Database System Implementation- CSE 507

Homework 1

Due date: January 27, 2016 9:00am

Instructions:

- All submissions must be made through usebackpack site for this course (https://www.usebackpack.com/iiitd/w2016/cse507)
- Only one submission per team would be considered and graded. It would be assumed that all members of the team have participated equally and same score would be given to all members of the team.
- Your submission should have names of all the members of your team.
- Any assumptions made while solving the problem should be clearly stated in the solution.
- Question 3 and Question 4 are for teams of size 3. These questions will not be graded for teams for size 2 or less.

Question 1 (Programming Question) (120 points)

This question would require you to implement and compare the performance of Extendible Hashing and Linear Hashing. You may use any high-level language to implement these structures. Java or C++ would be most preferable. Implementations using statistical packages like Matlab or R would not considered. Specialized libraries for managing hash tables must not be used. You may existing libraries for basic data structures like vectors and associative array (map in C++ STL libraries). You may also use math libraries as needed for tasks like, generating random numbers, converting to and from binary format, etc.

Details of implementation:

You may choose to implement the required structures in main memory, but secondary memory behavior must be simulated. Following tips would help you achieve this.

(a) The secondary memory can be simulated through a vector of fixed length arrays. Here, each of the arrays would be a bucket and the indices of the vector would be taken as the bucket address.
(b) Apart from records, each bucket should also have information on number of empty spaces and an index (in the vector) containing the next bucket of the file (or the overflow chain).
(c) The last bucket of the file (or the overflow chain) must have a special character denoting that it is end of the file (or the overflow chain).
(d) It is advisable to keep a separate area in the vector for storing the overflow buckets.
(e) Main memory is another vector of fixed size and is not dived into buckets. Note that you would need simulated main memory only in case of Extendible hash.

Information on File Records

(a) All records are of fixed size. For this homework, each record would be a single integer between 0 and 800000.
(b) The bucket capacity is fixed in terms of number of records it can contain, i.e., a bucket is basically a fixed number of integers.
(c) Expansion takes place as soon as a bucket overflows.

Extendible Hash

(a) The most significant bits are extracted to find the directory entry.
(b) The overflow bucket is split at most once. In other words, we would not try a second split even if the first split fails to release the overflow bucket.
(c) “Main memory” can hold upto 1024 directory entries. The rest resides in “Secondary memory.”

Linear Hash

(a) A simple division with modulo arithmetic is used to find the relevant bucket, i.e., it will not have a directory similar to Extendible hash.
(b) If needed you add or subtract a fixed number from the result of the modulo arithmetic to map it to appropriate index in the vector simulating the secondary memory.

Datasets to be created:

(a) Dataset-Uniform: Contains 100000 uniformly distributed random numbers between the range of 0 and 800000. Numbers may get repeated.
(b) **Dataset-HighBit:** Contains 60000 numbers uniformly generated between the range 700000 and 800000, and 40000 generated between the range 0 and 700000. Numbers may get repeated.

Experimental Analysis:

Goal: The performance of Extendible Hashing and Linear Hashing needs to be compared in terms of number of "disk accesses" for insert and search operations on these hashing techniques.

Parameters used:

(a) **N:** The number of records currently in the hash table.
(b) **B:** The number of buckets current in the hash table.
(c) **b:** Bucket capacity
(d) **bs**: The number of buckets accessed for a successful search (+1 if the directory is not in the main memory in case of extendible hash)
(e) **s:** Number of successful searches

Comparison Metrics

(a) Storage utilization: N/(B*b)
(b) Average successful search cost: bs/s
(c) Splitting cost:

Linear Hash: 1 access to read the bucket to be split + k accesses to read k overflow buckets + extra accesses to writ the overflow buckets attached to new and old buckets

Extendible Hash: 1 access to write the old bucket + 1 access to write the new bucket + extra accesses to write the overflow buckets attached to old and new buckets + accesses needed to update the directory pointers if the directory resides on "secondary memory."

Experiment to be conducted

(I) Keep inserting records from the dataset you created and continuously measure the values of the metrics (a) and (c). While metric (a) is perfectly continuous in nature, metric (b) will gather data-points only you see a split.
(II) After every 5000 records randomly generate 50 search queries and evaluate the metric (b)

Repeat these for both the datasets you created and for each dataset and plot the following three Plots:

<u>Plot 1:</u> Metric (a) against the number of records in the file for both Linear and Extendible hash for Bucket size 10 and 40.

<u>Plot 2:</u> Metric (b) against the number of records in the file for both Linear and Extendible hash for bucket size 10 and 40.

<u>Plot 3:</u> Metric (c) against the number of records in the file for both Linear and Extendible hash for bucket size 10 and 40.

Deliverables for Question 1

> **(A) Implementation code for Linear and Extendible Hash (70 points)**
> **(B) The required 9 plots (9 points)**
> **(C) A brief explanation of the trends (cross over points and general trends) observed (21 points)**
> **(D) Datasets used in the experiment.**

Question 2 (20points):- Suppose that a disk unit has the following parameters: seek time s=20 msec; rotational delay rd=10 msec; block transfer time btt=1 msec; block size B=2400 bytes; interblock gap size G=600 bytes. An EMPLOYEE file has the following fields: SSN, 9 bytes; LASTNAME, 20 bytes; FIRSTNAME, 20 bytes; MIDDLE INIT, 1 byte; BIRTHDATE, 12 bytes; ADDRESS, 30 bytes; PHONE, 10 bytes; SUPERVISORSSN, 9 bytes; DEPARTMENT, 4 bytes; JOBCODE, 4 bytes; deletion marker, 1 byte. The EMPLOYEE file has r=30000 STUDENT records, fixed-length format, and unspanned blocking. Write down appropriate formulas and calculate the following values for the above EMPLOYEE file:

(a) The record size R (including the deletion marker), the blocking factor bfr, and the number of disk blocks b.

(b) Calculate the wasted space in each disk block because of the unspanned organization.

(c) Calculate the transfer rate tr and the bulk transfer rate btr for this disk (see Appendix B of the textbook for definitions of tr and btr).

(d) Calculate the average number of block accesses needed to search for an arbitrary record in the file, using linear search.

(e) Calculate the average time needed in msec to search for an arbitrary record in the file, using linear search, if the file blocks are stored on consecutive disk blocks and double buffering is used.

(f) Calculate the average time needed in msec to search for an arbitrary record in the file, using linear search, if the file blocks are not stored on consecutive disk blocks.

(g) Assume that the records are ordered via some key field. Calculate the average number of block accesses and the average time needed to search for an arbitrary record in the file, using binary search.

Question 3 (Extra question which is compulsory for teams of size 3) (20 points) Consider a disk with block size B=512 bytes. A block pointer is P=6 bytes long, and a record pointer is P R =7 bytes long. A file has r=30,000 EMPLOYEE records of fixed-length. Each record has the following fields: NAME (30 bytes), SSN (12 bytes), DEPARTMENTCODE (9 bytes), ADDRESS (35 bytes), PHONE (12 bytes), BIRTHDATE (8 bytes), SEX (1 byte), JOBCODE (4 bytes), SALARY (4 bytes, real number). An additional byte is used as a deletion marker.

(a) Suppose the file is ordered by the key field SSN and we want to construct a primary index on SSN. Calculate (i) the index blocking factor bfr i (which is also the index fan-out fo); (ii) the number of first-level index entries and the number of first-level index blocks; (iii) the number of levels needed if we make it into a multi-level index; (iv) the total number of blocks required by the multi-level index; and (v) the number of block accesses needed to search for and retrieve a record from the file--given its SSN value--using the primary index.

(b) Suppose the file is not ordered by the key field SSN and we want to construct a secondary index on SSN. Repeat the previous exercise (part a) for the secondary index.

Question 4 (Extra question which is compulsory for teams of size 3) (15 points) A PARTS file with Part# as key field includes records with the following Part# values: 23, 65, 37, 60, 46, 92, 48, 71, 56, 59, 18, 21, and 10. Suppose the search field values are inserted in the order given above in a B + -tree of order p=4 and p leaf =3; show how the tree will expand and what the final tree looks like.