there is enough information in the suffix tree structures you will be using for repeat detection to work out every filter among the ones specified above. **CHEM 441** students **may request other types of filters**: it is your job to figure out which specific filtering criteria are of interest to them, and **how** the filtering can be done using the information in the suffix trees. In case of doubt, please consult the instructor - I can work with individual teams on more *exotic requests*.

UI. Due to a potentially large number of filtering criteria, this program must be interactive. In fact, it is my preference, that this program comes with a simple graphical user interface (GUI). Teams working in Java, can utilize the **Lab 2** UI prototype provided by Aldrin Montana — it might have to be extended with more text fields to accommodate all filtering/querying options, but it lends itself nicely to the overall structure of the program. Teams can also design and implement their GUIs from scratch. If a team decides to forego GUI, a justification needs to be provided (e.g., you found a good way of conveying information to the program w/o the GUI, or your **CHEM 441** partners actually **do not want** a GUI).

Search for Palindromes

The second program you will develop for your **CHEM 441** colleagues will utilize suffix tree to search for palindromes in DNA strings.

From the design point of view, this program should be similar in spirit to the repeat search program. It shall take a file in FASTA format and (if needed) a GFF file as input. It shall perform palindrome searches, and filter the palindromes based on a number of criteria outlined below. Finally, it shall output information in a user-friendly way.

Note. If you are developing GUIs for your programs, you can choose to supply two different programs to **CHEM 441** students - one for each of the two tasks. You may also choose to merge the two programs and supply just one executable, in which the functionality is clearly delineated (e.g., by incorporating it into two different tabs on the main screen).

Palindrome search. Searching for palindromes and gapped palindromes shall be performed using the suffix tree functionality from the library you are creating for this lab assignment. In this assignment, you are going to search for DNA palindromes. However, your generic palindrome search implementation should allow for searching for traditional palindromes as well.

Filtering. The users of this program may be interested in looking for specific types of palindromes. The information that can limit which repeats to report may include the following criteria:

1. Length of palindromes. Users may be interested in palindromes of specific lengths - e.g., palindromes whose wings exceed 10 base pairs.
2. Gap Length. Users may be interested in perfect palindromes (i.e., palindromes with the gap size not exceeding 1), or in palindromes with a certain gape size (e.g., 5) or with a gap that falls into a range of sizes (e.g., 5-10 base pairs).
3. Specific locations of palindromes. Users may desire to only see palindromes that occur in specific locations of the DNA string (a range of positions, coding/non-coding regions, etc).

Note, that there is enough information in the suffix tree structures you will be using for palindrome detection to work out every filter among the ones specified above. **CHEM 441** students **may request other types of filters**: it is your job to figure out which specific filtering criteria are of interest to them, and **how** the filtering can be done using the information in the suffix trees. In case of doubt, please consult the instructor - I can work with individual teams on more *exotic requests*.

UI. Same considerations as for the first program (repeat sequence detection) apply.

Submission Instructions

These instructions are for your graded deliverables for **CSC 448**. **CHEM 441** students have their own set of deliverables: they rely on being able to run your software to produce them.

Use handin to submit all your files. Make certain you have a `README` file that accompanies your submission and explains how to set it up and to run it.

Use the following command to submit:

```
$handin dekhtyar 448-lab3-2 <files>
```