COMP9319 2018s1 Assignment 1: Huffman Coding

Your task in this assignment is to implement the original, static Huffman coding algorithm. Your C/C++ program, called **huffman**, will encode a file (that may be ASCII or binary); decode the encoded file; and search an encoded file (with the search term limited to 7-bit ASCII).

huffman accepts a commandline option of either:

1. `-e` for encoding, or
2. `-d` for decoding, or
3. `-s` for search.

1. huffman -e [input-src-path] [output-src-path]

where:

- [input-src-path] is the path to an existing source file;
- [output-src-path] is the path to the encoded file (to be created). If the encoded file already exists, it will be overwritten.

Sample:

```
wagner> huffman -e ~cs9319/tmp/data/file.txt ~/data/output.huffman
```

2. huffman -d [input-src-path] [output-src-path]

where:

- [input-src-path] is the path to an existing encoded file;
- [output-src-path] is the path to the decoded file (to be created). If the decoded file already exists, it will be overwritten.

Sample:

```
wagner> huffman -d ~/data/output.huffman original.txt
```

3. huffman -s [query-string] [file-path]

where:

- [query-string] is the search term. The search term can be up to 256 characters and will not be empty. To make the assignment easier, we assume that the search is case sensitive.
- [file-path] is the path to an existing encoded file.

Sample:

```
wagner> huffman -s "apple" ~/data/output.huffman
```

Details

Any statistics (such as frequencies) of the source file or any extra information can be stored as a header of the encoded file. The header must be 1024 bytes long. You may not use more than 1024 bytes for statistics or any overhead information. If your program uses fewer than 1024 bytes, you must still allocate exactly 1024 bytes as the header of your encoded files.

Consider a file containing the text: `aaaabbcd`. Using Huffman encoding, it will be encoded into 14 bits, which will fit into 2

bytes. Combining this with the overhead of the header information, the size of the encoded file will be 1026 bytes.

For search, your program is only required to output the number of matches. For a file containing the text: `aaaabbcd`, if the search term is "aa", your program will output 3.

Your encoder and decoder shall work properly even when the file is empty or contains only a few characters, i.e., Decoding the encoded file should produce an identical source file. We will not verify and test the correctness of your encoded file when the encoded file (excluding the header overhead) is fewer than or equal to 8 bits.

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 a CSE Linux machine, e.g., wagner. Solutions that have compilation errors will receive zero points for the entire assignment.

```
make
```

Your solution is **not** allowed to write out any external files other than the output file specified in the commandline arguments. Any solution that writes out external files (even temporarily) other than the specified output file will receive zero points for the entire assignment.

Example

Consider a file called test1.txt consists of 30 a's, 30 b's, 20 c's, 10 d's and 10 e's (without any other characters such as newline etc); and another file called test2.txt as follows:

```
cababada
```

(According to Lecture 1, excluding the space for the header, the encoded test1.txt will be in total 220 bits and encoded test2.txt will be 14 bits.)
Some examples:

```
wagner % cd data
wagner % ls -l *.txt
-rw-r----- 1 cs9319 cs9319   100 Jun 15 02:46 test1.txt
-rw-r----- 1 cs9319 cs9319     8 Aug  8 20:36 test2.txt
wagner %
wagner % wc test1.txt
0 1 100 test1.txt
wagner % wc test2.txt
0 1 8 test2.txt
wagner % cd ..

wagner % huffman -e ~/a1/data/test1.txt data/test1.huff
wagner % huffman -d data/test1.huff data/test1.output
wagner % huffman -e data/test2.txt data/test2.huff
wagner % huffman -d data/test2.huff data/test2.output
wagner %
wagner % diff data/test1.txt data/test1.output
wagner %
wagner % diff data/test2.txt data/test2.output
wagner %
wagner % cd data
wagner % rm *.output
wagner % ls -l test*
-rw-r----- 1 cs9319 cs9319  1052 Aug  8 20:39 test1.huff
-rw-r----- 1 cs9319 cs9319   100 Jun 15 02:46 test1.txt
-rw-r----- 1 cs9319 cs9319  1026 Aug  8 20:39 test2.huff
-rw-r----- 1 cs9319 cs9319     8 Aug  8 20:36 test2.txt

wagner % ../huffman -s "abc" test2.huff
0
wagner % ../huffman -s "baba" test2.huff
1
wagner % ../huffman -s "aba" test2.huff
2
wagner % ../huffman -s "a" test2.huff
4
```

Performance

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

Runtime memory is assumed to be always less than 20MB. 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. Any solution that violates this memory requirement will receive zero points for that test. Any solution that runs for more than **60 seconds** on a machine with similar specification as *wagner* for a test be killed, and will receive zero points for that test. We will use the `time` command and count both the user and system time as runtime measurement.

Documentation

You will be marked on your descriptions in README.txt of the design of your Huffman tree, the header and how your search works. Your source code will be also inspected and marked based on readability and ease of understanding.

Assumptions/clarifications/hints

1. The input filename is a path to the given source file. Please open the file as read-only in case you do not have the write permission.
2. 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.
3. You can assume that the input query string will not be an empty string (i.e., "").

Marking

This assignment is worth 100 points. Below is an indicative marking scheme:

Component	Points
Auto marking (encoding, decoding)	45
Auto marking (search)	50
Documentation	5

Bonus

Bonus marks (up to 10 points) will be awarded for the solution that achieves 100 points and runs the fastest overall (i.e., the shortest total time to finish **all** the tests). This solution will be shared with the class. 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 24th August 23:59. Late submissions will have marks deducted from the maximum achievable mark at the rate of roughly 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 a1 makefile *.h *.c *.cpp README.txt
```

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.