Programming Assignment – Huffman Encoding and bzip

Correctness

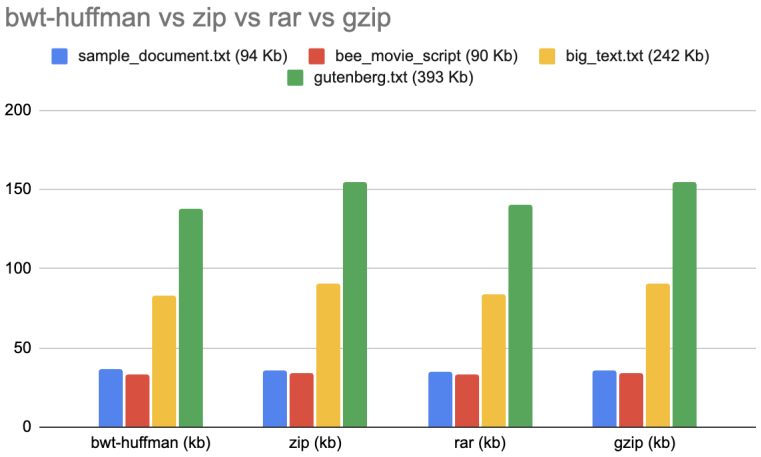
It is correct and compresses the input file to a smaller size and decompresses the compressed file to the original file.

Benchmarking

We benchmarked our implementation on several inputs and compared our implementation with several other compression software like zip, rar, and gzip. We tested it with different file sizes and different types of files. We first tested it with smaller files like 90 and 94 kb then moved on to medium and larger files like 242, 393, and 2,600 kb. For our implementation, we used BWT for all documents except for the bit map image where we disabled it.

Compression Sizes	bwt-huffman (kb)	zip (kb)	rar (kb)	gzip (kb)
sample_document.txt (94 Kb)	34	36	35	36
bee_movie_script (90 Kb)	33	34	33	34
big_text.txt (242 Kb)	83	91	84	91
gutenberg.txt (393 Kb)	138	155	140	155
office_hours.bmp (2600 Kb)	395	60	44	60

Table 1: Compression Sizes



Our implementation is roughly the same size as zip and gzip with the exception of larger files. At smaller and medium files like 90, 94, 242 kb, and 393 our implementation is roughly

the same size as all the other compression software. However, at larger files like 2,600 kb, our implementation without bwt and mtf algorithms, is significantly larger than the other compression software. This is most evedent in the office_hours.bmp file where our implementation is 6.5 times larger than zip and gzip and almost 9 times larger than rar. We then also calculated the compression ratio with the given data. We do this by dividing the size of the original file by the size of the compressed file.

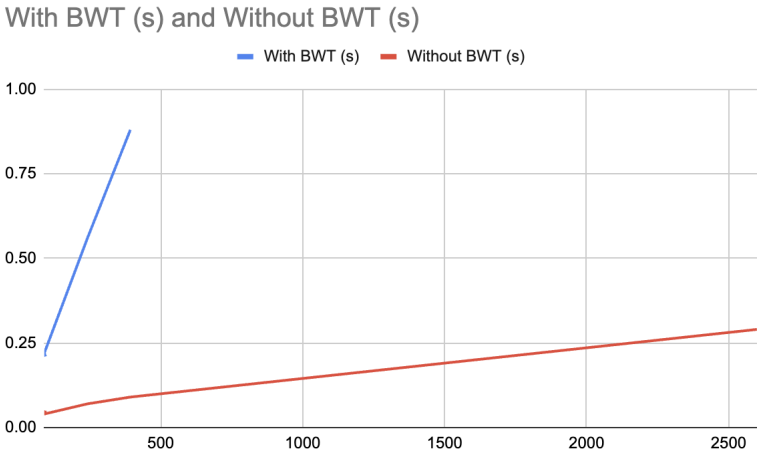
Ratio	bwt-huffman	zip	rar	gzip
sample_document.txt (94 Kb)	2.76	2.61	2.69	2.61
bee_movie_script (90 Kb)	2.73	2.65	2.73	2.65
big_text.txt (242 Kb)	2.92	2.66	2.88	2.66
office_hours.bmp (2600 Kb)	6.58	43.33	59.09	43.33
gutenberg.txt (393 Kb)	2.85	2.54	2.81	2.54

Table 2: Compression Ratio

The ratios also support our findings along with showing that our implementation is roughly the same ratios for smaller and medium files. We also compared the time it took to compress the files. We mainly tested the time difference between our implementation with bwt and without bwt. All other compression software were all almost instantanious or just .01 seconds.

	With BWT (s)	Without BWT (s)
sample_document.txt (94 Kb)	0.21	0.05
bee_movie_script (90 Kb)	0.22	0.04
big_text.txt (242 Kb)	0.56	0.07
gutenberg.txt (393 Kb)	0.88	0.09
office_hours.bmp (2600 Kb)	Never finishes	0.29

Table 3: Compression Time



We can see that our implementation with bwt is significantly slower than without bwt. This is most evident in the office_hours.bmp file where our implementation with bwt never finishes. Having to run bwt almost exponentially increases the time it takes to compress the file. While without bwt it linealy increases the time it takes to compress the file. This makes sense as bwt is a whole other algorithm that has to be run before encoding which also checks through the entire file. Though it is slower to compress, we can now check how the compression and its ratio changes with and without bwt.

Compression	With BWT (kb)	Without BWT (kb)
sample_document.txt (94 Kb)	34	51
bee_movie_script (90 Kb)	33	55
big_text.txt (242 Kb)	83	141
gutenberg.txt (393 Kb)	138	226
office_hours.bmp (2600 Kb)	Never finishes	395

Table 4: Compression Size With and Without BWT

After running with and without bwt, we can see that the compression size is significantly larger without bwt. This shows now that by sacrificing time, we can get a much better compression. This shows the importance of bwt as it can significantly reduce the size of the file. This makes sense as bwt and mtf can arrange the strings in a way where the same strings can be grouped together, making it easier to compress. The time it takes also makes sense as it is running through the entire file to grab all the strings.

The ratios also show that the compression ratio is much better with bwt. Having bwt is very important in compressing, however for larger files like the bit map, it would take too long to compress therefore it would be better to compress it without bwt. The decompression time is also changed with and without bwt.

Ratio	With BWT (kb)	Without BWT (kb)
sample_document.txt (94 Kb)	2.76	1.84
bee_movie_script (90 Kb)	2.73	1.64
big_text.txt (242 Kb)	2.92	1.72
gutenberg.txt (393 Kb)	2.85	1.74
office_hours.bmp (2600 Kb)	Never finishes	6.58

Table 5: Compression Ratio With and Without BWT

Decompression Time	With BWT (s)	Without BWT (s)
sample_document.txt (94 Kb)	0.09	0.07
bee_movie_script (90 Kb)	0.08	0.08
big_text.txt (242 Kb)	0.21	0.14
gutenberg.txt (393 Kb)	0.36	0.21
office_hours.bmp (2600 Kb)	Never finishes	0.31

Table 6: Decompression Time With and Without BWT

We can see that the decompression time is much faster without bwt. This is again contributed to the fact that having to translate the bwt with the inverse bwt takes more time.

Analysis

In this assignment we implemented the basis of a bzip2 compression. We did this by implementing the Huffman encoding algorithm, the Burrows-Wheeler transform, and move-to-front encoding. We then used these algorithms to compress and decompress a file. In the benchmarking section, we tested our implementation on several files and compared it to several other compression software. We compared the compression ratio in table 2 which shows that our implementation does very well with smaller and medium files but does not do as well with larger files. This is because the larger files take too long to compress using bwt and mtf, therefore we could only test it without those algorithms. The ratio for our implementation is around 2 to 3 from the original file size. If we were to test it with bwt and mtf, it would most likely have similar compression ratios to the other compression software. Overall the compression ratio of our implementation with bwt and mtf is around the same or slightly better than the other compression software. When we remove bwt and mtf, our implementation does not do as well as the other compression software. As shown in tables 3, 4, and 5, by removing bwt and mtf, the compression ratio is much worse but much faster. This is because bwt and mtf are very important in compressing files as they can group similar strings

together. By grouping them together, there are less unique strings to encode which makes it easier to compress. By grouping them and encoding less unique strings, the compression ratio would then be much better as there are strings to encode, making a smaller file. The compression ratio without bwt and mtf is around 1.5 to 2 from the original file size which is much worse than the other compression software and our implementation with bwt and mtf. The time it takes to compress and decompress is also shown in tables 3 and 6 which shows how much faster it is without bwt and mtf. This is because bwt and mtf take a lot of time to go through the entire file order the strings. This is why for larger files like the bit map, it is better to compress it without bwt and mtf as going through the entire file would take too long. Overall, our implementation does very well with smaller and medium files but does not do as well with large files mainly due to the time it takes to compress.