COMP20290 - Assignment 2
Huffman Compression

Noemi Banal, Rebeca Buarque and Ruth Dooley

Contents:

Contributors:

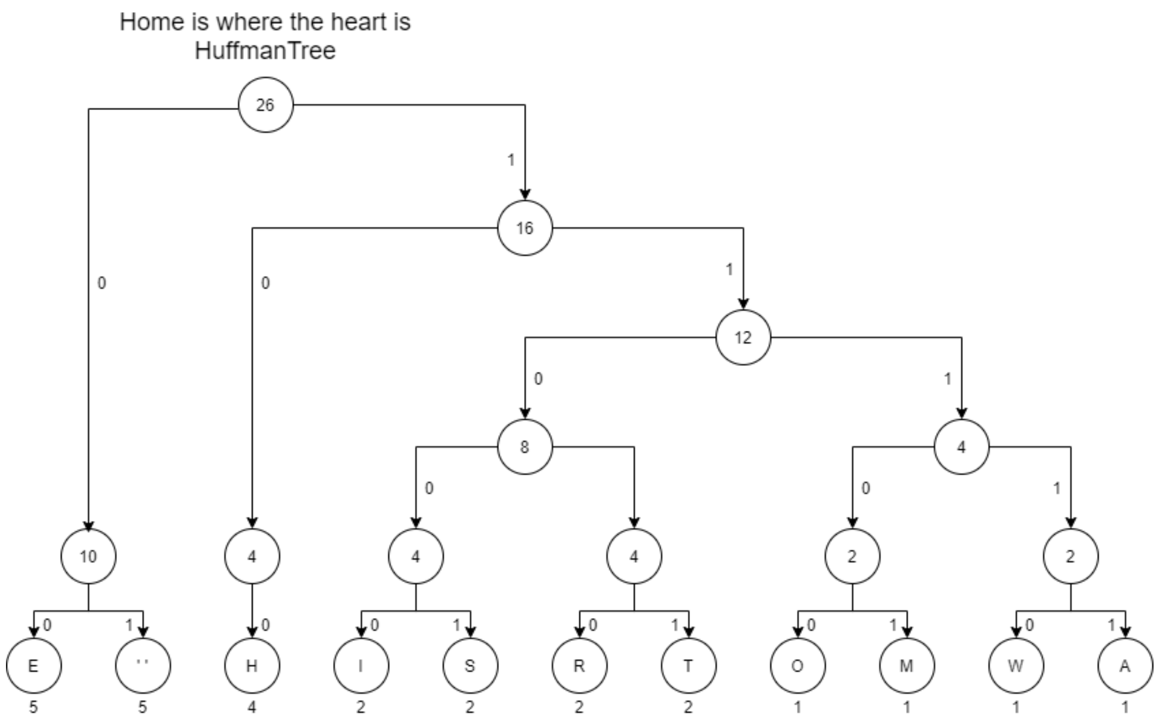
Name	Rebeca Buaque
Student Number	20204895
Github Repository	https://github.com/CompAlgorithms/algorithms20290-2021-repository-BecaBuarque

Name	Ruth Dooley
Student Number	19300753
Github Repository	https://github.com/CompAlgorithms/algorithms20290-2021-repository-RuthDooley

Name	Noemi Banal
Student Number	19413292
Github Repository	https://github.com/CompAlgorithms/algorithms20290-2021-repository-noemiBa

Task 1. Code Huffman Tree of phrase by hand

Create a Huffman tree and codeword table for the phrase: “Home is where the heart is”.



The total number of characters in the phrase is 26.

Char	Frequency	P(char)	Binary Code
“E”	5	5/26	00
“ “	5	5/26	01
“H”	4	4/26	100
“I”	2	2/26	11000
“S”	2	2/26	11001
“R”	2	2/26	11010
“T”	2	2/26	11011
“O”	1	1/26	11100
“M”	1	1/26	11101
“W”	1	1/26	11110
“A”	1	1/26	11111

The table outlines the characters contained in the phrase "Home is where the heart is", as well as their frequency. P(char) is the probability of the occurrence of a character in the phrase, which is obtained by dividing the frequency of each character by the total number of characters. Lastly, we have the binary code corresponding to each character, obtained from the Huffman Tree above.

Task 2. Build your Huffman Compression Suite

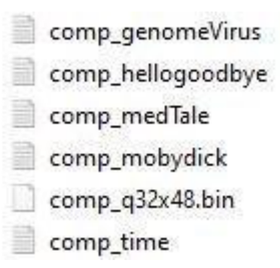
Using the command prompt, navigate to the correct directory which contains the Huffman suite. Once there, execute the following commands:

- To compile: `javac Huffman.java`
- To compress: `java Huffman compress inputName.type outputName.type`
- To decompress: `java Huffman decompress inputName.type outputName.type`

Helper classes used: BinaryIn, BinaryOut, MinPQ, Stopwatch.

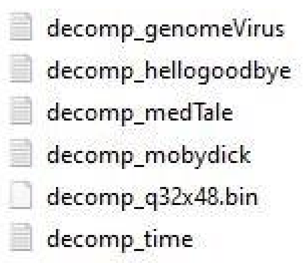
Task 3. Compression Analysis

Files compressed:



comp_genomeVirus
comp_hellogoodbye
comp_medTale
comp_mobydick
comp_q32x48.bin
comp_time

Files decompressed:



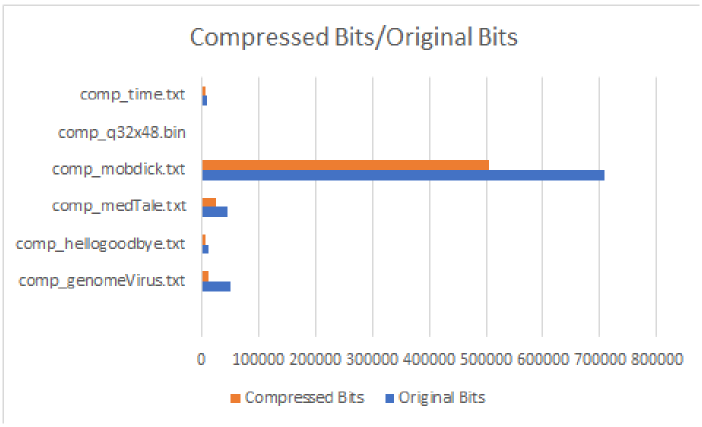
decomp_genomeVirus
decomp_hellogoodbye
decomp_medTale
decomp_mobydick
decomp_q32x48.bin
decomp_time

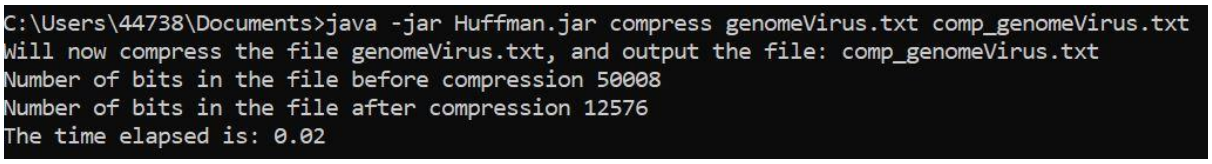
1. Experiments with compression:

$$\boxed{\text{Compression Ratio in percentage} = \frac{\textit{Compressed Bits}}{\textit{Original Bits}} * 100}$$

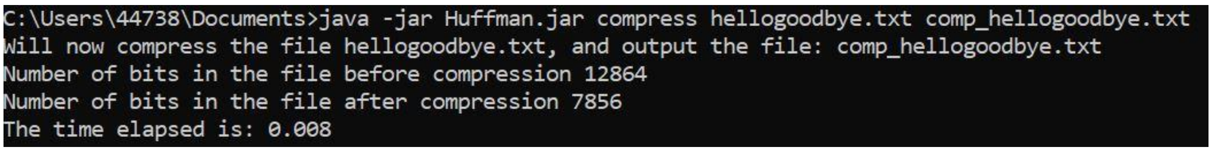
Input File	Output File	Original Bits	Compressed Bits	Compression Ratio (%)	Time (milliseconds)
genomeVirus.txt	comp_genomeVirus.txt	50008	12576	25.15%	0.02
hellogoodbye.txt	comp_hellogoodbye.txt	12864	7856	61.00%	0.008

medTale.txt	comp_medTale.txt	45872	24664	53.77%	0.008
mobydick.txt	comp_mobdick.txt	9708968	5505432	56.70%	0.168
q32x48.bin	comp_q32x48.bin	1536	816	53.13%	0.0
time.txt	comp_time.txt	9360	5640	60.30%	0.008

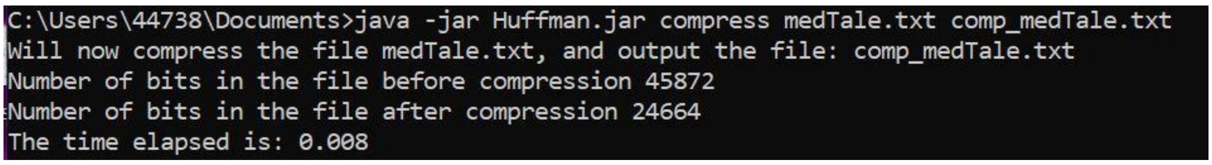




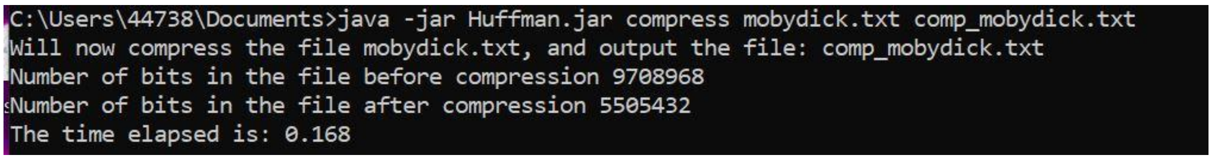
```
C:\Users\44738\Documents>java -jar Huffman.jar compress genomeVirus.txt comp_genomeVirus.txt
Will now compress the file genomeVirus.txt, and output the file: comp_genomeVirus.txt
Number of bits in the file before compression 50008
Number of bits in the file after compression 12576
The time elapsed is: 0.02
```



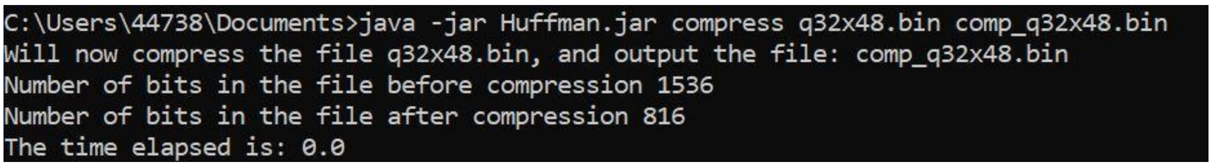
```
C:\Users\44738\Documents>java -jar Huffman.jar compress hellogoodbye.txt comp_hellogoodbye.txt
Will now compress the file hellogoodbye.txt, and output the file: comp_hellogoodbye.txt
Number of bits in the file before compression 12864
Number of bits in the file after compression 7856
The time elapsed is: 0.008
```



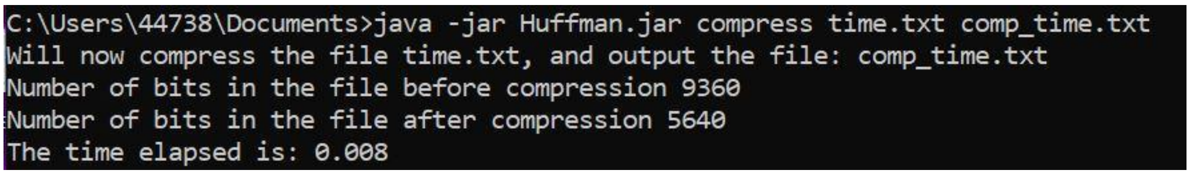
```
C:\Users\44738\Documents>java -jar Huffman.jar compress medTale.txt comp_medTale.txt
Will now compress the file medTale.txt, and output the file: comp_medTale.txt
Number of bits in the file before compression 45872
Number of bits in the file after compression 24664
The time elapsed is: 0.008
```



```
C:\Users\44738\Documents>java -jar Huffman.jar compress mobydick.txt comp_mobydick.txt
Will now compress the file mobydick.txt, and output the file: comp_mobydick.txt
Number of bits in the file before compression 9708968
Number of bits in the file after compression 5505432
The time elapsed is: 0.168
```



```
C:\Users\44738\Documents>java -jar Huffman.jar compress q32x48.bin comp_q32x48.bin
Will now compress the file q32x48.bin, and output the file: comp_q32x48.bin
Number of bits in the file before compression 1536
Number of bits in the file after compression 816
The time elapsed is: 0.0
```



```
C:\Users\44738\Documents>java -jar Huffman.jar compress time.txt comp_time.txt
Will now compress the file time.txt, and output the file: comp_time.txt
Number of bits in the file before compression 9360
Number of bits in the file after compression 5640
The time elapsed is: 0.008
```

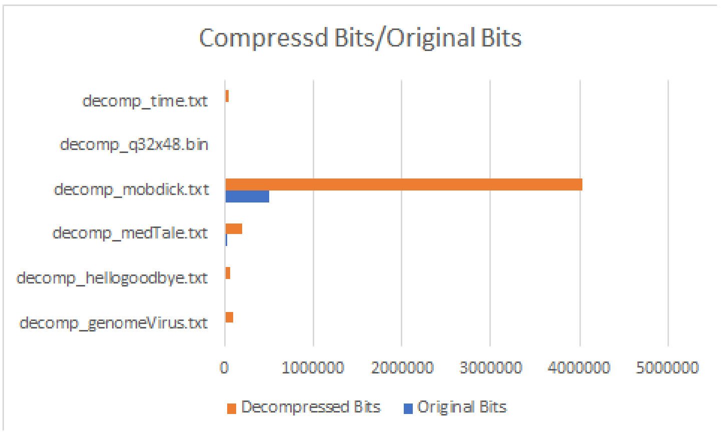
Analysis:

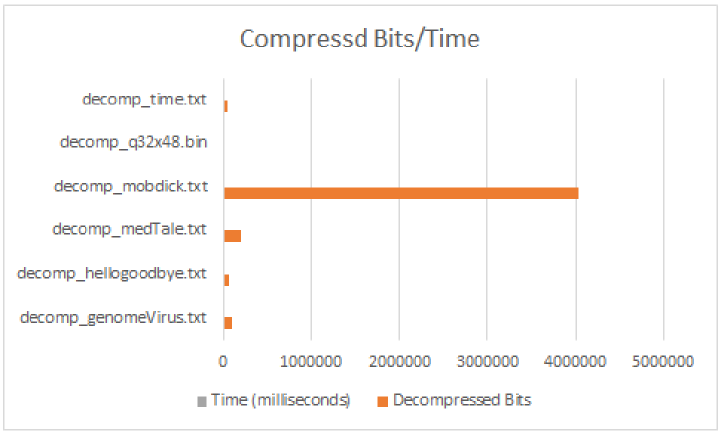
Running time of Huffman Compression: R , where N is the input size and R is the alphabet size.
Big O complexity of Huffman Compression: $n)$

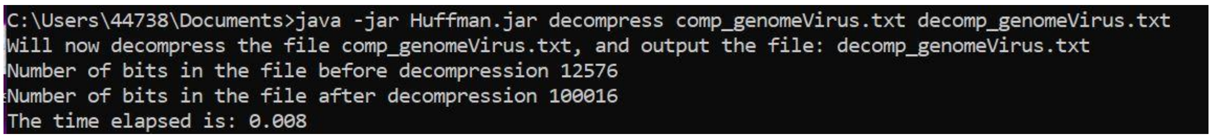
- **genomeVirus.txt:** This file contains genomic code (4 characters, ATCG with approximately equal frequencies). Therefore, the Huffman tree is balanced, and each character is compressed from 8-bit ASCII to a 2-bit code. Hence, the compression ratio is about 25.15%. Also, the repetition of the same character makes the compression smaller and slower.
- **hellogoodbye.txt**: This file contains English text from a romance poem. The compression ratio of 61% is very high, the compressed size is less than half the original. The compression time was very low as expected of a small text in 8-bit ASCII.
- **medTale.txt:** This file contains English text from the medieval age. The compression ratio of 53.77% is very good, nearly half of the size of the original. The compression time was very low as expected of a small text in 8-bit ASCII.
- **mobydick.txt:** This is larger file that contains English text of nearly 10 million bits of a novel by American writer Herman Melville. This file was the longest that the Huffman Algorithm took to compress, taking up to 166 milliseconds because of the extent of the file but the compression rate of 56.70% is very good and this is due the file containing only 8-bit ASCII text.
- **q32x48.bin:** This is a bitmap that the Huffman algorithm can compress with a ratio of 53.13%, thus reducing the size of the map by about half and running instantly.
- **time.txt:** This file contains a small English text from a poem about time. Compression ratio of 60.30% % is very high, the compressed size is less than half the original. The compression time was very low as expected of a small text in 8-bit ASCII.

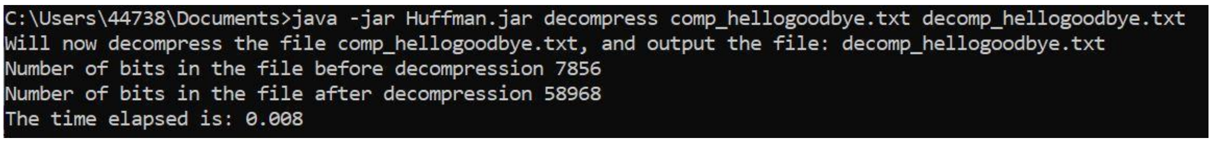
2. Experiments with decompression:

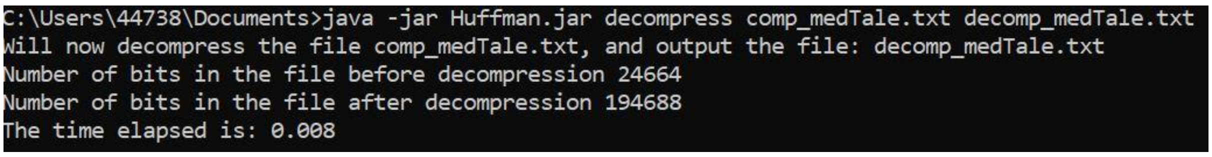
Input File	Output File	Original Bits	Decompressed Bits	Time (milliseconds)
genomeVirus.txt	decomp_genomeVirus.txt	12576	100016	0.008
hellogoodbye.txt	decomp_hellogoodbye.txt	7856	58968	0.008
medTale.txt	decomp_medTale.txt	24664	194688	0.008
mobydick.txt	decomp_mobdick.txt	5505432	44036704	0.264
q32x48.bin	decomp_q32x48.bin	816	4648	0.0
time.txt	decomp_time.txt	5640	41384	0.008

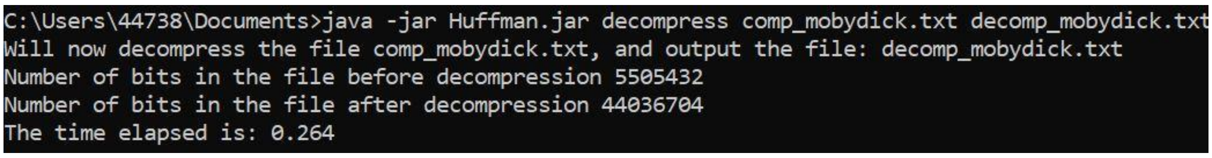




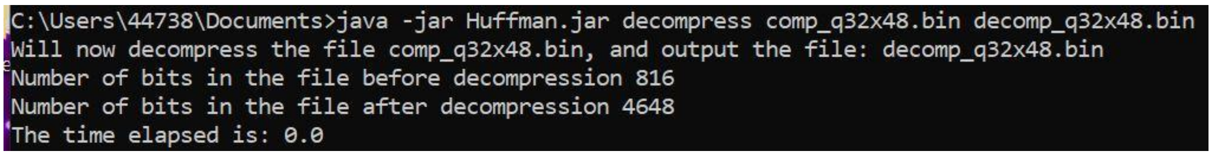

```
C:\Users\44738\Documents>java -jar Huffman.jar decompress comp_genomeVirus.txt decomp_genomeVirus.txt
Will now decompress the file comp_genomeVirus.txt, and output the file: decomp_genomeVirus.txt
Number of bits in the file before decompression 12576
Number of bits in the file after decompression 100016
The time elapsed is: 0.008
```


```
C:\Users\44738\Documents>java -jar Huffman.jar decompress comp_hellogoodbye.txt decomp_hellogoodbye.txt
Will now decompress the file comp_hellogoodbye.txt, and output the file: decomp_hellogoodbye.txt
Number of bits in the file before decompression 7856
Number of bits in the file after decompression 58968
The time elapsed is: 0.008
```

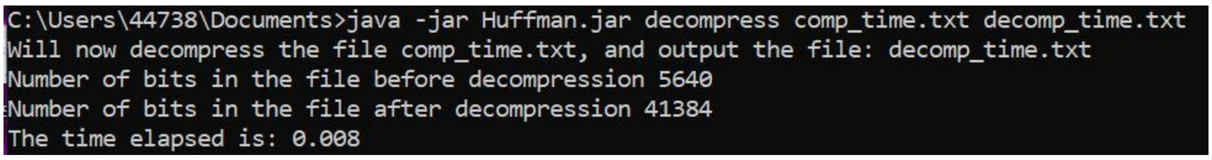

```
C:\Users\44738\Documents>java -jar Huffman.jar decompress comp_medTale.txt decomp_medTale.txt
Will now decompress the file comp_medTale.txt, and output the file: decomp_medTale.txt
Number of bits in the file before decompression 24664
Number of bits in the file after decompression 194688
The time elapsed is: 0.008
```



```
C:\Users\44738\Documents>java -jar Huffman.jar decompress comp_mobydick.txt decomp_mobydick.txt
Will now decompress the file comp_mobydick.txt, and output the file: decomp_mobydick.txt
Number of bits in the file before decompression 5505432
Number of bits in the file after decompression 44036704
The time elapsed is: 0.264
```



```
C:\Users\44738\Documents>java -jar Huffman.jar decompress comp_q32x48.bin decomp_q32x48.bin
Will now decompress the file comp_q32x48.bin, and output the file: decomp_q32x48.bin
Number of bits in the file before decompression 816
Number of bits in the file after decompression 4648
The time elapsed is: 0.0
```



```
C:\Users\44738\Documents>java -jar Huffman.jar decompress comp_time.txt decomp_time.txt
Will now decompress the file comp_time.txt, and output the file: decomp_time.txt
Number of bits in the file before decompression 5640
Number of bits in the file after decompression 41384
The time elapsed is: 0.008
```

Analysis:

Since the trie has already been built the complexity of the decompression is a lot lower than the compression algorithm. The Huffman Algorithm will only need to traverse the trie and expand the previously compressed bits. We can guess the algorithm works when we see the decompressed bits getting back to the original bits number. We can notice that the time of medTale.txt, hellogoodbye.txt and q32x48.bin stayed the same. The genomeVirus.txt decreased the time and mobdick.txt almost doubled up the time compared to the compression algorithm. The running time of the Huffman Decompression depends on the number of bits.

3.

- Q3. If we try to attempt to compress a file that we already have compressed, the size of the compressed version in bits increases to a larger size again. At best the size of the compressed file will remain the same. The idea of compression is to remove the redundancies that are in a file using an optimized alphabet system. If the file was compressed with a good compression algorithm the file should be already free from redundancies therefore there will be nothing extra to compress.

- Q4.

Algorithm	Input File	Original Bits	Compressed Bits	Compression Ratio
Huffman Algorithm	q32x48.bin	1536	816	53.13%
Run Length Function	q32x48.bin	1536	1144	74.48%

From the above analysis it shows that the Huffman algorithm did a better job at compressing the input file compressing the size to 53.13% of the size compared to the run length function that only compressed the input file to 74.48% of the size. From The implementation of the code I can understand this is due to the following:

- <u>Run length encoding:</u> This form of compression implements a sequence of single data values and counts. This is most useful for data that is representative and most useful for compressing input files where there is a continuous stream of identical data values eg. compressing an image where a large proportion of the image is the same colour. Repetition creates a greater compression.
- <u>Huffman Encoding:</u> The huffman algorithm is a more generic form of compression. It identifies the optimal alphabet to reduce the size of the file, removing redundant data. This would suggest that this encoding style would be much more powerful for a file of data that wouldn't have much repeating data such as the input file in question q32x48.bin. This type of encoding also reduces bits per character. With a large alphabet you can reduce the number of bits because a number of characters will have less than 8 bits of a standard character in binary code, thus reducing the bit size again.

Helper_Code:

- **BinaryStdIn** - Reads bits from the system
- **BinaryStdOut** - Writes bits to the system
- **StdIn** - Reads in data of various types from standard input.
- **StdOut -** Writes data of various types to standard output
- **MinPQ -** Generic min priority queue implementation with a binary heap.
- **StopWatch** – Class to calculate the running time.
- **RunLength** - an implementation of RunLength encoding that you can use in Task 3 to benchmark your Huffman algorithm. You can call it from the command line with: **java RunLength - < yourfilename** to compress your chosen file and **java RunLength + < yourfilename** to decompress it.

Helper Data

Several text files have been provided in the github assignment folder that are commonly used to test the effectiveness of compression algorithms. Use them to assess the performance of your implementation of Huffman and benchmark against that of other compression algorithms.

- `genomeVirus.txt`
- `hellogoodbye.txt`
- `medTale.txt`
- `mobydick.txt`
- `q32x48.bin`
- `time.txt`