

INDEXING THROUGH BUCKET HASHING

For this assignment, you will implement an index to a database of student records through bucket hashing (pp. 309-310 of the textbook). A separate database will store the actual student records while the index file will contain pairs of key values and pointers (to the database file). This index file will be structured as a hash table that groups slots into equal-sized buckets (one bucket per hash value). Slots in a given bucket would, in effect, store entries that hash to the same position. There will also be a common overflow space in this index file, to accommodate the cases where the number of entries that hash to the same position is greater than the bucket size. This overflow space will occur right after the hash table (overflow entries are simply appended to the index file).

The efficiency of this index depends on the hash function and the size of a bucket. The key to a student record is a 9-digit social security number and the hash function will be based on the last d digits of this key; that is, $h(K) = K \bmod 10^d$.

The program will first load from a text file a list of student records from which the initial database and its index will be built. To test the correctness (and efficiency) of the index, the program will then process a sequence of search and insert commands, from a command file. The program will be invoked as follows:

buckethash <datafilename> <dbname> <s> <d> <cmdfilename> <outfilename>

The parameters are described below:

<**datafilename**>: the name of the file containing the list of student records. The file will contain several text lines, one student record per line. Each line will have 6 space-separated fields: social security number (9 digits), last name (at most 15 letters), first name (at most 15 letters), year (1 digit), major (at most 4 letters), and email address (at most 20 characters). This means a record will require at most 64 bytes of space.

<**dbname**>: the name of the database and index files. These files will be created by the program, and will have a .dat and .idx extension for the database file and index file, respectively. For example, if <**dbname**> is "student", the database file will be named "student.dat" while the index file will be named "student.idx"

<**s**>: the size of a bucket in the hash table.

<**d**>: the number of rightmost digits used by the hash function.

<**cmdfilename**>: the name of the file containing the commands to be processed. Each line in this file will be a command and will either be of the form: **find** [ssn] or **add** [record details]. [ssn] is a 9-digit social security number while [record details] take the same format as a line in the list of student records.

<outfile>: the name of the file where the output is sent. There will be one output line per command. After processing all commands, a summary line will also be reported, indicating the size of the index file and total number of hash table accesses made. Output details are described below.

For a find command, the message

record found: [record details]. [access count] hash table accesses.

will be output if the record exists in the database; if it does not, output the error message:

[ssn] not found. [access count] hash table accesses.

For an add command, output the message

[record details] added. [access count] hash table accesses.

unless a record with the same **[ssn]** already exists in the database; otherwise, output the error message

[ssn] already in database. [access count] hash table accesses.

Note that **[access count]** indicates the number of accesses performed on the hash table for the given operation. After processing all commands, print a line summarizing index performance:

Size of index file in bytes: **[length]**. Total number of hash table accesses: **[total access count]**.

This assignment is an exercise in hashing and binary file I/O. The database file should store fixed-length records (64 bytes), in sequence. New records are simply appended to this file. The index file will contain the bucket hash table and overflow space whose entries each consist of a pair of integers: the social security number (key), and a record number that will allow your program to access the corresponding record in the database file (multiply it by 64). These integers should be stored in binary form (an int will typically occupy 4 bytes). Note that, upon creation, the index file should be large enough to store at least $2 * \langle s \rangle * 10^{\langle d \rangle}$ integers. It may actually store more depending on the number of entries that end up in overflow space.

You will be asked to execute your program for different values of **<s>** and **<d>**. Assuming the same set of input files, the same output should result (except for access count information) but execution time ought to be different. You may verify this using the Unix *time* command.

Sample input and output files (as well as values for **<s>** and **<d>** that you should test) will be made available in the website by Friday, 1 August 2003. The program is due on 5 August 2003 11:59 pm. You are to submit a tarred, gzipped file (.tar.gz) containing all source code and a makefile. Make sure your project compiles in the Mandrake Linux environment. As before, refer to the course web site for Curator system instructions, program documentation guidelines, honor code policies, and the pledge sheet that you must include in your main source file.