CSE250 Fall 2016
Assignment A1 – K-mer Counting

Due: 09/25/2016, 11:59PM

Last updated: *2016-09-12 21:33*

Objectives

- Practice using of arrays in non-trivial applications.
- Work with functions, control structures, expressions and files.

Introduction

K-mer counting is one of the simplest yet very powerful types of DNA analysis. We can represent any DNA sequence as a string of four letters: `[A, C, G, T]`. For example, this code

```
std::string dna_seq = "ACTTGACTT";
```

stores some DNA sequence in a variable `dna_seq`. A k-mer of size k is a substring of k letters. For a sequence of length l we can enumerate $l - k + 1$ k-mers. For instance, if we consider `dna_seq` and pick $k = 3$ we can enumerate the following 3-mers: `[ACT, CTT, TTG, TGA, GAC, ACT, CTT]`. Sometimes, DNA sequence is corrupted and on some positions `[A, C, G, T]` is replaced by `N`. For example, this code

```
std::string dna_err = "ACTNGACT";
```

stores in `dna_err` a corrupted sequence with one `N` on position 3.

The problem of k-mer counting is posed as follows. Given a DNA sequence and fixed parameter k, report count of each **unique** and **correct** k-mer in the sequence. Here, correct k-mer is a k-mer without `N`. If we again consider `dna_seq` and $k = 3$ then the k-mer counting will give us:

```
ACT   2
CTT   2
TTG   1
TGA   1
GAC   1
```

On the other hand, if we consider `dna_err` we will get:

```
ACT   2
GAC   1
```

Your task in this assignment is to write an efficient program that for a given sequence and parameter k will perform k-mer counting.

Hint

To implement your k-mer counter consider the following observation. Let A=0, C=1, G=2 and T=3 (i.e. think about DNA letters as digits). We can represent each k-mer as a number in base-4 system, which next can be converted to a regular base-10 index. For example, a 3-mer CGA can be represented as 120_4 which is 24 in the decimal system (i.e. 24_{10}). We can use this simple mechanism to assign index to each k-mer and use array to store count of different k-mers. What should be the size of such count array? Notice that for a given k there are 4^k possible correct k-mers. As long as k is small (and this is the case for this assignment) we can easily store count of all k-mers in the main memory.

Instructions

1. Create directory `A1` where you will place your code.
2. Create `Makefile` to automate compilation of your code. Your main source code file should be named `a1.cpp` and it should compile to `a1` executable. You can directly adopt `Makefile` from Assignment 0 (replace `a0` with `a1`).
3. In `a1.cpp` add the following comment header. The comment should be in the very first lines of your file:

   ```
   // File: a1.cpp
   // Author: FirstName LastName UBITName
   ```

   You must provide your correct information in `Author` line. **Do not confuse your UBITName with your UB person number!**
4. Write a program to perform k-mer counting given the following specification:
   (a) Your program must take as a command line argument name of an input file with the input data (in format described below). For example, if invoked like this:

   ```
   ./a1 foo.txt
   ```

   your program should perform k-mer counting using data from `foo.txt`.

   (b) Your program should accept input files in the following format:

   ```
   k n
   s1
   s2
   ...
   sn
   ```

   where `k` represents parameter k, `n` is the number of lines in which sequence is stored, and `s`i, $i = 1 \ldots n$, is a sequence fragment. The format requires that $k \in [3, 10]$, $n > 1$, and all sequence fragments are uppercase. Moreover, the length of the input sequence is greater than k. Below is an example correct file with a sequence over five lines and $k = 3$:

   ```
   3 5
   ACATGGATGACTT
   CTCTTNACTT
   ```

```
ACTAANNAAAATGACTT
ACTTGACTTCGCG
ACTTGACTTTT
```

and example incorrect file:

```
2 2
ACTAAAAAAATGACTT
ACTTGACTTCGCG
ACTTGACTTTT
```

(c) Your program should check if the input file is correct. It is sufficient that you test if k and n match the specification and that there are at least n lines with the actual input sequence (i.e. we are concerned with the first n lines of sequence). You can assume that the entire sequence is **always** uppercase and consists of `[A, C, G, T, N]` **only** (in other words, you do not have to check that).

(d) Your program should print to the standard output a list of all unique and correct k-mers found in the input sequence together with their count. If the input file is incorrect, your program should print to the standard output just one word: `error`. For example, for the following input:

```
3 3
ACTG
ACTG
ACTG
```

your program should print to the standard output:

```
ACT 3
CTG 3
TGA 2
GAC 2
```

and for the input below:

```
3 1
AAA
```

your program should output:

```
error
```

Note that the order in which you output k-mers is irrelevant.

Submission

1. Remove your binary code and other unrelated files (e.g. your test files).
2. Create a tarball with your `A1` folder.
3. Follow to https://autograder.cse.buffalo.edu and submit `A1.tar` for grading.
4. You have unlimited number of submissions, however, any submission after the deadline will have 50% points deducted.

Grading

- 10pt: `a1.cpp` compiles via `make` and runs.
- 90pt: There will be nine test input files. You will get 10pt for each correctly processed file.
- If your code has a memory leak, you will lose half of points.
- If your program is **extremely** inefficient, autograder will terminate your code and you will receive 0pt.

Remarks

- Make sure that all file and directory names are exactly as instructed. Otherwise the grading system will miss your submission and you will get 0pt.