CS 351 – DATA ORGANIZATION AND MANAGEMENT

FALL 2010 - Programming Project #2

Linear Hashing File Reorganization

Due: December 17, 2010 at 11:59 PM

1. The Objective

In this programming assignment, you are asked to implement a JAVA console application, which simulates a linear hashing file environment. The aim of the assignment is to make you understand linear hashing and have fun with it. Additionally, you will provide a short report (max. 1 page) that explains your algorithm to solve the problem. **Please read the entire assignment before starting implementation.**

2. Problem Specification

A linear hashing file structure with an unknown load factor is given in an input file. First, your program should find the load factor. Then, it must reorganize the linear hashing structure with a new load factor. Note that new load factor is also given to your program as an input. Finally, after reorganization, new records are added to the structure. Each is record is represented by an integer. At the end, your program would write the final linear hashing structure to the specified output file. The expansion of the linear hashing file structure is done as explained in Salzberg's book. (also see solution to Quiz #3). It means that you will come to the load factor, then, add ⌈(load factor) x (number of records per bucket)⌉ number of records and expand the file.

3. Input Specification

- **Initial Linear Hashing Structure File:** In this file, records that are in the same bucket are separated with a space character. Buckets of the same row are separated with a tab character. Rows are separated with an end-line. The file path is given as the <u>first parameter</u> to your program.

In the input file, each row indicates the buckets that are accessed with the same hash key value (the first bucket in each row represents the prime area bucket and the others that follow the first represent overflow buckets).

Sample Linear Hashing Structure File

```
56 64
33 57 65   17

67 43 27   19
12
13
14
79 15 23
```

In the above file, the address of row one is 0 and it contains only one bucket that contains 56 and 64. The address of the second row is 1 (in decimal) and it contains two buckets: the first bucket contains 33, 57, and 65 and the second bucket, which represents an overflow bucket, contains only one item and it is 17. The address of the third row is then 2 and it contains an empty bucket that is no record has been inserted to that bucket yet.

- **Number of Records per Bucket:** It is given as the second parameter to your program.

  Sample Number of Records per Bucket: 3

- **New Load Factor:** The load factor value of the new linear hashing file structure. It is given as the third parameter to your program.

  Sample New Load Factor: 0.666

- **New Records File:** This file contains only one row, which contains some integers representing new records to be added to the new linear hashing file structure after reorganization. The file path is given as the forth parameter to your program.

  Sample New Records File:

  ```
  31
  ```

- **Linear Hashing File Structure Output File:** The output file where final linear hashing structure is printed. The file path is given as the fifth parameter to your program.

- **Initial Load Factor Output File:** The file where the load factor of the initial linear hashing structure is printed. The path is given as the sixth parameter to your program.

4. Output Specifications

- **Linear Hashing Structure Output:** Your program should print the final state of the linear hashing structure to the specified output file.

Expected Linear Hashing Structure Output

```
56 64
33 57 65   17

67 43 27   19
12
13
14
79 15 23   31
```

- **Initial Load Factor Output:** Your program should print the minimum possible load factor (**with three decimal places**) to create the initial linear hashing file structure to the specified output file.

- **Expected Initial Load Factor Output:** 0.334

5. Sample Run Command for Your Program

java your-java-class initial-linear-hashing-file-path 3 0.666 new-records-file-path linear-hashing-file-structure-output-file-path initial-load-factor-output-file-path

6. General Rules

- Your program must be named LinearHashing.java
- Your report must be named **StudentID.doc** (e.g. 20504152.doc)
- Note that any input file could be used for testing purposes. Also a supportive tool is used to detect plagiarism.

7. Tutorial: How to R/W File in JAVA

In JAVA, there are various ways to handle r/w a file; but you must use the following way. Other implementations will **not** be accepted.

You have to use BufferedReader and BufferedWriter classes to read and write a file respectively. Consider using the following methods of BufferedReader (You can also see JAVA API for all methods):

- *constructor*: **BufferedReader(Reader in):** Opens the file to read. The argument is an instance of Reader class. See JAVA API for more details.
- **readLine()**: Reads a line from the file.
- **close()**: Closes the file.

Also use the following methods of BufferedWriter:

- *constructor*: **BufferedWriter(Writer out):** Opens the file to write. The argument is an instance of Writer class. See JAVA API for more details.
- **write(String str):** Writes a string to the file.
- **close():** Closes the file.

8. Submission

I. Create a RAR archive file for your LinearHashing.java and 20504152.(doc | docx) files.

II. Name your RAR archive file as **StudentID_cs351p2.rar**(e.g. 20504152_cs351p2.rar). Files with other formats will be **ignored** automatically.

III. Use the submission page http://cs.bilkent.edu.tr/~ctoraman/cs351fall10/project2/submission/ to upload your RAR archive file. Read the upload rules in the submission page as well. Late submissions will **not** be accepted.

IV. You can resubmit your project until the due date.

V. For any question, contact with Emre VAROL(evarol@cs.bilkent.edu.tr).