# COMP9319 2024T2 Assignment 2: BWT Backward Search

Your task in this assignment is to create a search program that implements BWT backward search, which can efficiently search a BWT transformed record file without decoding the file back to its original 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.

Your C/C++ program, called **bwtsearch**, accepts the path to a BWT encoded file; the path to an index file; and **one** to **five** quoted query strings (i.e., search terms) as commandline input arguments. Each search term can be up to 64 characters. Using the given query strings, it will perform backward search on the given BWT encoded file, and output all the records that contain ALL input query strings (i.e., a boolean AND search). 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 half of the size** of the given, input BWT file. If your index file is larger than half of the size of the input BWT file, you will receive zero points for the tests that using 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.

## 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 BWT encoded file during auto marking.)

The command:

```
%db-perftest> bwtsearch ~cs9319/a2/dummy.bwt ~MyAccount/XYZ/dummy.idx "in"
```

should result in the following:

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

Another example:

```
%db-perftest> bwtsearch ~cs9319/a2/dummy.bwt ~MyAccount/XYZ/dummy.idx "in "
```

The output is shown as below:

```
%db-perftest> bwtsearch ~cs9319/a2/dummy.bwt ~MyAccount/XYZ/dummy.idx "in "
[8]Computers in industry
%db-perftest>
```

Here is an example with multiple search terms:

```
%db-perftest> bwtsearch ~cs9319/a2/dummy.bwt ~MyAccount/XYZ/dummy.idx "in" "ata" "ing"
[11]Big data indexing
%db-perftest>
```

You can find the above dummy.bwt file plus more sample files by logging into CSE machines and going to folder ~cs9319/a2. Note that the original text files (i.e., .txt 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 BWT encoded files (.bwt files) during marking.

A simple sanity test script is also available in ~cs9319/a2. You should run the sanity test script before you submit your solution. The sanity test script is provided to help test the basic correctness (such as the program name and the number of input arguments). As part of the assignment, you are expected to perform further testing before submitting your solution.

To run the sanity test script on db-perftest, 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 bwt 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 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.

## bsearch

A simple search script called bsearch that accepts the original record text file (.txt and not .bwt) and one search term is provided at ~cs9319/a2 to assist you to construct more test cases.

For example, you can find out the expected result for querying medium2.bwt by searching its corresponding .txt file as follows:

```
%db-perftest> ~cs9319/a2/bsearch ~cs9319/a2/medium2.txt "Compression"
[32208]Compression of static and dynamic three-dimensional meshes.
[35732]Compression and visualization of large and animated volume data.
[48012]Compression and transmission of haptic data in telepresence and teleaction systems.
%db-perftest>
```

Althoguh bsearch only supports one search term, you can easily determine the results of queries with multiple search terms by using `grep` on its output. For example, the following determines the query result for `"Compression" "data"` on `medium2.bwt`.

```
%db-perftest> ~cs9319/a2/bsearch ~cs9319/a2/medium2.txt "Compression"|grep "data"
[35732]Compression and visualization of large and animated volume data.
[48012]Compression and transmission of haptic data in telepresence and teleaction systems.
%db-perftest> ~cs9319/a2/bsearch
Usage: bsearch FILE.txt PATTERN
%db-perftest>
```

# 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 5,000 matches in its output.
3. 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 (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. All matching shall be performed on the text values and not on record identifiers. Please pay attention to the special case when a search term contains only numbers. For example, a query with search term `"5"` will include the match of `[12] Born in 1995.` but exclude `[5] Born in 1996.`
6. You can assume that an input search term will not be an empty string (i.e., ""). Furthermore, search terms containing any square bracket, search terms containing dash dash (i.e., "--"), or search queries resulting in no matches will not be tested.
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. 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 `db-perftest`, a particular CSE Linux machine. Solutions that have compilation errors on `db-perftest` 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 `db-perftest`.

You can connect to the db-perftest server by firstly connecting to a CSE Linux machine (such as vlab) and then ssh to db-perftest using: `ssh Your_zID@db-perftest.cse.unsw.edu.au`

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 bwt encoded files generated from source text files that are larger than 160MB.

Runtime memory is assumed to be always less than **12MB**. 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 `db-perftest` 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 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: Wednesday 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 original deadline.** For example, if you have your special consideration granted by UNSW for a one-week extension, there will be no late penalty if the assignment is submitted within 7 days after the original deadline. However, no further late submissions will be accepted after these 7 days. Please read the General Assessment Information 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 db-perftest, 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 Academic Honesty and Plagarism section in the course outline for details.