COMP9319 2022T2 Assignment 2: Compressed BWT Backward Search

Your task in this assignment is to create a search program that implements BWT backward search, which can efficiently search a run-length encoded and BWT transformed (RLB) record file without decoding the file back to a larger form. The o plain text file (before BWT) format is:

```
[<offset1>]<text1>[<off fset3>]<text3>... ...
```

where <offset1>, <offset2 re integer values that are used as unique record identifiers (increasing, not necessarily integers);
and <text1>, <text2>, <te ues, which include any visible ASCII alphabets (i.e., any character with ASCII value sively), tab (ASCII 9) and newline (ASCII 10 and 13). For simplicity, there will be no open or close square bracket in the text values.

After the above text file is BWT transformed, it will be run-length encoded to save storage space. Since the BWT file only includes ASCII characters up to ASCII value 126. The first bit of a byte is used to determine if a byte represents an ASCII character, or if it is part of a number that represents the count. That is, to differentiate a count from a character, a count will have the first bit on, and use the remaining 7 bits to represent the actual value. When a number is larger than 2^7-1, you will need to consider any subsequent "count" byte(s) to form the final value. Furthermore, to minimize the size of a count, 3 runs are represented as 0 (that is, 3 is the minimum number used to represent runs using count). An example in the next section (under 'Run-length Examples') will illustrate this run-length encoding scheme in detail.

Your C/C++ program, called **bwtsearch**, accepts a command argument of either `-n` for the number of matching substrings (count duplicates) or `-a` for listing the identifiers of all the matching records (no duplicates and in ascending order); the path to a RLB encoded file; the path to an index file; and a quoted query string (i.e., the search term) as commandline input arguments. The search term can be up to 512 characters. To make the assignment easier, we assume that the search is case sensitive.

If `-a` is specified, using the given query string, bwtsearch will perform backward search on the given RLB encoded file, and output the sorted and unique identifiers (no duplicates) of all the records that contain the input query string to the standard output. Each identifier is enclosed in a pair of square brackets, one line (ending with a '\n') for each match. If `-n` 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.

Your solution is allowed to write out **one** external index file that is no larger than the size of the given, input RLB file. If your index file is larger than the size of the input RLB file, you will receive zero points for the tests that generating/using that file. You may assume that the index file will not be deleted during all the tests for a given RLB file, and all the test RLB files are uniquely named. Therefore, to save time, you only need to generate the index file when it does not exist yet.

Examples

Usage Examples

Suppose the original file (dummy.txt) before BWT is:

```
[3]Computers in industry[25]Data compression[33]Integration[40]Big data indexing
```

Some examples:

```
%grieg> bwtsearch -n ~cs9319/a2/dummy.rlb ~/dummy.idx "in"
4
```

```
%grieg>
%grieg>
%grieg> bwtsearch -a ~cs9319/a2/dummy.rlb ~/dummy.idx "in"
[3]
[40]
%grieg>
%grieg> bwtsearch -n ~cs9319/a2/dummy.rlb dummy.idx "in"
1
%grieg>
%grieg> bwtsearch -a ~c                    b dummy.idx "In"
[33]
%grieg>
```


You can find the above dum sample files by logging into CSE machines and going to folder ~cs9319/a2. Note tha l BWT files (i.e., .txt and .bwt files) are provided there for your reference only. Your sc ume the availability of these files during the assignment marking. You will only be p coded files (.rlb files).

Testing Examples

To assist you to test the correctness of your program easily, two script programs (called `count` and `fetch`) have been made and are available in ~cs9319/a2. Instead of taking a RLB file as input, they take the original text file as input and then produce the output that a correct assignment solution is expected to produce. During marking, we will use the `diff` command to determine if your solution output is different from the expected output, as shown in the examples below. Note that the script programs are memory based and may not scale to 10MB+ testcases, but they will still help you to establish your solution correctness by starting with files of small or medium size.


```
cs9319@grieg:~$
cs9319@grieg:~$ ~cs9319/a2/count
Usage: /import/kamen/A/cs9319/a2/count FILE.txt PATTERN
cs9319@grieg:~$ ~cs9319/a2/count ~cs9319/a2/dummy.txt "in"
4
cs9319@grieg:~$ ~cs9319/a2/fetch ~cs9319/a2/dummy.txt "in"
[3]
[40]
cs9319@grieg:~$ ~cs9319/a2/count ~cs9319/a2/dummy.txt "in" > 1.ans
cs9319@grieg:~$ ~cs9319/a2/fetch ~cs9319/a2/dummy.txt "in" > 2.ans
cs9319@grieg:~$
cs9319@grieg:~$ bwtsearch -n ~cs9319/a2/dummy.rlb ~/dummy.idx "in" > 1.output
cs9319@grieg:~$ bwtsearch -a ~cs9319/a2/dummy.rlb ~/dummy.idx "in" > 2.output
cs9319@grieg:~$
cs9319@grieg:~$ diff 1.ans 1.output
cs9319@grieg:~$ diff 2.ans 2.output
cs9319@grieg:~$
cs9319@grieg:~$ ~cs9319/a2/count ~cs9319/a2/medium.txt "data"
248
cs9319@grieg:~$
```


Run-length Examples

The folder ~cs9319/a2 also includes some examples to illustrate the run-length encoding scheme used in this assignment.

```
cs9319@grieg:~$ cd ~cs9319/a2
cs9319@grieg:~/a2$ more abcde.bwt
aAAbbbBBBBccccccCCCCCCdDDeeeEEEE
cs9319@grieg:~/a2$
cs9319@grieg:~/a2$ xxd -b abcde.rlb
00000000: 01100001 01000001 01000001 01100010 10000000 01000010  aAAb.B
00000006: 10000001 01100011 10000010 01000011 10000011 01100100  .c.C.d
0000000c: 01000100 01000100 01100101 10000000 01000101 10000001  DDe.E.
cs9319@grieg:~/a2$
```

As shown in the above xxd output, the first bit of a byte is used to determine if a byte represents an ASCII character or a count, i.e., a count will have the first bit on. Therefore, the above xxd output can be summarized as:

```
aAAb{0}B{1}c{2}C{3}dDDe{0}E{1}
```

where the value inside a pair of brackets represents its corresponding count observed from the xxd output. As mentioned earlier, 3 is the minimum number used to represent runs. You can observe from the above example why this will lead to the mi t.

Another example is to illust s, i.e., when a count is larger than 2^7. Consider two BWT files a150 and a20k (contain 's, respectively) in ~cs9319/a2

```
cs9319@grieg:~/a2$ xxd
00000000: 01100001 1001                                a..
cs9319@grieg:~/a2$ xxd
00000000: 01100001 10011101 10011100 10000001         a...
cs9319@grieg:~/a2$
```

Since 3 (3 runs) is the minimum number and is represented as 0 count, 150 is represented as 147 (in binary 10010011). As shown above, a large count will be divided into 7-bit blocks by starting from the least significant bit. Therefore, the last 7 bits representing 147 (10010011) will be captured by the first byte of the count (10010011), and the remaining bit representing 147 (10010011) will be captured in the subsequent byte (10000001).

Similarly, 20,000 is represented as 19,997 (in binary 1 0011100 0011101) and it will be divided into 3 bytes accordingly: 10011101 10011100 10000001.

Remarks

1. It does not matter if your program needs to be executed as `./bwtsearch` instead of `bwtsearch`.
2. To avoid large runtime memory for sorting, none of the testcases for marking will result in more than 5,000 matches when -a is specified.
3. The input filename is a path to the given RLB encoded file. Please open the file as read-only in case you do not have the write permission (e.g., those files located in ~cs9319/a2)
4. Marks will be deducted for 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.
5. You can assume that the input query string will not be an empty string (i.e., ""). Furthermore, search terms containing only numbers, search terms containing any square bracket, search terms containing dash dash (i.e., "--"), or search terms resulting in no matches will not be tested.
6. You may assume that offset >= 0 and will fit in an `unsigned int`.
7. When counting the number of substring matches (i.e., with -n option), to make it easier for your implementation of backward search matching, all combinations of matches should be counted, e.g., 2 matches of "ana" on "banana".
8. You are allowed to use one external index file to enhance the performance of your solution. However, if you believe that your solution is fast enough without using index file, you do not have to generate the file. Even in such a case, your solution should still accept a path to the index file as one of the input arguments as specified.
9. A record (in the original record/text file before BWT) will not be unreasonably long, e.g., you will not see a text value that is 5,000+ chars long.
10. Empty records may exist in the original files (before BWT). However, these records will never be matched during searching because the empty string will not be used as a search term when testing your program.
11. Your source code may be inspected. Marks may be deducted if your code is very poor on readability and ease of understanding.

Marking & Performance

This assignment is worth 35 points, all based on auto marking.

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 on `grieg`, a CSE Linux machine. Solutions that have compilation errors will receive zero points for the entire assignment.

```
make
```

Your solution should **not** wr iles other than the index file. Any solution that writes out external files other than the e zero points for the entire assignment.

Your solution will be compi eg.

In addition to the output correctness, your solution will also be marked based on space and runtime performance. Your soluton will not be tested against any RLB encoded files generated from source text files that are larger than 160MB.

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 `grieg` for the first query on a given RLB file will be killed, and will receive zero points for the queries for that RLB file. After that, any solution that runs for more than **10 seconds** for any one of the subsequent queries on that RLB 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 take the sum of the user and system time as runtime measurement.

Bonus

Bonus marks (up to 3.5 points, i.e., 10 percent) will be awarded for the solution that achieves 35 points (i.e., full marks) 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

Deadline: Tuesday 26th July 12:00pm (noon). The penalty for late submission of assignments will be 5% (of the worth of the assignment) subtracted from the raw mark per day of being late. In other words, earned marks will be lost. **No assignments will be accepted later than 5 days after the deadline.**

Use the give command below to submit the assignment:

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

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

```
9319 classrun -check a2
```

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 read the Student Con urse outline for details.