Washington State University Vancouver

Introduction to Databases - CS 351

Assignment 4

Instructor:
Ben McCamish

Overall Assignment

In this assignment you will be creating a bitmap index from a data file containing information about pets and whether they are adopted or not. Once you create the bitmap, it needs to be compressed using WAH. The data file will also need to be sorted and then another bitmap created over that. The sorted bitmap index will then need to be compressed using WAH.

For this assignment, output all the bitmaps as characters. A character on most architectures consists of a single byte. If the file says, for example, that the size is 8 Bytes, then it would actually contain 8 bits. This is fine for our purposes, but in practice bitmaps are created and queried using actual binary files and bits to increase their speed and reduce the size. **You may not use any libraries for bitmap creation or compression**. Also note, your design of WAH should be independent of word size. That is, your code should easily be able to compress with word size 16, 8, etc. I will be testing on 8, 16, 32, and 64 bit word sizes.

Sample Of Database

There are three attributes in the data file, *Animal*, *Age*, and *Adopted*, in that order. I have include two files that you may perform all of your compression and bitmap index creation on in order to test. The larger one titled 'animals.txt' will be good to make sure you are compressing runs correctly. Don't forget to sort the raw data before some of your tests to maximize the runs.The other file, 'animals_small.txt' is a smaller file to test your code on.

Animal	Age	Adopted
Cat	12	True
Dog	68	False
Dog	33	False
...	...	...

The attributes have the following domains:

- **Animal:** ['cat', 'dog', 'turtle', 'bird']
- **Age:** [1,100]
- **Adopted:** [True, False]

Code Notes

For testing, I will be importing your code as a module into my own test bank. Ensure that I only need to import a single module and use the specified functions listed below. The files you generate must follow the naming convention specified below:

```
inputFile_<sorted>_<compression>_<wordSize>

Where: 'inputFile' is the specified input file of the original data. '<sorted>' is optional
and is only included if the original data was sorted. '<compression>' contains the method
used for compression. '<wordsize>' contains the word size used for the compression.
```

Example of file output name when creating a bitmap that is sorted from the input data of 'animals.txt':

```
animals.txt_sorted
```

Example of compressing said bitmap index with WAH using 32 bit words.

```
animals.txt_sorted_WAH_32
```

Create a Bitmap Index (20%)

Your interface should be as follows:

```
create_index(input_file, output_path, sorted)

Where: 'input_file' is a file that you will use to create the bitmap index. 'output_path' is the
destination directory for your output bitmap file. It must be a regular file with no suffixes
(.txt, .c, etc). 'sorted' is a boolean value that specifies whether your data will be sorted.
```

Unsorted Bitmap Index

The first bitmap you should make should have the same order as the data file. Your bitmap index should have a column for each possible animal, in the order represented in its domain. Following the four animal columns, the bitmap should have 10 columns that represent the bins for the *Age* attribute. Each bin should be 10 long. For example, 1-10, 11-20, 21-30, etc. The last two columns in your bitmap should represent True and False in that order for the *Adopted* attribute.

Sorted Bitmap Index

Now sort the data file lexicographically and create the bitmap over it again. You should now have two bitmap indexes over the dataset, one created over the unsorted data file and another created over the data file that has been sorted.

Compressing the Bitmap Index

Patents for both methods are expired, so use guilt free for tips on how yours should operate. However, I simplified some of the details, so implement from the patents with caution. Your compression methods should operate as discussed in class. Your interface should be as follows:

```
compress_index(bitmap_index, output_path, compression_method, word_size)

Where: 'bitmap_index' is the input file that will be used in the compression. 'output_path'
is the path to a directory where the compressed version will be written using the naming scheme
specified above. 'compression_method' is a String specifying which bitmap compression method
you will be using (WAH, BBC, PLWAH, etc). 'word_size' is an integer specifying the word size
to be used.
```

Word Aligned Hybrid (50%) WAH Patent

You must implement Word Aligned Hybrid (WAH). It should be capable of operating on any word size specified. If the bitmap index has a set of bits at the end, then you may encode it as a literal and pad the rightmost side with 0s. A single run of either 1's or 0's should be encoded as a run and not a literal.

Bit-aligned Bitmap Compression (Extra Credit: 1 Quiz) BBC Patent

As extra credit, you may add the BBC method to your module. This will use the same interface and file naming conventions as WAH. For the dirty bits you may count from the leftmost bit. You will be using 'one-sided' BBC compression which only compresses 0s. Remember, with BBC we don't use words. This means that your interface shouldn't care about the value passed as word size if the method is 'BBC'.

For example:

```
01000000 would be encoded as 00010001 since the dirty bit is in position 1.
```

Writeup and Comparison (30%)

- Writeup a summary of all the files that you included. This includes a small description for each file.
- Then you must compare the size of the bitmap indexes and compressed versions on the large test file. Write an analysis on why you think they are different size. Did sorting help with the compression and by how much? Did different word sizes have different compression ratios and why do you think that is?
- In addition to your analysis, include the number of fill words and literal words that were compressed for each file.

What to turn in (in a zip on Canvas):

- All code (well commented, if not self-documenting)
- Make sure your interfaces and file naming schemes are correct.
- README.txt on how to run your code (detailed if required).
- **Note:** Your code when run should produce all the required output. I WILL NOT be changing any code inside the python files.