COSC 311 PROGRAMMING PROJECT #2 Random access File I/O to implement hashing.

Distributed: 4/4/2017 **Due**: 4/18/2017

You will implement open addressing hashing using linear probing on the hard disk (in a file).

Record format

Each element of the hash table contains a record. Each record is formatted as follows:

20 bytes (for 10 Unicode characters)
2 bytes (for String termination character)
1 byte (data information byte: dirty bit, empty bit)
The data are laid out on the record as shown above.

Note! You may use an object to store the data, but only if the size of the object is exactly 23 bytes.

Note!! It does not matter what the size of an input String is. You must pad or truncate (on the right) the String data to make every stored data item exactly 23 bytes in total.

"Meta-data"

The meaning of the byte that stores the dirty bit is:

0b0000 there is no data present
0b0001 dirty — the data is invalid. I.e., this is a tombstone
0b0100 clean data — there is data present and the data is valid
0b0101 dirty data – there is data present, but the data is dirty.

Note! Only the bottom 4 bits of the byte is shown. The top four bits are, obviously, all 0.

The basic hash table

The hash table is a random access file comprising records. You must create the initial hash table in the file by setting up 16 records: *The hash table is initially empty and is size 16*.

Use Java Object's `hash()` function for the hash function.

Operations on the hash table

You will insert and delete several records. When the hash table reaches 50% full, you must

pause reading input and applying operations,
create a new hash table doubled in size,
rehash the old data,
delete the old table.

After every insertion, check to see if your table has exceeded capacity (50%).

Deletions will not cause contraction of the hash table, not even if the table is empty.

The operations are contained in a char stream file named `input.dat`. There are three kinds of operations in that file:

input (String)
delete (String)
printTable()

Input operation

`input( String str)`

(1) Read the String `str` from charstream `input.data`

(2) Output "Input `str`" to the console (obviously using the value of `str` in the output statement)

(3) Modify the String `str` to make it 23 bytes following the format given above. Note, the final byte is 0x04 (data present, not dirty).

(4) hash the input parameter s to find the hash value. Then modify the hash value to find the seek location.

(5) Seek to the computed location.

(6) Read the record at the current seek position

(7) If the record location is marked 'dirty' 0x01 or 0x05, or marked 'empty' 0x00, then insert the data,

else repeat linear probe until data can be inserted.

(8) After the record been inserted, check the table capacity.

If the table is over capacity:

Output "table size `n` is overcapacity. Rehashing" (obviously, give the current value of `n`)

Rehash table

Delete any old file that will no longer be used

Delete operation

`delete ( String str )`

(1) Read the String `str`

(2) Output "Delete `str`" to the console. Give the current value of `str` in the output

(3) Hash the data, compute the seek location

(4) Use linear probe by reading records, testing for value match or testing for dirty bit until either: data is found — mark it dirty

or: data is not present (probed to empty record).

PrintTable operation

(1) Determine the size of the file in bytes and in number or records. Output the size(s) of the file.

(2) Position the file cursor at the start of the hash table.

(3) On a separate line, output the String data, followed by the data information byte, in the order as recorded in the file. The data information byte must be given as a bit string. If the record is empty, output blanks for the String data followed by the data information byte.

(4) Repeat (3) for the entire file (there will be 32 entries).

Testing the data information byte:
Suppose you have put the data information byte into the unsigned byte variable named `info`.

- The data is dirty: `info && 0x01`
- The data is present and clean: `!dirty(info) && (info && 0x04)`
- The record is empty: `info == 0x00`

Setting the data information byte
Suppose you have put the data information byte into the unsigned byte variable named `info`.

- Set the dirty bit: `info = info || 0x01`
- Clear the dirty bit: `info = info && 0xFE`
- Set the data present bit: `info = info || 0x04`
- Clear the data present bit: `info = info && 0xFB`

Random access file I/O
See the description of the Java class: RandomAccessFile:
http://docs.oracle.com/javase/7/docs/api/java/io/RandomAccessFile.html

A super elementary example, with annoying advertisement, is given here:
https://www.java-tips.org/java-se-tips-100019/18-java-io/1998-how-to-use-random-access-file.html

Another example is given here: https://examples.javacodegeeks.com/core-java/io/randomaccessfile/java-randomaccessfile-example/