
Report of Assignment3

Huffman v.s Arcd

Using Open-Source

錢宇倫

通訊二

110503511

DEPARTMENT OF COMMUNICATION ENGINEERING,

January 8, 2023

Contents

1	Usage	2
2	Result	3
3	Problem and solution	4

1 Usage

Huffman

- MakeFile

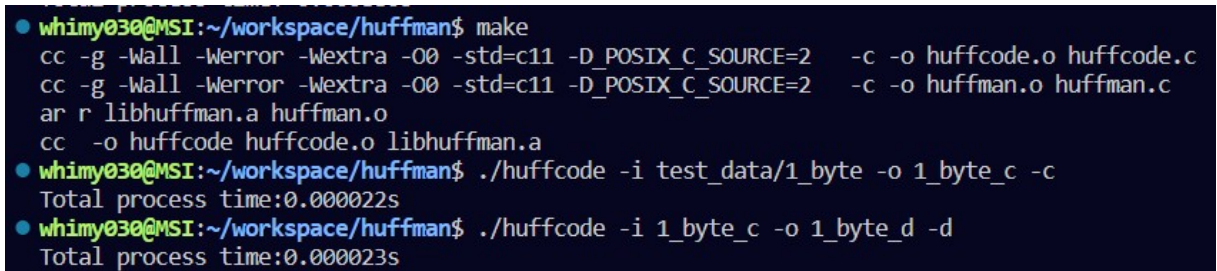
```
$ make
```

- Run for encode

```
$ ./huffcode -i [input_file_name] -o [output_file_name] -c
```

- Run for decode

```
$ ./huffcode -i [input_file_name] -o [output_file_name] -d
```



```
● whimyo30@MSI:~/workspace/huffman$ make
cc -g -Wall -Werror -Wextra -O0 -std=c11 -D_POSIX_C_SOURCE=2 -c -o huffcode.o huffcode.c
cc -g -Wall -Werror -Wextra -O0 -std=c11 -D_POSIX_C_SOURCE=2 -c -o huffman.o huffman.c
ar r libhuffman.a huffman.o
cc -o huffcode huffcode.o libhuffman.a
● whimyo30@MSI:~/workspace/huffman$ ./huffcode -i test_data/1_byte -o 1_byte_c -c
Total process time:0.000022s
● whimyo30@MSI:~/workspace/huffman$ ./huffcode -i 1_byte_c -o 1_byte_d -d
Total process time:0.000023s
```

figure.1 - Compile + Run of Huffman

Arcd

- MakeFile

```
$ make
```

- Run for encode

```
$ ./arcd_stream -e < [input_file_name] | tee [output_file_name]
```

- Run for decode

```
$ ./arcd_stream -d < [input_file_name] | tee [output_file_name]
```

```

● whimyo30@MSI:~/workspace/arcd$ make
[ 25%] Built target arcd
[ 50%] Built target adaptive_model
[ 75%] Built target arcd_stream
[100%] Built target codec_tests
● whimyo30@MSI:~/workspace/arcd$ cd examples
● whimyo30@MSI:~/workspace/arcd/examples$ ./arcd_stream -e < test_data/1_byte | tee encode_data/1_byte_e
A
Total process time:0.000008s
● whimyo30@MSI:~/workspace/arcd/examples$ ./arcd_stream -d < encode_data/1_byte_e | tee decode_data/1_byte_d
A
Total process time:0.000048s

```

figure.2 - Compile + Run of Arcd

2 Result

Sizeof data	Huffman encode	Arcd encode	Huffman decode	Arcd decode
1 byte	0.000033	0.000008	0.000008	0.000048
2 byte 1 syb	0.000020	0.000014	0.000009	0.000017
2 byte 2 syb	0.000019	0.000071	0.000010	0.000007
3 byte 1 syb	0.000022	0.000042	0.000008	0.000007
3 byte 3 syb	0.000025	0.000036	0.000010	0.000007
3 line	0.000033	0.000054	0.000011	0.000019
128 byte 1 syb	0.000027	0.000025	0.000010	0.000032
128 byte 2 syb	0.000037	0.000017	0.000011	0.000019
Revelation	0.000211	0.000372	0.000129	0.000799

Conclusion

- Huffman coding runs faster when processing big size data.
- Arithemtic coding runs faster when processing small size data.
- Both coding runs faster when decoding, slower when encoding.

3 Problem and solution

About code...

- Arcd
 - Cannot found arcd.h
=> solution -> Add `#define _XOPEN_SOURCE 600`
 - Cannot feed in file
=> solution -> Use `SHELL` command.

About Cmake...

- Cannot use cmake command
=> solution -> Install **CMake**
- Fail to MakeFile
=> solution ->
 1. Install **CCMake**
 2. Enter `ccmake .`
 3. Go into cmake curses (terminal handling library) interface.
 4. Turn `ARCD_EXAMPLES + ARCD_TESTS` on.

References

- [1] GitHub - drichardson/huffman: huffman encoder/decoder
- [2] GitHub - wonder-mice/arc4: Simple arithmetic coding library in C