

## Program #2: Extendible Hashing

*Due Date: September 23<sup>rd</sup>, 2008, at the beginning of class*

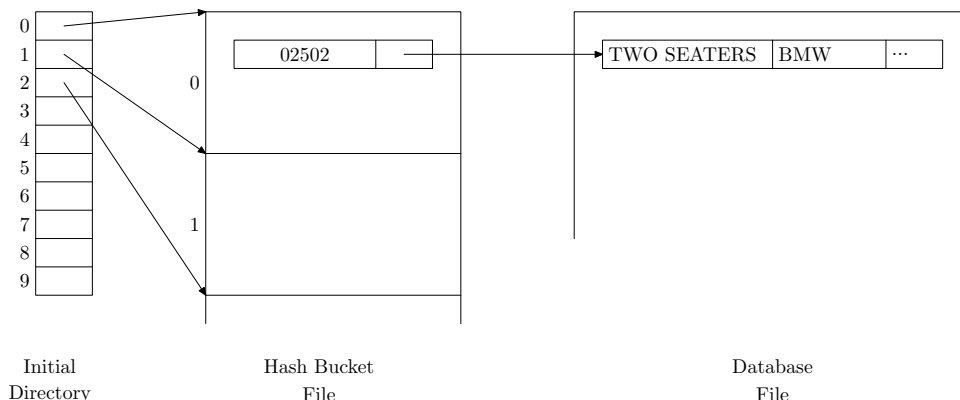
Special Option:  
You MAY choose to do this assignment with a partner (i.e., in teams of two).

**Overview:** In class we talked about Extendible Hashing under the assumption that the directory is indexed on bit patterns derived from the search key. This isn't a law; if the situation permits, we can index based on any part of the key we wish, even full characters.

**Assignment:** Create an extendible hash index for the binary file of fuel economy records that your Program #1 created, and use the index to satisfy some simple queries.

Obviously, before you can create the index, you need to have already converted the text file supplied with Program #1 to a directly-accessible binary format. This is necessary so that we can make use of the direct access feature of binary files to speed up our access to the data. Hopefully your Program #1 (or your partner's, if you have one) created a suitable binary file; if not, you'll need to repair it (or your partner's) before starting on this assignment.

The first task will be the creation of the index. In this case, you will be using ANL FL CST (the 25th field) as the key. ANL FL CST values will not exceed five digits any time soon (hopefully!), so for the purposes of indexing we will treat all such values as five-digit numbers (thus, an ANL FL CST of 3400 would be treated as 03400). Because of this assumption, we have a predictable key size, and we can easily index into our extendible hash directory with full characters (digits) rather than binary bits. Thus, the initial directory will have 10 entries, labeled 0 through 9, as shown in the diagram below:



The directory is holding “pointers” to the buckets in the Hash Bucket File. Initially, the bucket file is empty; a bucket is added when an index entry needs to be stored. For this program, size a bucket so that it can hold a maximum of 50 entries, where each entry is just a 5 character string and a “pointer” to the Database File. These “pointers” are actually just byte offsets into the files. The offset of the first bucket is 0. The offset of the  $n^{\text{th}}$  bucket is  $n$  times the size of a bucket. A similar calculation applies to the location of a record in the database file.

What's in a bucket? A bucket is essentially just an array of ‘slots’ (key–pointer pairs) and perhaps a count of how many of those slots are currently in use. (Note that this count could also be held in the directory if desired.) Recall that when a bucket is full and a new addition arrives, we split the bucket, and perhaps ‘double’ the directory as well. Creating the hash bucket file as a binary file makes the splits easier to handle.

The second task is to process a simple variety of query using your index. Prompt the user to enter an ANL FL CST value prefix (for example, “034”) and return to the user the complete data records associated with all keys having the given prefix (03400, 03401, etc.), and a count of the number of data records returned. Naturally, you are expected to use the extendible hash index to find all matching records, rather than searching the entire database file. Be sure to produce the record list in a clear, organized, well-labeled format. (Feel free to reuse code from Program #1.)

**Data:** The data that this program needs to construct the index is the binary file your Program #1 program created from `2008_FE_guide.csv`. As mentioned above, the index will be constructed on the ANL FL CST field of the binary file.

For the second part of the assignment (the queries), we will not be supplying sample queries; we expect that you can perform your own testing. Don’t forget to print the quantity of records returned by each query.

**Output:** Run your program on each of the queries provided in the Data section, and print out the results. You are to print all of the database records with keys that match the given query prefix. Be sure to produce the record list in a clear, organized, well-labeled format. (You may reuse output code from Program #1.)

**Hand In:** Submit your completed program file(s) using the `turnin` facility on lectura. The submission folder is `cs460p2`. Instructions are available from the document of turnin instructions linked to the class web page. Name your main program source file `Prog2.java` so that we don’t have to guess which files to compile, but feel free to split up your code over additional files if you feel that doing so is appropriate.

Remember: You will need to be sure that your program works successfully on lectura before you submit it.

**To Partner or Not to Partner:** As mentioned at the top of this handout, you have the option to work on this assignment in teams of two class members. Before you rush around the room trying to secure your favorite “code monkey” as a partner, read on.

If you work with a partner:

1. Include both names in your program documentation (so that you both get credit!).
2. Your team will have to store the directory *within* the hash bucket file. It cannot be in a separate file.
3. Your team must write a separate program to perform querying. This second program will have to begin by loading the directory before processing any queries.
4. The index creation program is to conclude by displaying the following information about the index: The global depth of the directory, the number of directory entries pointing to buckets, the number of buckets in the hash bucket file, the average bucket occupancy, and the most common ANL FL CST value.

If you work solo:

- You **do not** need to store the directory into the hash bucket file, and you **do not** need to print any statistics, and you should write just **one** program. This means that your index creation program is also your querying program; create the index, then immediately begin getting queries from the user and answering them.

Obviously, the point of this is to balance the workload a little more equitably. Even so, I think that (personality and scheduling conflicts aside) it will be easier to do this as a team. But, the choice is yours.

### Other Requirements and Hints:

- Start early! This is a large program; it will take a lot of time to complete.
- Work modularly. For example, you will undoubtedly want to create an extendible hash index class.
- When you test your code, use only a subset of your binary file, not the whole thing. Create a test binary file that has just a few records, and try your code on it first. If everything seems OK, then try it on the complete file.
- Remember to comment your code according to the style guidelines. In particular, please use the block commenting templates to ensure that you have all of the information we require. Include both partner’s names in the documentation!