Bitmap Compression Writeup

**Files Included:**

Bitmapcomp.py – This is the program for the assignment. It will take a file and create an unsorted and sorted bitmap. Then, with each version, it will perform both 32-bit compression as well as 64-bit compression on both files.

README.txt: A text file the describes the program and how to run it.

data/animals.txt: A text file with tuples of data on animals. This will be the file that the program creates a bitmap and compresses.

animals\_sorted.txt: The same data as animals.txt but the tuples are sorted lexicographically.

animals\_bitmap.txt: Bitmap created from animals.txt

animals\_bitmap\_sorted.txt: Bitmap created from sorted\_data.txt

animals\_compressed\_32.txt: 32-bit word WAH compression of unsorted\_bitmap.txt

animals\_compressed\_sorted\_32: 32-bit word WAH compression of sorted\_bitmap.txt

animals\_compressed\_64.txt: 64-bit word WAH compression of unsorted\_bitmap.txt

animals\_compressed\_sorted\_64.txt: 64-bit word WAH compression of sorted\_bitmap.txt

**Stats of Compression:**

Unsorted WAH 32 Compression

File size: 1.7 MB

Number of 0 fills: 1260

Number of 1 fills: 0

Number of literals: 50356

Sorted WAH 32 Compression

File size: 115 KB

Number of 0 fills: 41045

Number of 1 fills: 8795

Number of literals: 1776

Unsorted WAH 64 Compression

File size: 1.6 MB

Number of 0 fills: 18

Number of 1 fills: 0

Number of literals: 25390

Sorted WAH 64 Compression

File size: 224 KB

Number of 0 fills: 19740

Number of 1 fills: 3870

Number of literals: 1798

**Summary:**

WAH compression can be a great tool at shrinking file size and storing data but can also create files that end up being larger than the original file. Since literals store the same bits it read plus a header bit, when you start getting many literals, your file size starts expanding. The fact that you are also not counting runs means you will have to keep writing 32-bit literals over and over again until you start getting runs. While files have a possibility of growing, this compression algorithm shines when it reads many runs in a row. This is why sorting will give you the best results of compression.

Comparing the sorted versus unsorted version of 32-bit word compression will show you the benefit of sorting. Since the file is sorted, you will encounter a lot more runs and thus shrinking the size of the file. The unsorted compressed file is a lot larger because it stores almost 50 times as many literals. In the sorted version, instead of writing 32-bits for each literal, it uses 32-bits to count runs almost 50,000 times. That reduces the file size dramatically.

Another thing that affects the efficiency of compression is the word size. With the 64-bit words, the file is a lot larger than the 32-bit word file. This is because literals are 64-bits long and runs use 63-bits to count runs. This is really inefficient when you only have a couple of runs in a row. Using 63-bits to count to two or three is a huge space waster. The last thing that affects efficiency is when you have 64-bit word size, you will have a lot less runs. The more literals you have, the less you are actually compressing. You can see this as WAH 32 unsorted still has 1260 runs while WAH 64 unsorted only has 18. That means most of the file is the same bits as the original file but with a header bit added to it. It makes since that the unsorted 64-bit compressed file is, by far, the largest file of them all.

WAH compression is great at shrinking files as long as you choose a smaller bit word size and make sure you sort your data before compressing. This will allow you to gather more runs and minimize the number of literals you must store.