COMP9319 2023T2 Assignment 2: BWT Backward Search (Run-length Encoded)

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 original record file as plain text file (before BWT) format is:

```
[<offset1>]<text1>[<offset2>]<text2>[<offset3>]<text3>... ...
```

where `<offset1>`, `<offset2>`, `<offset3>`, etc. are integer values that are used as unique record identifiers (increasing and consecutive, positive integers, not necessarily starting from 0 or 1);
and `<text1>`, `<text2>`, `<text3>`, etc. are text values, which include any visible ASCII alphabets (i.e., any character with ASCII value from 32 to 126 inclusively), 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, 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 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. Using the given query string, it will perform backward search on the given BWT encoded file, and output all the records that contain the query string to the standard output. To make the assignment easier, we assume that the search is case sensitive.

The output is sorted by the record identifiers in ascending order (according to their integer values) and contains no duplicates. Each match consists of a record identifier enclosed in a pair of sqaure brackets, then its text value, and is finally ended with a newline charactor ('\n').

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 this size limitation, 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:

```
[8]Computers in industry[9]Data compression[10]Integration[11]Big data indexing
```

(Note that you will not be given the original file. You will only be provided with the run-length encoded BWT file.)

The command:

```
%grieg> bwtsearch ~cs9319/a2/dummy.rlb ~MyAccount/XYZ/dummy.idx "in"
```

should result in the following:

```
%grieg> bwtsearch ~cs9319/a2/dummy.rlb ~MyAccount/XYZ/dummy.idx "in"
[8]Computers in industry
[11]Big data indexing
%grieg>
```

Another example:

```
%grieg> bwtsearch ~cs9319/a2/dummy.rlb ~MyAccount/XYZ/dummy.idx "in "
```

The output is shown as below:

```
%grieg> bwtsearch ~cs9319/a2/dummy.rlb ~MyAccount/XYZ/dummy.idx "in "
[8]Computers in industry
%grieg>
```

You can find the above dummy.rlb file plus some sample files by logging into CSE machines and going to folder ~cs9319/a2. Note that the original text and BWT files (i.e., .txt and .bwt files) are provided there for your reference only. Your solution should not assume the availability of these files during the assignment marking. You will only be provided with RLB encoded files (.rlb files).

A simple sanity test script is also available in ~cs9319/a2. You should run the sanity test script before you submit your solution. To run the sanity test script on grieg, simply go inside the folder that contains your makefile and program source files and type: ~cs9319/a2/autotest that will run tests based on small example RLB files provided there. It does not test on larger files provided there, as it may take much longer time for your program to complete those tests. However, you can easily construct more tests on larger files by using a provided search program called bsearch (see below for further information) to produce expected search results for comparisons.

Run-length Examples

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

```
cs9319@grieg:~$ cd ~cs9319/a2/helper
cs9319@grieg:helper$ more abcde.bwt
aAAbbbBBBBccccccCCCCCCdDDeeeEEEE
cs9319@grieg:helper$
cs9319@grieg:helper$ 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:helper$
```

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 minimum size of a count.

Another example is to illustrate multi-byte counts, i.e., when a count is larger than 2^7. Consider two BWT files a150 and a20k (contain 150 a's and 20,000 a's, respectively) in ~cs9319/a2

```
cs9319@grieg:helper$ xxd -b a150.rlb
00000000: 01100001 10010011 10000001                              a..
cs9319@grieg:helper$ xxd -b a20k.rlb
00000000: 01100001 10011101 10011100 10000001                     a...
cs9319@grieg:helper$
```

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.

bsearch, a sample makefile and C program

The folder ~cs9319/a2/helper also includes a search program called bsearch, a sample makefile and two C program files. Please read README.txt there for further details before you use them.

Remarks

1. It does not matter if your program needs to be executed as `./bwtsearch` instead of `bwtsearch`.
2. None of the testcases for marking will result in more than 10,000 matches in its output.
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. 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.
8. 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.
9. 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.

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

```
make
```

Your solution should **not** write out any external files other than the index file. Any solution that writes out external files other than the index file will receive zero points for the entire assignment.

Your solution will be compiled and tested on `grieg`.

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 **13MB** (i.e., 13,631,488 bytes). 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: Monday 24th July 12:00pm AEST (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.** Please read the Assignments section in the course outline for details.

Use the give command below to submit the assignment or submit via WebCMS3:

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

Please use classrun to make sure that you have submitted the correct file(s).

```
9319 classrun -check a2
```

Note: Unfortunately the give and classrun commands are not available on grieg, but are on any other CSE linux machine, so you'll have to use these other machines to run the give/classrun command.

Plagiarism

The work you submit must be your own work. Submission of work partially or completely derived from any other person or sources 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 Conduct section in the course outline for details.