CS351 DATA ORGANIZATION AND MANAGEMENT

Programming Assignment #2

Transactions on Static Hashing in JAVA

Due Date: November 16, 2009, 23.59

Table of Contents

1. Assignment Definition

In the first assignment, you implemented static hashing using "RandomAccessFile" class in Java. In that implementation, you only inserted records from an input file into a hash file using static hashing. In this (**second**) assignment, you will implement another program using "RandomAccessFile" class again. Instead of modifying "**StaticHashing.java"**, which you implemented in the first assignment, you will implement another program called **"Transactions.java"**, in which you will make transactions on a hash file given to you, such as addition of a record, modification of a record and deletion of a record. This process will be a **batch processing**, which implies that there is no interaction with the user while the program is being executed.

In this assignment, you will not ask user to enter prime area and overflow area numbers. You will use 20 for prime area bucket number and 10 for overflow area bucket number. (These numbers will be **constant**; they will not change during the tests). So you will use 20 in the hash function h(x), **x mod 20** (x is the StudentID)). So bucket numbers will be 0, 1, 2… 19 in prime area; 20, 21… 29 in overflow area. Bkfr (Bucket Factor) will be 1 again which indicates there will be only one record in each bucket. Each bucket size and record size will be 20 bytes as it is in the first assignment.

In this assignment, you will be given 4 files:

1) "Students.txt": In this file, you can see the list of student records which were inserted into HashFile.txt by using the program implemented in the first assignment. (This file is not important and you will not use it in your implementation, it is only for you to see which records are already inserted into the hash file.)
2) "Overflow.txt": In this file, you will read the value of the overflow pointer after inserting all records in Students.txt into HashFile.txt (It is written 540 in it, so it means overflow pointer points to 27th bucket initially) and you will update its value at the end of program with new overflow pointer value.

3) "Transactions.txt": In this file, the transactions will be listed. There will be only one transaction in each line. Each field will be separated by 1 blank character ( ), so you can use StringTokenizer to separate the fields. For insertion, there will be 4 fields: Transaction field ("A"), StudentID field, StudentName field and StudentDept field. For modification, there will be 3 fields: Transaction Field ("M"), StudentID field and StudentDept field. For deletion, there will be 2 fields: Transaction Field ("D") and StudentID field. There will be 3 types of transactions and the letter at the beginning of each line indicates the type of transaction:
    a. Addition: "A 200032 Ali CS" This is an example of addition transaction. The letter "A" indicates it is an addition transaction. "2000032 Ali CS" is the record which will be inserted into given HashFile.txt.
    b. Modification: "M 200007 IE" This is an example of modification transaction. The letter "M" indicates it is a modification transaction. "2000007" is the StudentID field of the record which will be modified in HashFile.txt. You will just modify the department of the record. "IE" indicates department of that record will be modified as IE in HashFile.txt
    c. Deletion: "D 200006" This is an example of deletion transaction. The letter "D" indicates it is a deletion transaction. "2000006" is the StudentID field of the record which will be deleted from the given HashFile.txt

4) "HashFile.txt": This file was generated by using "Students.txt" and the program implemented in the first assignment, which means this file is an output of the first assignment using **20 for prime area** and **10 for overflow area**. (You will not create this file; it will be given to you as records already inserted using the hash function). But you will perform transactions on this hash file. For ex: if there is an addition transaction, you will insert that record into this HashFile.

*Notification: The only files which will be used during grading process are "Overflow.txt" and "HashFile.txt". Initial content of these two files will be constant, so that during grading process, we will use the same two files given to you. You will update these two file in your program appropriately. But "Transcations.txt" is a sample file for you to test your program. During the grading process, different transactions will be tested on your program, not exactly the ones in "Transactions.txt". "Students.txt" is given for you to see which records are already inserted, which will not be used during the grading process and in your program.

2. Structure of Hash File (HashFile.txt)

The structure of a hash file will be same as it is in the first assignment. There will be 4 fields for each bucket: StudentID is 6 bytes, StudentName is 8 bytes, StudentDept is 2 bytes and OverflowAreaLink is 4 bytes (20 bytes in total). But as it is in the first assignment, StudentName field will be more than 0 bytes and **maximum** 8 bytes. If StudentName is "Emre", which is 4 bytes, the rest 4 byte of StudentName field will be empty. Similarly, for OverflowAreaLink field, it will be maximum 4 bytes. If OverflowAreaLink is "24", which is 2 bytes, the rest 2 bytes of link field will be empty.

200040	Emre	CS	24	200001	Ali	IE	20	-1	………………………..

In the figure above, you can see the first 2 buckets (bucket 0 and 1) of HashFile.txt. In total, there are 30 buckets in HashFile.txt each has 20 bytes (20 bucket in prime area and 10 bucket in overflow area). You will make transactions on this file like inserting a record in to a bucket, deletion of a record from a bucket and modification of a record in a bucket. If you look at the HashFile.txt given to you, you can see that there are already inserted records into both overflow and prime areas. But there are also some buckets which are empty (-1 indicates empty for that bucket as it is in the first assignment) in both prime and overflow areas. At the end of your program, this HashFile.txt will be modified with the transactions given in Transactions.txt.

3. Overflow Pointer

In this assignment, again, you will have an overflow pointer, which will point the first empty bucket in the linked list of the overflow area. The value of the overflow pointer will be read from "Overflow.txt" at the beginning of the program that is 540 if you look at "Overflow.txt". So, the overflow pointer will initially point to the 27th bucket. When you insert any record into a bucket in the overflow area or delete a record from a bucket in the overflow area, this overflow pointer will be updated. When there is no empty bucket in overflow area, which means overflow area is full, you will set the value of the overflow pointer to 0**.** At the end of the program, you will write the updated value of this pointer into the same file **"Overflow.txt"**.

4. Implementation

You will have only one java class, **"Transactions.java"** which will make these operations in order:

- read the value of the overflow pointer from "Overflow.txt" and take the overflow pointer to that address
- read transactions line-by-line from "Transactions.txt"
- perform each transaction by applying transaction rules below onto "HashFile.txt" which has records already inserted ("HashFile.txt" will be modified by transactions listed in "Transactions.txt")
- print the values below on the screen at the end of the program:
  - Total number of transactions (number of transactions which were read from Transaction.txt file)
  - Total number of erroneous transactions (number of unsuccessful transactions)
  - Total number of successful addition transactions
  - Total number of successful modification transactions
  - Total number of successful deletion transactions
- write the final value of the overflow pointer into "Overflow.txt"

Basically, your program will take "Oveflow.txt", "Transactions.txt" and "HashFile.txt"; will apply the transactions in Transactions.txt onto HashFile.txt and will update the value of the overflow pointer in Ovreflow.txt.

5. Useful Methods (from Previous Assignment)

You will use **"RandomAccessFile"** class of Java in this assignment for read and write operations. So you will create the random access file in read/write mode ("rw"). There are a few types of read and write methods in this class. But, if you use **"writeBytes ()"**http://msdn.microsoft.com/en-us/library/aa988623(VS.80).aspx method to write records into buckets, you can see the inserted records in the output file during your tests.

For read operations, you can use **"readFully ()"**http://msdn.microsoft.com/en-us/library/aa986251(VS.80).aspx method.
There is a kind of cursor called the ***file pointer*** of RandomAccessFile class. Input operations read bytes starting at the file pointer and advance the file pointer past the bytes read. If the random access file is created in read/write mode, then output operations are also available; output operations write bytes starting at the file pointer and advance the file pointer past the bytes written. The file pointer can be read by the **getFilePointer** () method and set by the **seek()** method.

You will use **"seek()"**http://msdn.microsoft.com/en-us/library/aa987814(VS.80).aspx method of RandomAccessFile to write the record into the bucket defined by hash function.

When you reading from input file, you can use "s.trim()" method (s is a Sting variable) to **eliminate blank characters**. For e.g. if you read "20 " and store in a String variable s, if you use s.trim(), it will return you "20".

When you want to convert an Integer variable to a String variable, you can use **Integer.toString(i),** where i represents the Integer variable which will be converted to String.

When you want to convert a String variable to an Integer variable, you can use **Integer.parseInteger(s),** where s represents the String variable which will be converted to Integer.

6. Transaction Rules

1) Insertion Rules:

For insertion of records, you will use the same process given in the first assignment.

"A 200032 Ali CS" is an example addition transaction. The letter "A" indicates the transaction type which is an addition transaction. You will insert the record "200032 Ali CS" into your hash file. Each field is separated by one blank character ( ) for your easiness to separate into 4 fields. You will use the same insertion process it is told in the first assignment, in which you will use a hash function h(x), **x mod 20** where x is StudentID to define into which bucket you will insert the record. After calculating the appropriate bucket by hash function, there are some types of insertion:

a. Calculated bucket is empty, so you can insert the record without checking other things such as StudentID uniqueness
b. Calculated bucket is full, but OverflowAreaLink field is 0 which means, there is no bucket in the overflow area pointed by calculated bucket. So, insert the record into the first empty bucket pointed by the overflow pointer.
c. Calculated bucket is full and OverflowAreaLink field is not 0. Then you have to iteratively search the linked list to find the last bucket of the linked list. Then insert the record into the bucket pointed by the overflow pointer and update OverflowAreaLink field of the last bucket in the linked list with bucket number of filled bucket.

*For option b and c where bucket in prime area is full, you have to check whether overflow area is full or not before insertion of record. If it is full, you don't need to go through on linked list; you just give an error message on the screen: **"Overflow area is full, record couldn't be inserted".** And this transaction will be an erroneous transaction.

*In a hash file, every StudentID is unique, so when you insert a record into the hash file, you have to check whether that StudentID exists. But StudentName and StudentDept fields are not unique. So you will not check uniqueness of these two fields. **For ex: if the record will be inserted is "204020 Emre IE" or "204020 Ali IE" or "204020 Ali CS" and we have a record already inserted as "204020 Ali CS", for both three option, you will not be able to insert those records because StudentID already exist.**

For options 2 and 3(For options b and c)*; before insertion of record, you have to go through on linked list to check whether another record with same StudentID is already inserted or not. You don't have to check all records from the beginning of hash file because if there is another record with same StudentID, it has to be somewhere in the linked list (all records will be inserted by using hash function so for e.g. you can't insert). If you found a record with same StudentID, give an error message on the screen: **"Duplicate record, record couldn't be inserted". And this transaction will be an erroneous transaction.

After each insertion transaction, don't forget to increment total number of transactions. For each successful transaction, also increment total number of successful addition transactions and for each unsuccessful transaction, also increment total number of unsuccessful addition transactions.

2) Modification Rules:

"M 200007 IE" is an example modification transaction. The letter "M" indicates the transaction type which is a modification transaction. For this given transaction, you will modify the StudentDept field of the record, whose StudentID is 200007, with "IE" in the hash file. Each field is separated by one blank character for your easiness to separate into 3 fields. You will use the same insertion process it is told in the first assignment, in which you will use a hash function h(x), **x mod 20** where x is StudentID to define into which bucket you will insert the record. There are some issues you have to check before modification:

a. For the given example transaction above, before modification of the record, firstly you have to find the bucket which consists the record whose StudentID is 200007. After calculating the appropriate bucket with hash function, you will check whether SudentID is 200007. If it is not, you will again go through linked list as you do in insertion until you see a "0" in any bucket in the linked list which indicates it is the last bucket in the linked list. If you can't find any bucket in the linked list whose StudentID is 20007, this transaction will be an erroneous one and you will give an error message on the screen: **"Non-existent record, record couldn't be modified"**

b. For the given transaction above, you have to check if StudentDept is already “IE”. If record’s StudentDept is already “IE”, you will not be able to perform transaction and this transaction will be an erroneous transaction. In addition, you will give an error message on the screen: **“Same department name, record couldn’t be modified”**.

After each modification transaction, don’t forget to increment total number of transactions. For each successful transaction, also increment total number of successful modification transactions and for each unsuccessful transaction, also increment total number of unsuccessful modification transactions.

3) Deletion Rules:

“D 200006” is an example deletion transaction. The letter “D” indicates the transaction type which is a deletion transaction. For this given transaction, you will delete a record whose StudentID field is 200006 from its bucket in the hash file. Each field is separated by one blank character for your easiness to separate into 2 fields. Deleting a record from a bucket means, you will write “-1” for StudentID field of that bucket. StudentName field and StudentDept field will be empty as you initialized in your first assignment. OverflowAreaLink field will be different for each possibility. The possibilities:

a. If the record to be deleted is in prime area and OverflowAreaLink field is “0” of that bucket. Then you will just delete (empty) bucket and OverflowAreaLink field will remain “0”
b. If the record to be deleted is in prime area and OverflowAreaLink field is not “0”, which indicates there is some bucket in overflow area linked to this bucket. Then firstly you will delete (empty) the record from the bucket in prime area, and then you will carry (copy) the record in the first bucket of the overflow linked list to emptied bucket. And the bucket where its record is carried to the emptied bucket will be the first empty element of the overflow linked list (look notifications part). The overflow pointer will point this bucket and this bucket’s OverflowAreaLink field will be the bucket number of bucket pointed by the overflow pointer previously. (We simply insert the emptied bucket from head to the overflow linked list and for e.g. OverflowAreaLink field will be 13 if the overflow pointer points the address 260)
c. If the record to be deleted is in overflow area instead of being in prime area, and its OverflowAreaLink field is “0”, which indicates the bucket consisting this record will be the last bucket in the linked list. Then you will just delete (empty) bucket by writing “-1” for StudentID field and leaving StudentName and StudentDept fields empty. Then you will insert this emptied bucket into the overflow linked list as its first (head) node. Now, the overflow pointer will point this bucket and OverflowAreaLink field of this bucket will be the bucket number of bucket pointed previously by the overflow pointer.

d. If record to be deleted is in overflow area and OverflowAreaLink is not “0”, which indicates that bucket is somewhere in the middle of the linked list. You will repeat steps in part c; but this time the only difference is, you will have to find the previous bucket comes before the bucket to be deleted in the linked list and next bucket comes after the bucket to be deleted in the linked list. You will update OverflowAreaLink field of previous bucket with bucket number of next bucket, so they will be linked to each other. And the emptied bucket will be inserted to the overflow linked list from the head.

*Don’t forget to update the value of the overflow pointer when some bucket is emptied in the overflow area.
*You have to go through the linked list to find the bucket to be emptied. You don’t have to check all records from the beginning of hash file to find the bucket to be deleted because if the record with given StudentID exists, it has to be somewhere in the linked list.
*After going through the linked list, if you are at the last bucket of the linked list and you still didn’t find the record with given StudentID, it means a record with a given StudentID does not exist in the hash file. This will be an erroneous deletion transaction. Then you will give an error message on the screen: **“Record with given StudentID does not exist”**

After each deletion transaction, don’t forget to increment total number of transactions. For each successful transaction, also increment total number of successful deletion transactions and for each unsuccessful transaction, also increment total number of unsuccessful deletion transactions.

7. Notifications

* **Notice that each “write” and “read” operation advances the file pointer**, so especially after read operations, don’t forget to reposition file pointer to the beginning of the bucket.
* When you are reading strings from input file, for StudentID field, don’t forget to parse it into integer before using them in the hash function.
* There is one blank between each field in the transaction file. That’s, each field is separated by one blank.
* When you update the OverflowAreaLink field of a bucket, you will write the bucket number of the bucket where the new record is inserted, not the address of that bucket. (For ex. you will write 11 into OverflowAreaLink field not 220(11*20), but when you travelling through buckets by using these links, don’t forget to multiply it with 20, which is the size of each bucket.)
* writeBytes(“Emre”) costs 4 bytes, one byte for each character and it advances file pointer 4 bytes. And also during the readFully(x) advances the file pointer past the bytes read.
*****Buckets in the overflow area will be linked to each other in the hash file initially. After each insertion and deletion, their link will be updated. For insertion, bucket will be removed from link and the overflow pointer will be taken to the next empty bucket in the linked list. For deletion, deleted (emptied bucket) will be the head bucket of the file pointer, which means you will insert that empty bucket from head of the linked list and the overflow pointer will point that bucket. The reason of inserting the bucket from head is we have only one overflow area pointer which points the head of the linked list, so finding to the tail of linked list for each deletion transaction will be time-wasting. When you insert the bucket from head, you

will just take the pointer to that bucket and update this bucket's OverflowAreaLink field with the overflow pointer's previous value.
*You will not check the validity of the input files; such as each StudentID field will be exactly 6 bytes and each StudentDept field will be exactly 2 bytes. But StudentName field can be less than 8 bytes but it will never be 0 bytes. When you reading from input file, you can use "s.trim()" method (s is a Sting variable) to **eliminate blank characters**. For e.g. if you read "20 " and store in link variable, if you use link.trim(), it will return you "20".
*If you use methods **Integer.parseInteger() or Integer.toString(),** notice that you have to use proper parameters in these methods. For ex. If you are using Integer.parseInteger(i) method where i represents the String variable which will converted, i must not contain blank characters. (If you enter "20 " as i, method will not return 20.)
*The content of given files HashFile.txt and Overflow.txt will be constant, but content of Transactions.txt will be different in testing your program. So in testing process, content of Transactions.txt will be different than given one.
*For your easiness, transactions in Transactions.txt is given in addition not mixed order, they are given firstly all additions, then all modifications and finally all deletions. And these **transactions test each possibility explained above**. After finishing your program, test your program by reordering transactions like addition, deletion, modification, deletion, addition, modification…
***There will be two types of linked list.** First is, when two or more records have same hash function result, the first inserted will be in the prime area and others will be in the overflow area and each will be linked to another bucket. Second is overflow linked list, there will be a linked list between empty buckets in overflow area. The head bucket will be pointed by the overflow pointer and after deletion and addition transactions, links will be updated appropriately.

8. Testing Details

A sample transaction file is posted to the website to help you test your program. This will not be the transaction file that will be used in grading your program. But given files, HashFile.txt and Overflow.txt will be the files which will be used in grading process, too. So, the content of these files be constant. While the transaction file provided should be useful, you should also do testing on your own transaction file to ensure that your program works correctly. *During your submission, you will only submit "Transactons.txt" which will be implemented by you.* You will not submit Overflow.txt and HashFile.txt. During tests, we will put "Overflow.txt" and "HashFile.txt" near your Transactions.java.

9. Submission Rules

You will submit only one file: **"Transactions.java"** java class. You will put only this java class into the winrar file will be *StudentID_Name_Surname.rar* (20491000_Emre_Celik.rar). (Don't use Turkish characters) Please don't send the project file of programs like Eclipse or Visual Studio. You will just put **"Transactions.java"** which is nested in "**src**" directory of these programs. The ones who do not obey the submission format will be penalized for 5 points. Late submissions will not be accepted.

Upload Page: http://www.cs.bilkent.edu.tr/~yenicag/