COMP9319 2020T2 Assignment 2: Searching & Decoding BWT Encoded DNA Sequence

Your task in this assignment is to create two programs: a search program called **bwtsearch** that implements BWT backward search, which can efficiently search a BWT encoded DNA file; and a decoder program called **bwtdecode** that efficiently decodes BWT encoded DNA file back to its original DNA sequence. In addition to correctness, your assignment solution also needs to satisfy some performance requirements.

You can find some sample files available at the folder `~cs9319/a2` (you need to login to a CSE linux machine to check the content of this folder). Each of the original DNA files there (files ended with .txt) contains a sequences of four letters: C, G, A and T (representing four nitrogen-containing nucleobases of a nucleotide: cytosine [C], guanine [G], adenine [A] or thymine [T]) and ended with a newline. For example:

```
wagner % cd ~cs9319/a2
wagner % cat dna-tiny.txt
ACTGACTGACTGACTGACTGAACTTACGTAGTCCAAGTA
wagner %
```

and its corresponding BWT encoded file:

```
wagner % cat dna-tiny.bwt
ATGCTGGGG
AATCTAAAAAAATTTTTACAGTGGCCCCCCwagner %
```

As emphasized in the class, BWT is an algorithm that transforms data to make it more compressible. For example, consider the sample `dna-100MB.txt`:

```
CAAACCTAAGGGTGCCAAACGATACCTTAATGTCGAAGGTTATATTGGGATAAATACCATCATTGGGTCAATTTTCTTTTATTCTCTTTG
GAGTTTTATAAAGGAATAAACAATGCAAAGATAAACTAGAGATCAGATGTGGATCCGAAGATTTAGTATATTAATATGTTTCATAAGTTG
GAATACCAAACCAACTGAACCGTAGATCAAAAGTTTTATTTTACTAAAACTACAAATATTTTTCACATTTTTTATAGAAATTTACCAAGT
AATAGAACATTCCATTAAATTATGGAATATTCTGTCCAAAAGGATAAAGCACTGAAGGTCAGGAGTAAATCAAAATCTTTGAAGCTTTTA
GGCTTACCTATTTGGAAAACACTGTACGCTTCAGATAGTAAAACGGCAAACCTAAATCAAACACACTTATTTATGGAGCCATTTTGTATG
ATGCTAATTTTACAGACTCGAATCTTTTTATCGCCTAGCAGAACCACCTGCAACACCACTTGAATTCACCACAGACTTGTTCATAGTTGG
ACAATCCTATAGATCTGAGGGATGCATCTAATCCATGATCGTTGTCAACCTAATAGACTAGGTGGTTTTGAATAGATTGATATGTCCGAC
GTTTTCAAAGGAGCAAATAACGCGGAATTCAAACCCCCCAGCCCCAAAAAAAATGTTGTGGATCCATGGAGAGAATCGAATATGGAATGG
CTACAGTTACTCTAGATAGGAGTGTCGACATAAGTCTACTTAGGGTTAGAAAAAAAATAAAGTAGCAATACCAAAAGAAAAATAATTGTT
GCAAATGGAATCAACGAGCAAATTCGGTTTTAAGGAATCAACGCATGCAAAATTCAGCTAATTTGGCCTAATGAAATTGCACTTAGCTTA
AGAAAGGTGTCAAGCACAGTATAATCAAGCTGAATCCATTAAATTACAAAATAACGATAATTTGACCAAGAGAAATCTAATTTCTGTATA
TGAACGTTCGATGGCTACTCTTTCCTGAGACCATTTGTTCGCGAGTCTAACAATTTTAGTGAAGATAACTCCCCATAGCCCCTGCATCCT
...
```

If you examine its transformed result, i.e., `dna-100MB.bwt`:

```
GTAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGAAAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
...
```

The transformed file can be easily compressed via RLE and/or MTF with a very good compression ratio. While BWT backward search can still be run against these compressed files (whereas, other usual search tools, such as `grep`, are incapable to do so), for simplicity, you are only required to search on a uncompressed BWT encoded file (such as those ended with .bwt in the sample file folder).

During the marking of your programs, only BWT encoded files are provided. In the sample file folder, the original DNA files (those ended with .txt) are provided for your reference and testing only. These files will not be available during marking. Furthermore, to easily detect any hardcoded solutions, your submitted solution may be tested with a different set of BWT encoded DNA files.

Your **bwtdecode** accepts two commandline arguments: the path to the input file (a BWT encoded file); and the path to the output file. The input filename is a path to the given BWT encoded file. Please open the file as read-only in case you do not have the write permission. When testing your program, we will ensure that the output file does not exist so that your program will need to create this file. For example:

```
bwtdecode ~/MyAssignmnt2/dna-tiny.bwt myData/TinyDNA-output.result
```

You can test and check the correctness of your decoder easily by comparing your output file with the corresponding, provided source DNA files. For example:

```
wagner % ./bwtdecode ~cs9319/a2/dna-tiny.bwt myoutput.txt
wagner % diff myoutput.txt ~cs9319/a2/dna-tiny.txt
wagner %
```

Your **bwtsearch** accepts one commandline argument: the path to the input file (a BWT encoded file). It will then take a sequence of search terms (one line each) from the standard input, performs a BWT backward search using the search term after each line of input, and output the number of matches (including zero) to the standard output. It can be input via the terminal and terminated by a Ctrl-D ("^D"). For example:

```
wagner % ./bwtsearch ~cs9319/a2/dna-tiny.bwt
A
12
TA
3
TAG
1
GA
5
ACT
6
ACTG
5
ACTGA
5
ACTGAC
4
AA
2
AAA
0
wagner %
```

where the lines containing A, C, G, T are the input, and the lines containing numbers (i.e., the number of matches) are the output.

Alternatively, we can also run `bwtsearch` with file redirecting. For example, consider a simple sanity test that is available in the sample file folder:

```
wagner % cat ~cs9319/a2/dna-tiny.input
A
TA
TAG
GA
ACT
ACTG
ACTGA
ACTGAC
AA
AAA
wagner % cat ~cs9319/a2/dna-tiny.output
12
3
1
5
6
5
5
4
2
0
wagner %
wagner % ./bwtsearch ~cs9319/a2/dna-tiny.bwt < ~cs9319/a2/dna-tiny.input > myoutput.txt
wagner % diff myoutput.txt ~cs9319/a2/dna-tiny.output
wagner %
```

To avoid any hiccups during auto marking, when you finish your assignment, **you should test your solution with the sanity test** above (`dna-tiny.input` and `dna-tiny.output`) just to ensure that the formatting (such as newlines) of your input and output is correct.

Your assignment solution will be auto-marked. Each `bwtsearch` on a BWT file will contain at least 10 search terms (i.e., at least 10 lines of search input) but no more than 20 search terms, so the runtime cost of building the backward search data structure is relatively less dominated.

You can verify the correctness of the number of matches for your solution using a baseline implementation of `bwtsearch` provided at `~cs9319/a2` as follows:

```
wagner % ~cs9319/a2/bwtsearch ~cs9319/a2/dna-tiny.bwt < ~cs9319/a2/dna-tiny.input > myoutput.txt
wagner % diff myoutput.txt ~cs9319/a2/dna-tiny.output
wagner %
```

Similarly, a baseline implementation of `bwtdecode` is also provided at `~cs9319/a2`, i.e., `~cs9319/a2/bwtdecode` To make the assignment simple, you can assume that: (1) the search is case sensitive (don't need to worry about lowercase); (2) a search term will not be an empty string; (3) a search term will not contain characters other than A, C, G and T; (4) a search term will not be longer than 100 characters.

However, marks will be deducted for output of any extra text, other than the required, correct answers. This extra information includes (but not limited to) debugging messages, line numbers and so on.

Your solution is **not allowed** to write out any (even temporary) files, except the output file required (as specified in this document). Otherwise, you will receive zero points for the entire assignment.

Although you do not need to submit a BWT encoder, if you wish, you may implement a simple BWT encoding program (this will help you in understanding the lecture materials and assist in testing your assignment, though you are provided with a reasonable number of sample files available at the sample file folder).

Performance Requirements

Your solution will be marked based on space and runtime performance. Your bwtsearch will not be tested against any BWT encoded DNA files that are larger than 100MB, and your bwtdecode will not be tested against any BWT encoded DNA files that are larger than 15MB. At least half of the test cases will be based on BWT files that are smaller than 10MB (FYI, in case if you struggle with the large files).

Total runtime memory of each of the two programs (including the program footprint size) is assumed to be always less than 15MB. Runtime memory consumption will be measured by `valgrind massif` with the option `--pages-as-heap=yes`, i.e., all the memory used by your program will be measured. We will also manually check your code to ensure that your program does not try to allocate memory in a way to avoid the valgrind measurement. Any solution that violates this memory requirement will receive zero points for any relevant tests.

You may use `ms_print` to help view your massif output. For example, you can check the usage of all the memory usage snapshots of your program by reviewing the final file output from ms_print, i.e., memory.txt, as shown in the example below:

```
valgrind --tool=massif --pages-as-heap=yes --massif-out-file=memory.out ./bwtsearch ~/a2/dna-100MB.bwt < mytest.in
ms_print memory.out > memory.txt
```

To measure the time performance of your solution, we will use the `time` command (specifically, `/usr/bin/time`) to count both the user and system time. The time limit to complete a test will be at least 1 second (by adding 1 second to the test performed by the baseline) for bwtdecode and at least 5 second for each whole bwtsearch test (i.e., a test with a given BWT file and a redirecting input file that contains 10–20 search terms), as described in detail below.

For decoding a BWT file, any solution that runs slower than the baseline `bwtdecode` (plus one extra second) on a machine with similar specification as *wagner* will be killed, and will receive zero points for the decoding test of that BWT file. For example, consider the user and sys time of the baseline bwtdecode on three different BWT files:

```
wagner % /usr/bin/time -p ~/a2/bwtdecode ~/a2/dna-5KB.bwt my5KB.txt
real 0.00
user 0.00
sys 0.00
wagner % /usr/bin/time -p ~/a2/bwtdecode ~/a2/dna-1MB.bwt my1MB.txt
real 0.48
user 0.47
sys 0.00
```

```
wagner % /usr/bin/time -p ~/a2/bwtdecode ~/a2/dna-medium.bwt myMedium.txt
real 3.66
user 3.43
sys 0.01
```

Then your decoder should also be able to successfully and correctly decode the above three files within 1.00s, 1.47s and 4.44s, respectively. The above numbers may be different when the programs are run on a different machine).

For `bwtsearch` on a given BWT file and a test input file that contains 10-20 search terms, any solution that runs slower than the baseline `bwtsearch` (plus five extra seconds) on a machine with similar specification as *wagner* will be killed, and will receive zero points for the bwtsearch test of that BWT file. For example, consider the baseline `bwtsearch` on the 100MB BWT file:

```
wagner % /usr/bin/time -p ~/a2/bwtsearch ~/a2/dna-100MB.bwt < mytest.in > mytest.output
real 0.14
user 0.11
sys 0.02
wagner %
```

Your `bwtsearch` will have up to (user + sys time + extra 5s) 5.13s to complete the same test above. Another example using the provided dna-tiny.input file:

```
wagner % /usr/bin/time -p ~/a2/bwtsearch ~/a2/dna-tiny.bwt < ~/a2/dna-tiny.input > test.output
real 0.00
user 0.00
sys 0.00
wagner %
```

Therefore, your `bwtsearch` will have up to 5.00s to complete the test.

Documentation and Code Readability

We assume a reasonable quality of your code readability and documentation in your submitted code. Your source code will be inspected. Marks may be deducted if your code is difficult to read and/or understand.

Compile

Your may complete your assignment in C or C++. You will also need to provide a makefile to compile your source files and generate the executable programs (bwtsearch and bwtdecode) running on a typical CSE Linux machine. In case if you have never created a makefile before, you can reuse and modify your assignment 1 makefile accordingly. You may include and use any C or C++ libraries available in CSE linux machines.

Your solution will be compiled, run and tested on a typical CSE machine e.g. wagner. We will use the commands below to compile your solution. Please ensure that the code you submit can be compiled.

```
    make
```

Any solution that has compilation errors will receive zero points for the entire assignment.

Bonus

Bonus marks (worth 10 percent of the assignment total score) will be awarded for the solution that passes all the tests (correct; and within the memory and time limits) and runs the fastest overall (i.e., the shortest total time to finish **all** the tests). Note: regardless of the bonus marks you receive in this assignment, the maximum final mark for the subject is capped at 100.

Submission and Marking

Deadline: (Week 9) Monday 27th July 09:00am. Late submissions will attract a 1% penalty of the total mark achievable for this assignment per hour after the deadline (i.e., 24% per day), and no submissions will be accepted after 3 days late. Use the give command below to submit the assignment:

```
    give cs9319 a2 makefile *.h *.c *.cpp
```

Or submit via WebCMS: Login to Course Web Site > Assignments > Assignment 2 > Assignment 2 Specification > Make Submission > upload required files > [Submit]

This assignment is worth 35 points, and will be auto-marked. Your code will also be checked manually for readability and plagiarism.

Plagiarism

The work you submit must be your own work. Submission of work partially or completely derived from any other person or jointly written with any other person is not permitted. The penalties for such an offence may include negative marks, automatic failure of the course and possibly other academic discipline. Assignment submissions will be examined both automatically and manually for such submissions. Please refer to Student Conduct/Plagiarism section of the Course Outline (at WebCMS) for details.