COMP9319 2021T2 Assignment 2: Searching & Decoding BWT Encoded File

Your task in this assignment is to create a simple search program that implements BWT backward search, which can efficiently search a BWT encoded file. The program also has the capability to decode the BWT encoded file back to its original file in a lossless manner. The original file (before BWT) contains text in multiple lines and may include characters with ASCII values 10 and 32 to 126 (inclusively). There will be at least one newline character just before the end of file (to make sure that every line of text is ended with a newline character). Each line begins with a line number. All line numbers of a file have the same number of digits (packed with leading zeros as appropriate).

Sample input files (BWT encoded) have been provided in the folder `~cs9319/a2`. To help in testing your program, the corresponding original files (after reverse BWT) are also included in that folder. These original files will not be available when we test your submission.

Although you do not need to submit a BWT encoder, it is recommended that you will implement a simple BWT encoding program (this will help you in understanding the lecture materials and assist in testing your assignment).

Your C/C++ program, called **bwtsearch**, accepts an optional flag; the path to a BWT encoded file; the path to an index file; and **one** quoted, non-empty query string as commandline input arguments. Using the given query string, it will perform a backward search on the given BWT encoded file and output all the lines that contain the input query string to the standard output. The search results are sorted according to their line numbers in ascending order. You may assume that the query string does not contain any newline characters.

Since each line is ended with a newline character, your output will naturally be displayed as one line (ending with a '\n') for each match. No line will be output more than once, i.e., if there are multiple matches in one line, that line will only be output once.

If an optional flag `-m` is specified, given a query string, **bwtsearch** will output the total number of matching substrings (count duplicates) to the standard output. The output is the total number, with an ending newline character. Similarly, **bwtsearch** will output the total number of unique matching records (do not count duplicates) if `-n` is specified. If `-o` is specified, no query string is required in the commandline. Instead, a output filename needs to be specified, a reverse BWT is performed and the original file (before BWT) is output (created, or overwritten if already exists) to the specified output file.

Your solution can write out **one** external index file at any time that is no larger than half of the size of the given input BWT file, plus an extra 2048 bytes. If your index file is larger than half of the size of the input BWT file plus 2048 bytes, you will receive zero points for the tests that use that file. You may assume that the index file will not be deleted during all the tests for a given BWT file, and all the test BWT files are uniquely named. Therefore, to save time, you only need to generate the index file when it does not exist yet.

Example

Consider one of the given sample files `tiny.bwt`. You can find the file by logging into CSE machines and going to folder ~cs9319/a2. The original file (tiny.txt) before BWT is also provided in that folder. The following are some examples using tiny.bwt.

```
%wagner> bwtsearch -m ~cs9319/a2/tiny.bwt ./tiny.idx "ana"
2
%wagner> bwtsearch -n ~cs9319/a2/tiny.bwt ./tiny.idx "ana"
1
%wagner> bwtsearch ~cs9319/a2/tiny.bwt ./tiny.idx "ana"
01 banana
%wagner> bwtsearch -m ~cs9319/a2/tiny.bwt ./tiny.idx "erry"
3
%wagner> bwtsearch -n ~cs9319/a2/tiny.bwt ./tiny.idx "erry"
3
%wagner> bwtsearch ~cs9319/a2/tiny.bwt ./tiny.idx "erry"
10 raspberry
11 cherry
13 blackberry
%wagner> bwtsearch -o ~cs9319/a2/tiny.bwt ./tiny.idx tiny.decoded.txt
%wagner> diff ~cs9319/a2/tiny.txt tiny.decoded.txt
%wagner>
```

More examples:

```
%wagner> bwtsearch -n ~cs9319/a2/6MB.bwt ./6MB.idx "compilers"
1
%wagner> bwtsearch -m ~cs9319/a2/6MB.bwt ./6MB.idx "compilers"
1
%wagner> bwtsearch -m ~cs9319/a2/6MB.bwt ./6MB.idx "compiler"
8
%wagner> bwtsearch -n ~cs9319/a2/6MB.bwt ./6MB.idx "compiler"
```

```
8
%wagner> bwtsearch ~cs9319/a2/6MB.bwt ./6MB.idx "compiler"
020266 Verification of the C0 compiler implementation on the source code level.
021834 A framework for intelligent speculative compiler optimizations and its application to memory accesses.
035239 A compiler backend for generic programming with arrays.
044877 C compiler aided design of application specific instruction set processors using the machine description language LISA.
052175 Semantics-directed generation of compilers and abstract machines
123034 Evaluating compiler technology for control-flow optimizations for multimedia extension architectures.
123777 Design and implementation of a queue compiler.
125902 Hardware-compiler co-design for adjustable data power savings.
%wagner>
%wagner> bwtsearch ~cs9319/a2/sherlock.bwt ./SL.index ",000 pounds"
002001 the stake will be some 30,000 pounds; and for you, Jones, it will
004503 of some 14,000 pounds, which lay to his credit at the bank."
010537 50,000 pounds at once. I could, of course, borrow so trifling a
%wagner> bwtsearch -m ~cs9319/a2/sherlock.bwt ./SL.index ",000 pounds"
3
%wagner> bwtsearch -n ~cs9319/a2/sherlock.bwt ./SL.index ",000 pounds"
3
%wagner>
```

We will use the `make` command below to compile your solution. Please provide a makefile and ensure that the code you submit can be compiled. Solutions that have compilation errors will receive zero points for the entire assignment.

```
make
```

Your solution will be compiled and run on a typical CSE Linux machine e.g. grieg. Your solution should **not** write out any external files other than the index file (and the output, decoded file when "-o" is specified). Any solution that writes out external files other than the index file (and the ouput decoded file as required) will receive zero points for the entire assignment.

Performance

Your solution will be marked based on space and runtime performance. Your soluton will not be tested against any BWT encoded files that are larger than 100MB.

Runtime memory is assumed to be always less than **16MB**. Any solution that violates this memory requirement will receive zero points for that query test. 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. Your code may be manually inspected to check if memory is allocated in a way that avoids the valgrind detection and exceeds the above limit.

Any solution that runs for more than **60 seconds** on a machine with similar specification as *grieg* for the first query on a given BWT file will be killed, and will receive zero points for the queries for that BWT file. After that, any solution that runs for more than **15 seconds** for any one of the subsequent queries on that BWT file will be killed, and will receive zero points for that query test. We will use the `time` command (i.e., `/usr/bin/time`) and count both the user and system time as runtime measurement.

When testing for reverse BWT (i.e., decoding the BWT encoded file back to the original text file), you may assume that at least one query will be run before hand for a given test file, and you will have up to **90 seconds** before your running process is killed. For decoding, the maximum file size during auto marking will be approximately 32MB.

Documentation

Although the assignment is based on auto marking, your source code may be inspected and marks may be deducted if your code has poor readability and is very difficult to understand.

Notes

1. Same as Assignment 1, it does not matter if your program needs to be executed as `./bwtsearch` instead of `bwtsearch`.
2. To avoid a long output time, none of the testcases for marking will result in outputting more than 500 lines.
3. A line in the original file will not be unreasonably long, e.g., you can safely assume that a line is always less than 500 chars.
4. 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 write permissions.
5. Marks will be deducted for the output of any extra text, other than the required correct answers (in the right order). This extra information includes (but not limited to) debugging messages, line numbers and so on. A sanity testscript based on some examples in this spec is available (so you can check that your output format is correct). You can run the sanity testscript by:

   ```
   %wagner> ~cs9319/a2/sanity
   ```

```
Running bwtsearch on tiny.bwt...
Search 1 passed
Search 2 passed
Search 3 passed
Search 4 failed
Search 5 passed
Search 6 passed
Search 7 failed
%wagner>
%wagner> cat ~cs9319/a2/sanity
...
```

You can check the detail of each test by the cat command as shown above.

6. You can assume that every line (including the last line) of the original file corresponding to the input BWT file ends with a newline char.
7. Empty lines may exist in the original files (before BWT). However, these lines will never be matched during searching because the empty string will not be used when testing your program.
8. You can also assume that, during marking, search terms are chosen such that there will be at least one match.

Marking

This assignment is worth 100 points (later scaled to 35% of the total for the course). Below is an indicative marking scheme:

Component	Points
Auto marking (small test files up to 7MB)	60
Auto marking (large test files up to 100MB)	40

Bonus

Bonus marks of 10 points will be awarded for the solution that achieves 100 points and runs the fastest overall for searching (i.e., the shortest total time to finish **all** the backward search tests excluding those tests that index files are expected to be generated). In addition, bonus marks of 5 points will be awarded for the solution that achieves 100 points and runs the fastest overall for generating index (i.e., the shortest total time to finish **all** the backward search tests in which index files are expected to be generated). Note: regardless of the bonus marks you receive in this assignment, the maximum final mark for the subject is capped at 100.

Submission

Deadline: Friday 30th July 17:00 (AEST). Late submissions will have marks deducted from the maximum achievable mark at the rate of 1% of the total mark per hour that they are late (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 *.c *.cpp *.h
```

Note that the give command is available on any CSE linux machines (such as wagner) but not on grieg. Finally, please use classrun to check your submission to make sure that you have submitted all the necessary files.

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.

Relevant scholarship authorities will be informed if students holding scholarships are involved in an incident of plagiarism or other misconduct.

Do not provide or show your assignment work to any other person - apart from the teaching staff of this subject. If you knowingly provide or show your assignment work to another person for any reason, and work derived from it is submitted you may be penalized, even if the work was submitted without your knowledge or consent. This may apply even if your work is submitted by a third party unknown to you.