Compression Assignment

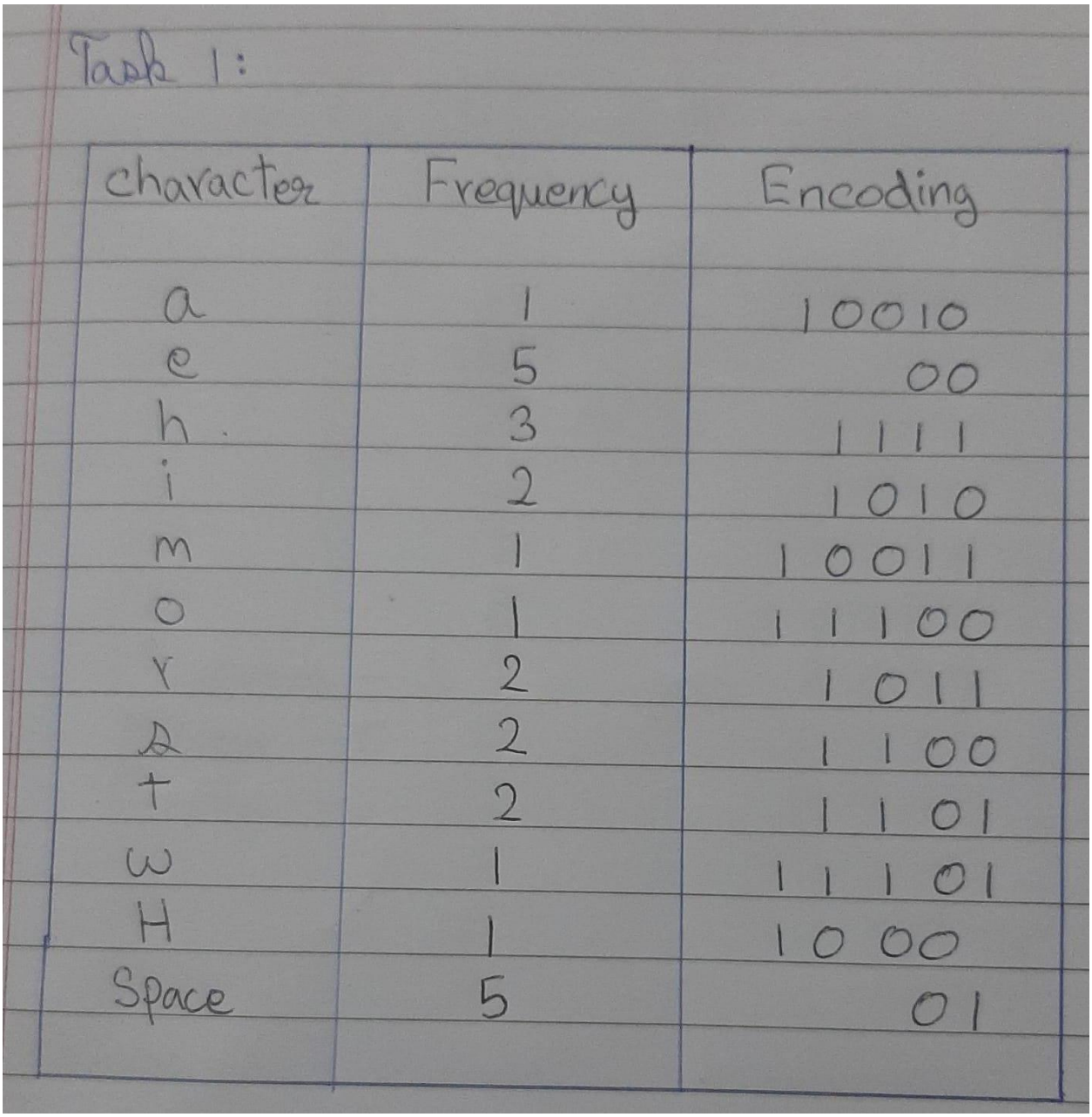
Group members-

David Fernandes(Student Number: 19203219)

Sagar Mahajan(Student Number:19204052)

Task 1: Code Huffman Tree of phrase by hand

Character frequency and codeword table:

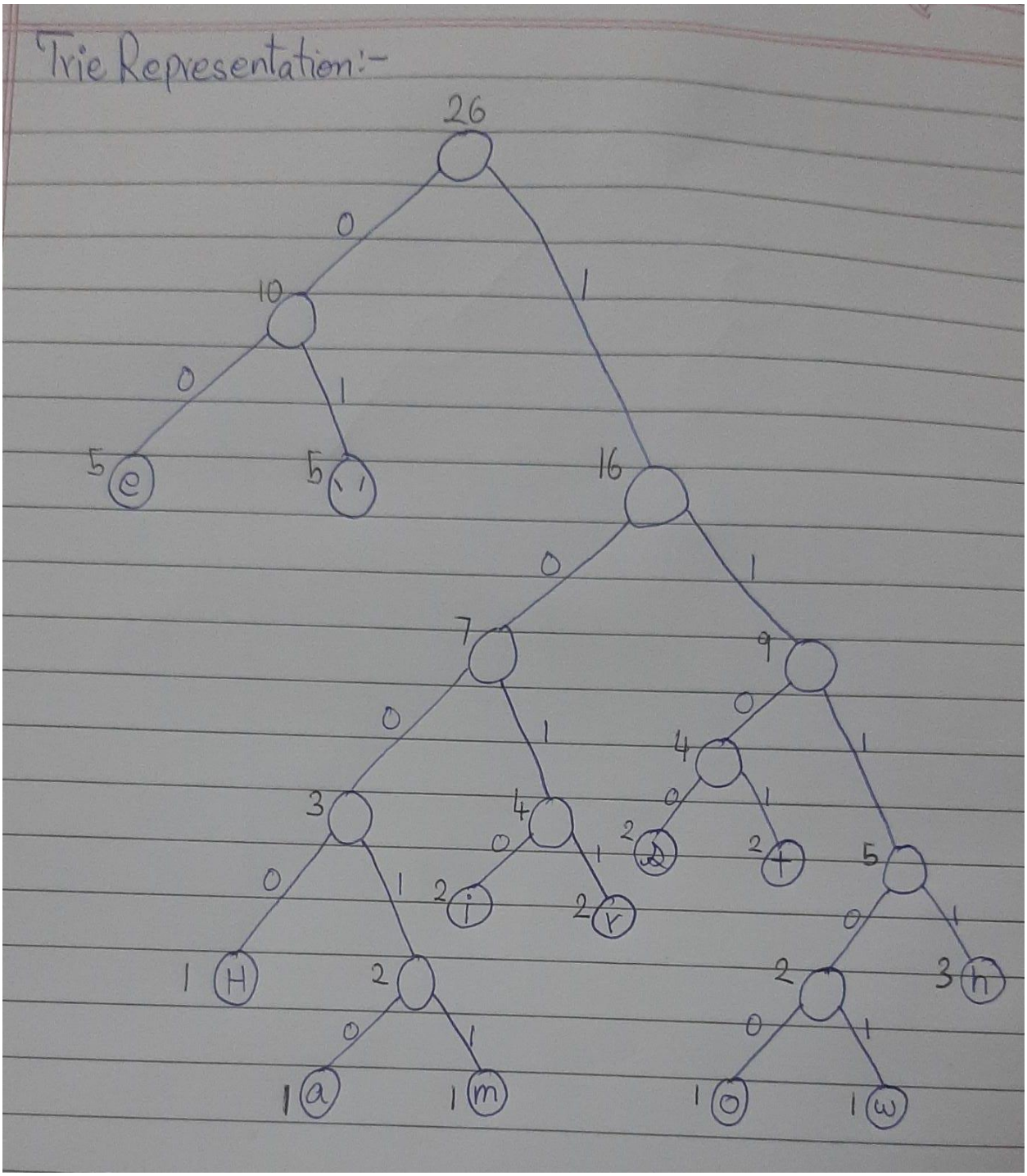


Task 1:

character	Frequency	Encoding
a	1	10010
e	5	00
h	3	1111
i	2	1010
m	1	10011
o	1	11100
r	2	1011
s	2	1100
t	2	1101
w	1	11101
H	1	1000
Space	5	01

Home - 1000 11100 10011 00
is - 1010 1100
where - 11101 1111 00 1011 00
the - 1101 1111 00
heart - 1111 00 10010 1011 1101
'space' - 01

Huffman tree:



Task 2: Coding the Huffman Algorithm in java

The class path in the repository is
src/CompressionAssignment/HuffmanAlg/Huffman.java

Compilation: javac HuffmanAlg/Huffman.java

To compress: java HuffmanAlg/Huffman compress HuffmanAlg/*inputfile* HuffmanAlg/*outputfile*

To decompress: java HuffmanAlg/Huffman decompress HuffmanAlg/*inputfile* HuffmanAlg/*outputfile*

Task 3: Compression Analysis:

_comp: Compressed file

_decomp: Decompressed file

1. We compress the given files and record its compression ratio and time taken to compress them.

Input file	Output file	Original size(bits)	Compressed Size(bits)	Compression ratio	Time (milliseconds)
medTale.txt	medTale_comp.txt	45032	23896	53.06%	12
mobydick.txt	mobydick_comp.txt	9531704	5341208	56.03%	151
genomeVirus.txt	genomeVirus_comp.txt	50016	14040	28.07%	8
Input.txt	Input_comp.txt	3040	2008	66.05%	4
q32x48.bin	q32x48_comp.bin	1536	816	53.12%	5

medTale.txt

```
(base) Sagars-MacBook-Pro:src sagarmahajan$ java HuffmanAlg/Huffman compress HuffmanAlg/medTale.txt HuffmanAlg/medTale_comp.txt
File to be Compressed: HuffmanAlg/medTale.txt
File compressed into: HuffmanAlg/medTale_comp.txt
The time taken: 12 milli second(s)
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/medTale.txt
45032 bits
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/medTale_comp.txt
23896 bits
```

mobydick.txt

```
(base) Sagars-MacBook-Pro:src sagarmahajan$ java HuffmanAlg/Huffman compress HuffmanAlg/mobydick.txt HuffmanAlg/mobydick_comp.txt
File to be Compressed: HuffmanAlg/mobydick.txt
File compressed into: HuffmanAlg/mobydick_comp.txt
The time taken: 151 milli second(s)
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/mobydick.txt
9531704 bits
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/mobydick_comp.txt
5341208 bits
```

genomeVirus.txt

```
(base) Sagars-MacBook-Pro:src sagarmahajan$ java HuffmanAlg/Huffman compress HuffmanAlg/genomeVirus.txt HuffmanAlg/genomeVirus_comp
.txt
File to be Compressed: HuffmanAlg/genomeVirus.txt
File compressed into: HuffmanAlg/genomeVirus_comp.txt
The time taken: 8 milli second(s)
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/genomeVirus.txt
50016 bits
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/genomeVirus_comp.txt
14040 bits
```

input.txt

```
(base) Sagars-MacBook-Pro:src sagarmahajan$ java HuffmanAlg/Huffman compress HuffmanAlg/input.txt HuffmanAlg/input_comp.txt
File to be Compressed: HuffmanAlg/input.txt
File compressed into: HuffmanAlg/input_comp.txt
The time taken: 4 milli second(s)
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/input.txt
3040 bits
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/input_comp.txt
2008 bits
```

q32x48.bin

```
(base) Sagars-MacBook-Pro:src sagarmahajan$ java HuffmanAlg/Huffman compress HuffmanAlg/q32x48.bin HuffmanAlg/q32x48_comp.bin
File to be Compressed: HuffmanAlg/q32x48.bin
File compressed into: HuffmanAlg/q32x48_comp.bin
The time taken: 5 milli second(s)
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/q32x48.bin
1536 bits
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/q32x48_comp.bin
816 bits
```

2. We then decompress the resultant compressed files and record their size and time taken to complete decompression.

Input file	Output file	Decompressed Size(bits)	Time(milliseconds)
medTale_comp.txt	medTale_decomp.txt	45032	6
mobydick_comp.txt	mobydick_decomp.txt	9531704	75
genomeVirus_comp.txt	genomeVirus_decomp.txt	50016	4
Input_comp.txt	Input_decomp.txt	3040	3
q32x48_comp.bin	q32x48_decomp.bin	1536	2

medTale_comp.txt

```
(base) Sagars-MacBook-Pro:src sagarmahajan$ java HuffmanAlg/Huffman decompress HuffmanAlg/medTale_comp.txt HuffmanAlg/medTale_decom
p.txt
File to be decompressed: HuffmanAlg/medTale_comp.txt
File decompressed into: HuffmanAlg/medTale_decomp.txt
The time taken: 6milli second(s)
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/medTale_decomp.txt
45032 bits
```

Mobydick_comp.txt

```
(base) Sagars-MacBook-Pro:src sagarmahajan$ java HuffmanAlg/Huffman decompress HuffmanAlg/mobydick_comp.txt HuffmanAlg/mobydick_dec
omp.txt
File to be decompressed: HuffmanAlg/mobydick_comp.txt
File decompressed into: HuffmanAlg/mobydick_decomp.txt
The time taken: 75milli second(s)
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/mobydick_decomp.txt
9531704 bits
```

genomeVirus_comp.txt

```
(base) Sagars-MacBook-Pro:src sagarmahajan$ java HuffmanAlg/Huffman decompress HuffmanAlg/genomeVirus_comp.txt HuffmanAlg/genomeVir
us_decomp.txt
File to be decompressed: HuffmanAlg/genomeVirus_comp.txt
File decompressed into: HuffmanAlg/genomeVirus_decomp.txt
The time taken: 4milli second(s)
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/genomeVirus_decomp.txt
50016 bits
```

Input_comp.txt

```
(base) Sagars-MacBook-Pro:src sagarmahajan$ java HuffmanAlg/Huffman decompress HuffmanAlg/input_comp.txt HuffmanAlg/input_decomp.tx
t
File to be decompressed: HuffmanAlg/input_comp.txt
File decompressed into: HuffmanAlg/input_decomp.txt
The time taken: 3milli second(s)
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/input_decomp.txt
3040 bits
```

q32x48_comp.bin

```
(base) Sagars-MacBook-Pro:src sagarmahajan$ java HuffmanAlg/Huffman decompress HuffmanAlg/q32x48_comp.bin HuffmanAlg/q32x48_decomp.
bin
File to be decompressed: HuffmanAlg/q32x48_comp.bin
File decompressed into: HuffmanAlg/q32x48_decomp.bin
The time taken: 2milli second(s)
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/q32x48_decomp.bin
1536 bits
```

ANALYSIS

- We observe that the time taken to compress a file depends on its size. The larger the file is, the longer it takes to compress.
- When we decompress a compressed file, it results in the original file with no loss of data, this shows that Huffman encoding is a lossless data compression algorithm.
- We observe that the time taken to decompress a file is less compared to the time taken to compress a file. This is because while decompressing a compressed file, the algorithm just has to go through the trie which has already been made when the file was first compressed and expand it.
- We observe that the compression ratio of a file depends on the number of different types of characters used in the file. For example, in the files genomeVirus.txt and medTale.txt. They are of similar size but the compression ratio of genomeVirus.txt is much lower compared to medTale.txt. This is because the genomeVirus.txt contains just 4 characters with varying frequencies whereas medTale.txt contains a lot more. This shows that compression is more efficient with a smaller range of characters rather than a wider range.

Q3. What happens if you try to compress one of the already compressed files? Why do you think this occurs?

Answer- We compress the medTale.txt file twice.

```
(base) Sagars-MacBook-Pro:src sagarmahajan$ java HuffmanAlg/Huffman compress HuffmanAlg/medTale_comp.txt HuffmanAlg/medTale_test.tx
t
File to be Compressed: HuffmanAlg/medTale_comp.txt
File compressed into: HuffmanAlg/medTale_test.txt
The time taken: 10 milli second(s)
(base) Sagars-MacBook-Pro:src sagarmahajan$ java Util/BinaryDump.java 0 < HuffmanAlg/medTale_test.txt
25928 bits
```

Input file	Compression size(bits)
medTale_comp.txt	23896
medTale_test.txt	25928

When compressed, the characters in the file get transformed from ASCII characters to some unreadable characters. When we then try to compress the compressed file, due to there being more characters spread out over lower average frequencies, the file size increases.

Q4. Use the provided RunLength function to compress the bitmap file q32x48.bin. Do the same with your Huffman algorithm. Compare your results. What reason can you give for the difference in compression rates?

Answer-
Original size = 1536 bits

Huffman Compression:

size = 816 bits

Ratio = 816/1536 X 100 = 53.13%

RunLength Compression:

size = 1144 bits

Ratio = 1144/1536 X 100 = 74.48%

It's clearly visible that Huffman compression performs better than RunLength as Huffman compresses the size of the file by 21% more than that of Runlength.