WASHINGTON STATE UNIVERSITY
VANCOUVER

CS 483 - Web Data
Assignment #2 - Bitmap Indexes

Ronald Cotton <ronald.cotton@wsu.edu>
Instructor: Ben McCamish

February 18, 2017

1 Introduction to Assignment

This assignment primarily dealt with Bitmap Indexes from a table whose fields consists of an Animal Type, Animal Age and Animal Adopted. The Type and Adopted field was converted to a grey code, while the Age field was converted in groups of 10 via binning. After analyzing the uncompressed unsorted text table vs. the uncompressed unsorted bitmap, it was discovered that the data was converted into a bitmap, in the following format:

Animal Type -	*cat*	1000000000000000
	dog	0100000000000000
	turtle	0010000000000000
	bird	0001000000000000
Animal Age -	*Age 1-10*	0000100000000000
	Age 11-20	0000010000000000
	Age 21-30	0000001000000000
	Age 31-40	0000000100000000
	Age 41-50	0000000010000000
	Age 51-60	0000000001000000
	Age 61-70	0000000000100000
	Age 71-80	0000000000010000
	Age 81-90	0000000000001000
	Age 91-100	0000000000000100
Animal Adopted -	*True*	0000000000000010
	False	0000000000000001

The data output was not binary, instead our assignment required us to output as the character one and zero (ie. '1' and '0' respectively) aiding in debugging visually. With minor changes to the bitmap code, it can be a true bitmap. Our bitmap data was aligned to optimize compression along the table's columns. In the case of our schema, the uncompressed bitmap was comprised of 16 lines of text with the column width of the file equal to the number of entries in the table.

We were required to create a total of six files from the initial CSV given. Two of the six were uncompressed bitmaps, one unsorted and one lexographically sorted. Four of the six were Bitmap Indexes utilizing Word Aligned Hybrid (WAH) compression: 32-bit unsorted, 32-bit sorted, 64-bit unsorted and a 64-bit sorted bitmap.

2 bitmap.py

Read the ***README.txt*** on how to execute *bitmap.py*.

bitmap.py takes in one argument, a filename of the CSV file to process into bitmaps. Upon completion of the python executable, **bitmap.py** generates the following files by appending the input filename with the extensions mentioned below:

Bitmap type	**file extension added to input filename**
uncompressed sorted	.orig.sorted
uncompressed unsorted	.orig.unsorted
WAH 32-bit compression sorted	.sorted.wah32
WAH 32-bit compression unsorted	.unsorted.wah32
WAH 64-bit compression sorted	.sorted.wah64
WAH 64-bit compression unsorted	.unsorted.wah64

Comparing my output files generated from animal_test to the output generated by bitmap.py, ***diff*** showed no difference. While there was no file to compare *WAH64* compression, the method of compression should be similar to *WAH32* with the exception of bit size.

Processing *animals.txt* generated the following files for **bitmap.py**:

```
-rw-r--r-- 1 ron ron 671429 Feb 18 11:49 animals.txt
-rw-rw-r-- 1 ron ron 800016 Feb 18 16:04 animals.txt.orig.sorted
-rw-rw-r-- 1 ron ron 800016 Feb 18 16:04 animals.txt.orig.unsorted
-rw-rw-r-- 1 ron ron 111712 Feb 18 16:04 animals.txt.sorted.wah32
-rw-rw-r-- 1 ron ron 130480 Feb 18 16:04 animals.txt.sorted.wah64
-rw-rw-r-- 1 ron ron 824928 Feb 18 16:04 animals.txt.unsorted.wah32
-rw-rw-r-- 1 ron ron 812720 Feb 18 16:04 animals.txt.unsorted.wah64
```

In the case of our encoding method, it's expected that the uncompressed bitmap will be larger than the original file. More bits are required to convert the bitmap as characters '1' and '0,' caused by the binning of Animal Age. Both the *animals.txt.orig.sorted* and *animals.txt.orig.unsorted* share the same filesize, as both have the same number of bits, only the order of the data has changed. The largest difference between file sizes lies in the compressed sorted and the compressed unsorted files. Unsorted, the files have too many dirty bits that WAH doesn't have a method in compressing. Unsorted, the file grew in size to write the literal bit, while 64-bit unsorted had to write half as many literal bits. When lexicographically sorted, it appears that the table was not large enough to take advantage of 64-bit compression.

Below is a table showing the compression ratios (as compared to the original file, textitanimals.txt):

File	**Percentage vs. *animals.txt***
animals.txt.orig.sorted	119.2%
animals.txt.orig.unsorted	119.2%
animals.txt.sorted.wah32	16.6%
animals.txt.sorted.wah64	19.4%
animals.txt.unsorted.wah32	122.9%
animals.txt.unsorted.wah64	121.0%

3 Pseudocode of WAH

What follows below is the pseudocode for processing one line (aka a column) of 1's and 0's into WAH format.

```
Input:
    line - one line (column) of data
    bits - number of being compressed minus one
Variables in function:
    str - the return buffer of the string
    loop_var
    line_length - the length of the line
    compressing - 0 or 1 if a value is being compressed, -1 if not
```

```
    compressing
    number_of_bits - when compressing, this is the number of bits
    to write to str

START
    initialize compressing to -1, num_of_bits to 0
    LOOP (start loop_var to 0, exit if loop_var >= line_length, when returning to loop,
          increment loop_var by bits when returning to loop)

    reading = the line from the loop_var up to bits

    if reading less than bits:
        if was compressing, write compressed buffer to str
        write rest of bits in literal to str
        return string
    if reading is equal to all zeros
        if not compressing
            start compressing zeros
            number_of_bits = 1
            LOOP
        if compressing 0's
            increment number_of_bits
            if number_of_bits > (2^bits-1)
                write compressed buffer of 2^bits-1 to str
                number_of_bits = 1
            LOOP
        if compressing 1's
            write compressed buffer to str
            start compressing 0s
            number_of_bits = 1
            LOOP
    if reading is equal to all ones
        if not compressing
            start compressing ones
            LOOP
        if compressing 1's
            increment number_of_bits
            if number_of_bits > (2^bits-1)
                write compressed buffer of (2^bits-1) to str
                number_of_bits = 1
            LOOP
        if compressing 0's
            write compressed buffer to str
            start compressing 1s
            LOOP
    if compressing:
       write compressed buffer to str
    write literal to str
    set compressing to -1
    set number_of_bits to 1
    LOOP

    return str
```