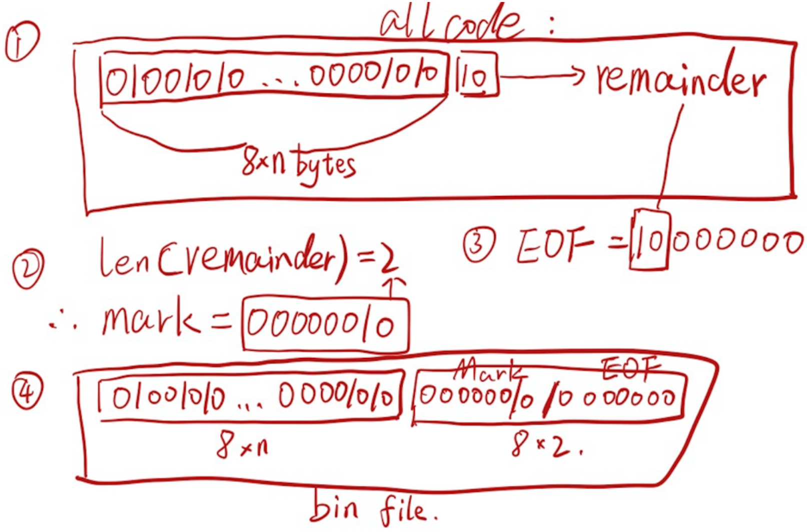
The report of assignment 2

1. A brief description of my program:

About the compress, firstly, I use regular expression to get the ‘char’ and ‘word’, put them in a list, using the counter method of library collections to get the frequency and then calculate the probability of every character and word, after sorted, saving them as a dictionary (i.e. the pkl file).

The process of encoding, I create two lists, one is used to store word or character’s name, another is to store their probability, then using loop to pop the last two elements of list every time and add ‘0’ or ‘1’ as their code, then append the new word and the new probability which are combined by the last two to their corresponding list respectively. Repeat this process until the word or char list only has one element, I’ll get the code of each word or character.

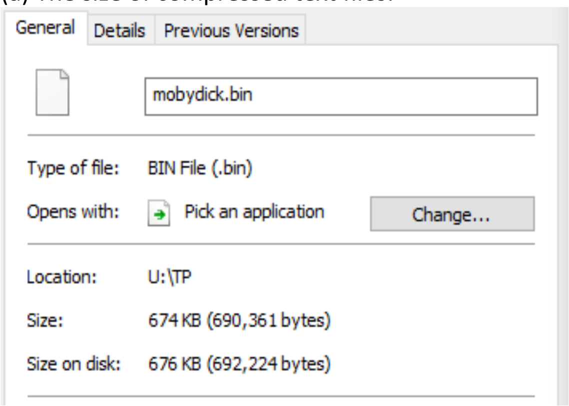
About the pseudo-EOF, I calculate whether the length of all code can be exact division by 8, if not, get the code of remainder and add 0 to extend to 8 bytes binary named ‘EOF’, at the same time, in order to know how many bytes in ‘EOF’ are original code, I convert the length of remainder to 8 bytes binary called ‘mark’, so I use the ‘mark’ and ‘EOF’ in place of the code of remainder to make the all code can be exact division by 8.



About the decoding, firstly, process the EOF, get the original code, then use loop to judge whether the code in model, if in, get the character of word, if not, change the length of code (every time plus one code).

2. The files size and time:

(a) The size of compressed text files:



mobydick.bin

Type of file: BIN File (.bin)

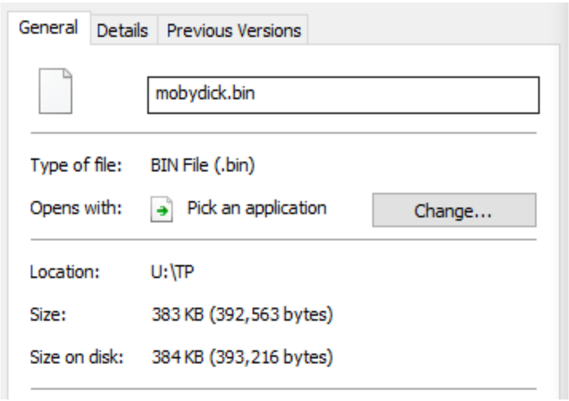
Opens with: Pick an application

Location: U:\TP

Size: 674 KB (690,361 bytes)

Size on disk: 676 KB (692,224 bytes)

(char)



mobydick.bin

Type of file: BIN File (.bin)

Opens with: Pick an application

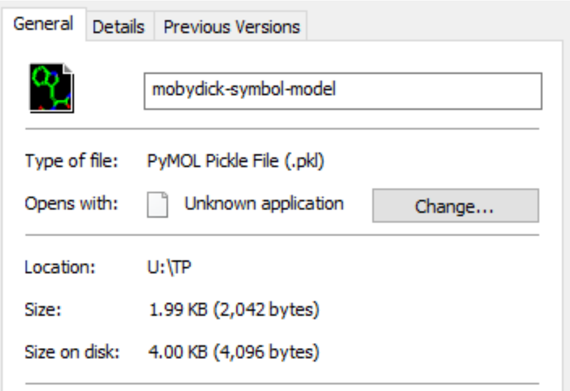
Location: U:\TP

Size: 383 KB (392,563 bytes)

Size on disk: 384 KB (393,216 bytes)

(word)

(b) The size of symbol models:



mobydick-symbol-model

Type of file: PyMOL Pickle File (.pkl)

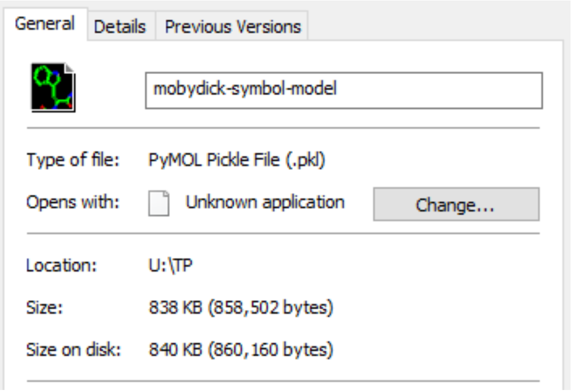
Opens with: Unknown application

Location: U:\TP

Size: 1.99 KB (2,042 bytes)

Size on disk: 4.00 KB (4,096 bytes)

(char)



mobydick-symbol-model

Type of file: PyMOL Pickle File (.pkl)

Opens with: Unknown application

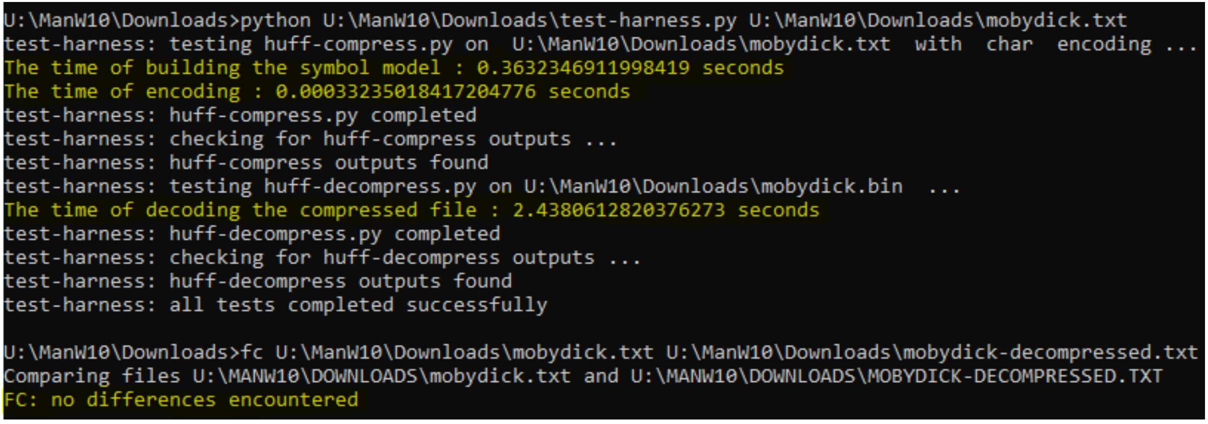
Location: U:\TP

Size: 838 KB (858,502 bytes)

Size on disk: 840 KB (860,160 bytes)

(word)

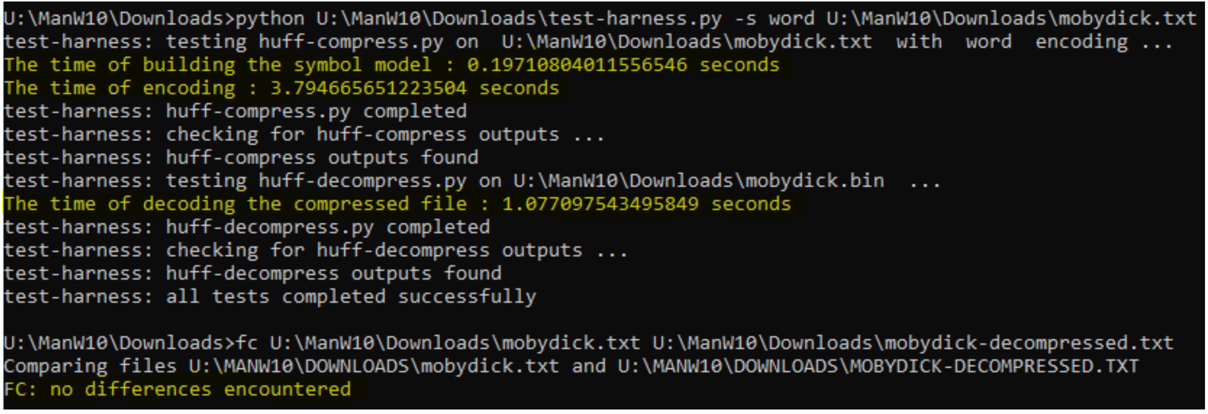
(c) The size of building symbol models, encoding, decoding:



```
U:\ManW10\Downloads>python U:\ManW10\Downloads\test-harness.py U:\ManW10\Downloads\mobydick.txt
test-harness: testing huff-compress.py on  U:\ManW10\Downloads\mobydick.txt  with  char  encoding ...
The time of building the symbol model : 0.3632346911998419 seconds
The time of encoding : 0.00033235018417204776 seconds
test-harness: huff-compress.py completed
test-harness: checking for huff-compress outputs ...
test-harness: huff-compress outputs found
test-harness: testing huff-decompress.py on U:\ManW10\Downloads\mobydick.bin  ...
The time of decoding the compressed file : 2.4380612820376273 seconds
test-harness: huff-decompress.py completed
test-harness: checking for huff-decompress outputs ...
test-harness: huff-decompress outputs found
test-harness: all tests completed successfully

U:\ManW10\Downloads>fc U:\ManW10\Downloads\mobydick.txt U:\ManW10\Downloads\mobydick-decompressed.txt
Comparing files U:\MANW10\DOWNLOADS\mobydick.txt and U:\MANW10\DOWNLOADS\MOBYDICK-DECOMPRESSED.TXT
FC: no differences encountered
```

(char)



```
U:\ManW10\Downloads>python U:\ManW10\Downloads\test-harness.py -s word U:\ManW10\Downloads\mobydick.txt
test-harness: testing huff-compress.py on  U:\ManW10\Downloads\mobydick.txt  with  word  encoding ...
The time of building the symbol model : 0.19710804011556546 seconds
The time of encoding : 3.794665651223504 seconds
test-harness: huff-compress.py completed
test-harness: checking for huff-compress outputs ...
test-harness: huff-compress outputs found
test-harness: testing huff-decompress.py on U:\ManW10\Downloads\mobydick.bin  ...
The time of decoding the compressed file : 1.077097543495849 seconds
test-harness: huff-decompress.py completed
test-harness: checking for huff-decompress outputs ...
test-harness: huff-decompress outputs found
test-harness: all tests completed successfully

U:\ManW10\Downloads>fc U:\ManW10\Downloads\mobydick.txt U:\ManW10\Downloads\mobydick-decompressed.txt
Comparing files U:\MANW10\DOWNLOADS\mobydick.txt and U:\MANW10\DOWNLOADS\MOBYDICK-DECOMPRESSED.TXT
FC: no differences encountered
```

(word)

3. Compare the result and how to improve:

I can see clearly from above pictures that the compress result of ‘word’ is better than ‘char’ since the size of ‘word’ bin file is smaller than the ‘char’ although the ‘word’ size of model is bigger than ‘char’. About the time, the time of creating model and decoding of ‘word’ is shorter than ‘char’, but the encoding time, the ‘char’ is far less than ‘word’. About improving, in my code of encoding, the speed of ‘word’ encoding still needs to be improved, concretely, the part of sorting, I can use ‘heapq’ to sort automatically to avoid using list to sort because using list might waste a lot of time. Besides, the part of decoding, I can find matched word or character from the minimum length of code instead of every time from one, which might reduce time.

4. Conclusion:

The better method of ‘word’ and ‘char’ is ‘word’, because the size of compress file is smaller than ‘char’ and the time of building model and decoding both smaller than ‘char’. So, in the future of work or study, I’ll use ‘word’.