A7: Pattern Matching

Overview

You will be given a body of text and asked to find all occurrences of a particular pattern.

Deliverables

- **A7.cpp** will be submitted to the A7 (and A7 Bonus) assignment on Autolab.
    - This will contain your code for all required functions. This file should not contain any main function.
    - This is the only code file to be submitted, so please ensure you contain all your code in here.
    - **Allowed includes: A7.hpp, any C++ container (<array>, <deque>, <forward_list>, <list>, <map>, <queue>, <set>, <stack>, <unordered_map>, <unordered_set>, <vector>), <algorithm>.**
    - As usual, it should compile with the appropriate flags.

Due Date

DUE DATE:

A7.cpp: 5/12/2017, at 11:59PM (this is Friday, May 12 at 11:59PM).

Suggested deadline:

- **Parts 1 and 2 should be completed by the end of the week.**

Start Early!!!!!!!!

Late Policy

The policy for late submissions is as follows:

- Before the deadline: 100% of what you earn (from your best submission).
- One day late: 20-point penalty (lower your earned points by 20, minimum of 0).
- > 2 days late: 0 points.
- You may use up to 2 grace days on this assignment. Submissions after 2 days will not be accepted.

Objectives

In this assignment you will be tasked with finding different patterns in a body of text. This is represented by taking a string of characters and finding all matchings of that character in the body of text.

Useful Resources

Review the lecture notes and provided examples to get an idea for how to use maps, unordered maps, and also check out another type of tree structure called a trie:

- https://en.wikipedia.org/wiki/Trie
- http://www.cplusplus.com/reference/stl/
- http://www.cplusplus.com/reference/algorithm/

Instructions

- Download the file `A7-skeleton.tar` and extract the skeleton files. **Rename the skeleton files appropriately.**
- A `CMakeLists.txt` file is included to properly build your program. You may use this by Importing the folder into CLion, or from the command line by the following commands:

```
cmake .
make
```

- You will have to add the function definitions in `A7.cpp`
- **Allowed includes in A7.cpp: A7.hpp, any C++ container (<array>, <deque>, <forward_list>, <list>, <map>, <queue>, <set>, <stack>, <unordered_map>, <unordered_set>, <vector>), <algorithm>.**
- You may assume that your input sequences are not empty strings, each one containing only characters in the range a, b, c, …, z (all lowercase).
- Efficient code will be necessary to complete the tasks. You are welcome to use any data structures you wish, however, it is suggested you use one we have been discussing recently in lecture.
- You should create a design for how you are planning to solve this problem prior to writing code.

Required:

Complete the functionality for each function. You will lose credit if your code leaks memory or runs inefficiently due to improper storage of data.

Part 1: Find all occurrences of a given sequence

```
/**
 * countOccurrences
 *
 * @param seq: sequence of characters you want to find all matches of.
 * @param text: body of text to find matches in. All characters are a-z.
 *
 * @return: return the count of the number of times seq occurs in text.
*/
unsigned int countOccurrences(const std::string& seq,
                              const std::string& text) {
    // Your code goes here.

}
```

This function will take in a sequence and a body of characters (both will be in the range of a-z, lowercase only). Your task is to count all appearances of the sequence in the text. For example, if seq is "aaa" and

text is "aaabcaaaa" you have 3 occurrences of "aaa" since you have "**aaa**bcaaaa", "aaabc**aaa**a", and "aaabca**aaa**" (the occurrences may overlap).

Part 2: Find all occurrences of a fixed length sequence

```
/**
 * countOccurrences
 *
 * @param text: body of text to find matches in. All characters are a-z.
 *
 * @return: return the occurrence count of each 4 character string
 *                          (including those that are not present).
*/
unsigned int countOccurrences(const std::string& text) {
    // Your code goes here.

}
```

This function will take in a body of characters (in the range of a-z, lowercase only) and return a list giving the occurrence count for each string of length 4. For instance, you may have the text "aaaaab" and you would return a vector with 456,976 entries where the first entry is 2 (2 occurrences of aaaa), the second entry is 1 (1 occurrence of aaab), and the remaining entries are all 0. The ordering in the vector should correspond to:

Index	String
0	aaaa
1	aaab
…	
25	aaaz
26	aaba
27	aabb
…	
51	aabz
52	aaca
53	aacb
…	
675	aazz
676	abaa
…	
17575	azzz
17576	baaa
17577	baab
….	
456974	zzzy
456975	zzzz

The ordering follows the same as dictionary ordering.

Part 3: Find all occurrences of all given strings in a vector of strings.

```
/**
 * countOccurrences
 *
 * @param seqs: vector holding sequences of characters to be matched.
 * @param text: body of text to find matches in. All characters are a-z
 *
 * @return: return a vector containing occurrence counts of each sequence
 *                    (counts should occur in the order sequences are given).
*/
std::vector<unsigned int> countOccurrences(
                              const std::vector<std::string>& seqs,
                              const std::string& text) {
    // Your code goes here.

}
```

This function will take in a list of characters and a body of characters (both in the range of a-z, lowercase only) and return a list giving the occurrence count for each string given. The counts returned should correspond to the indices they appear in the input seqs vector. For example, if you have input

seqs: [a, aa, ab, aab]

text: aaaabab

Then you have:

Occurrences of a: **a**aaabab, a**a**aabab, aa**a**abab, aaa**a**bab, aaaab**a**b (5 in total)
Occurrences of aa: **aa**aabab, a**aa**abab, aa**aa**bab (3 in total)
Occurrences of ab: aaa**ab**ab, aaaab**ab** (2 in total)
Occurrences of aab: aa**aab**ab (1 in total)

So the vector returned would be:
[5, 3, 2, 1]

Bonus:

Find all occurrences of pairs of given strings in a vector of strings.

```
/**
 * countOccurrences
 *
 * @param seqs: vector holding sequences of characters to be matched.
 * @param text: body of text to find matches in. All characters are a-z
 *
 * @return: return a vector containing occurrence counts for
 *          each pair of sequences
 *          (counts should occur in the order sequences are given).
*/
std::vector<unsigned int> countOccurrences(
                              const std::vector<std::string>& seqs,
                              const std::string& text
                              bool bonus) {
    // Your code goes here.

}
```

This function will take in a list of characters and a body of characters (both in the range of a-z, lowercase only) and return a list giving the occurrence count for each string given. The counts returned should correspond to the indices they appear in the input seqs vector. Pairs go in the order of:

(0,0) (0,1), (0,2), … , (0, n-1), (1,0), (1,1), (1,2), … , (1, n-1), … , (n-1,0), … , (n-1, n-1)

where n is the number of sequences input in seqs.

Submission

Code Submission (100 points):

You must submit your **A7.cpp** file to the A7 assignment on Autolab.

- Your submission must compile and run on Autolab. I suggest that you compile often as you build your code to avoid difficulties debugging syntax and other errors.
- Make sure your code does not leak memory.
- **Bonus (20 points):** Include bonus function in your A7.cpp file as well. Submit this to *A7 Bonus* for bonus grading (you may include all code in the same file, but grading will be done separately).

Tests will be run on your file and your score will be reported to you when finished. For this assignment there is no limit on the number of submissions you may perform. That said, you should not be using the submission system to test and debug your code.

DUE DATE:

A7.cpp: 5/12/2017, at 23:59 (this is Friday, May 12, at 11:59PM).

***** Please be aware of this deadline. Autolab will also tell you how long until the deadline.*****