CSE250 Assignment A1 – K-mer Counting

Due: 02/25/2018, 11:59PM

Last updated: 2018-02-12 08:49

Objectives

- Practice 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 counts 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 adapt `Makefile` from Assignment 0 (replace `a0` with `a1`).
3. In `a1.cpp`, add and then update with your first and last name the following comment header. The comment should be in the very first lines of your file:

   ```
   // File: a1.cpp
   // First Name
   // Last Name
   // I AFFIRM THAT WHEN WORKING ON THIS ASSIGNMENT
   // I WILL FOLLOW UB STUDENT CODE OF CONDUCT, AND
   // I WILL NOT VIOLATE ACADEMIC INTEGRITY RULES
   ```

   Make sure that you understand and will respect the affirmation set in the header.
4. Write your program to perform k-mer counting given the following specification:
   (a) Your program cannot use dictionaries like map or hash table.
   (b) Your program must take two command line arguments: name of an input file with the input data (in format described below), and integer size of k-mer (parameter k). For example, if invoked like this:

   ```
   ./a1 foo.txt 3
   ```

   your program should perform 3-mer counting using data from `foo.txt`.
   (c) Your program may assume that all input files, which it will have to process, will consist of one line containing a DNA string to process (i.e. you **do not** have to check if there are more lines, or if the file is empty. The string in the input file will always consist of only `'A'`, `'C'`, `'G'`, `'T'` and `'N'`. In other words, you can assume that the entire sequence is **always** uppercase `[A,C,G,T,N]` (again, you do not have to check if input string is correct).
   (d) You can assume that parameter $k \in [3, 10]$, and it is always passed correctly, i.e. you do not have to test if it is in correct range.
   (e) Your program should print to the standard output the following information. In the first line, it should print `0` followed by k and the length of the input sequence found in the input file. In the second line, it should print `1` followed by the number of As, the number of Cs, the number of Gs, the number of Ts, and the number of Ns in the input sequence. In the following lines, it should list all unique and correct k-mers found in the input sequence together with their count (one k-mer per line). If the input sequence is shorter than k, then instead of listing k-mers, your program should print to the

standard output one word: `error`. For example, suppose that the following sequence is stored in `foo.txt` and your application is invoked as: `./a1 foo.txt 3`

```
ACTGACTGACTG
```

then your program should print to the standard output:

```
0 3 12
1 3 3 3 3 0
ACT 3
CTG 3
TGA 2
GAC 2
```

If `foo.txt` were to store:

```
AA
```

your program should output:

```
0 3 2
1 2 0 0 0 0
error
```

Now suppose that `foo.txt` stores the following sequence:

```
TAN
```

If your program is invoked as `./a1 foo.txt 3` then the output should be:

```
0 3 3
1 1 0 0 1 1
```

and when invoked as `./a1 foo.txt 2` then the output should be:

```
0 2 3
1 1 0 0 1 1
TA 1
```

Final note: you can output k-mers in any order!

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 can make five submissions, and the last submission will be graded
5. Any submission after the deadline will have 50% points deducted.

Grading

- 10pt: `a1.cpp` compiles via `make` and runs.
- 10pt: k and length of the input sequence are reported correctly.
- 10pt: Counts of `ACGTN` are reported correctly.
- 70pt: There will be seven test input files. You will get 10pt for each correctly processed file.

- If your code has a memory leak, you will lose half of the 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.