JOSEPH BOOTHBY
3/11/21

CS 351, HOMEWORK 4

The included files are:

- Hw4.py --- The python program for creating bitmap indices and WAH compression.
- README.txt --- The file explaining how to use hw4.py.

The following are the results from creating and compressing bitmap indices from animals.txt which is a 100,000-line file where each line is a tuple that represents an animal, its age, and a Boolean value to indicate if it has been adopted or not.

File Name	Size (Bytes)	Runs Encoded	Literals Encoded	Compression Ratio
animals.txt_bitmap	1700000			
animals.txt_bitmap_WAH_8	1557992	76429	152147	0.916465882
animals.txt_bitmap_WAH_16	1661744	14025	92647	0.977496471
animals.txt_bitmap_WAH_32	1650000	1271	50345	0.970588235
animals.txt_bitmap_WAH_64	1626128	26	25382	0.956545882
animals.txt_sorted_bitmap	1700000			
animals.txt_sorted_bitmap_WAH_8	51728	226996	1580	0.030428235
animals.txt_sorted_bitmap_WAH_16	56720	104962	1710	0.033364706
animals.txt_sorted_bitmap_WAH_32	115600	49838	1778	0.068
animals.txt_sorted_bitmap_WAH_64	227216	23604	1804	0.133656471

The first 5 files are the bitmap and the compressed indices over the unsorted data. The second 5 files are the bitmap and compressed indices over the sorted data. As you can see, the compressed size of the sorted data is significantly less than the unsorted data. Further, the word size used in the Word Aligned Hybrid (WAH) compression algorithm had a definite impact on compressed size.

Sorting the data helped with compression, ranging from a factor of 1.09 from unsorted data to a factor of 30 for sorted data. The reason for this dichotomy is that compression takes place only in the case of runs. Encoding a literal takes up more space than the uncompressed literal because an additional bit must be added in the header, therefore maximizing runs and minimizing literals results in a higher ratio of compression. Sorting the data before converting to a bitmap groups like data together which results in more consecutive bits of the same type.

Different word sizes also resulted in differing amounts of compression, ranging from a factor of 30 (sorted WAH_8 compression) to a factor of 7 (sorted WAH_64 compression). The reason for this is that larger word sizes require more consecutive like bits to result in a run and as mentioned above, maximizing runs results in a higher compression. For example, if there exists a run of 28 bits, then a word size of 8 will count that as four separate runs of 7 and compress it into a single 8-bit word. However, using word size 64, a run of only 28 bits would be counted as a literal and no compression would take place. This is clear in the data above for the unsorted bitmap. Word size 8 had $6 * 76429 = 458{,}574$ bits encoded as runs, however word size 64 had only $62 * 26 = 1612$ bits encoded as runs.

This hugely differing number of runs was partially offset because, as mentioned above, each encoded literal takes up one additional bit of space than an unencoded literal, and word size 64 encoded a smaller number of larger literals.

Another reason for a higher compression ratio with a smaller word size is less wasted bits in run encoding bytes. A 64-bit word can encode a $2^{62} = 4.6 * 10^{18}$ in a single word. This is an enormously long run, and highly unlikely to occur. In all situations where there are less than that many bits in a run, there will be wasted space in the encoding word. Smaller word sizes like 8-bit words, however, can only encode a run of $2^6 = 64$ bits, which means a run of 64 bits gets encoded with maximum efficiency.

In conclusion, different word sizes affect compression ratios, but that difference is partially contingent on how sorted the data is. For sorted data, a smaller word size is significantly more efficient than a large word size because large word sizes waste space with unused bits. For unsorted data, this gap is narrowed because large word sizes encode fewer literals which results in a smaller number of additional header bits added to the data.