UG Software Lab - CS 409

Mini-project 1 Specification

Due date: 20-Feb-2021 11:55pm

Total Weightage 35%

Instructions:

- All submissions must be made through the Moodle site for this course
- Any assumptions made while solving the problem should be clearly stated in the solution.
- As always correctness of the algorithm must be ensured.
- TAs would be quizzing you on your code. You must understand each and every line of your submitted code. Also the implementation specifications mentioned in the questions need to be strictly followed. Failure to adhere to these requirements would result in substantial loss of points.
- Very Important: Your code should not have a directory structure. All files (code + dataset + written material for questions) should be present in just one folder. Note that this is absolutely crucial for grading this assignment.

In this project, you are required to implement Extendible Hashing. Specialized libraries for managing hash tables must not be used. You may use existing libraries for basic data structures like vectors and associative array (e.g, map in C++ STL libraries or equivalent in JAVA). You may also use math libraries as needed for tasks like, generating random numbers, converting to and from binary format, etc.

Dataset creation:

For this question, you would have to create a synthetic table (simulating sales records of department stores) containing 1 Lakh records. Each record in this file contains four fields: (1) Transaction ID (an integer), (2) Transaction sale amount (an integer), (3) Customer name (string) and, (4) category of item. Transaction ID is an integer to identify each transaction uniquely in the dataset. Transaction sale amount is a random integer between 1 and 500000. Customer name is a random 3 letter string. You can model this as a character array of length 3. Category of the item is a random integer between 1 --1500 (uniformly distributed across the table). Create 200 records in the mentioned format and store them in a file.

After creating the dataset, you are required to insert the records into an Extendible hash. Implementation specification of the extendible hash is given below:

Details of Extendible Hash:

- We would hashing on "TransactionID."
- Ideally, the buckets in the extendible hash should be stored in the secondary memory. However, for the purpose of this project, they would be stored in something called "Simulated Secondary Memory (detailed below)."
- The directory or bucket address table of the extendible hash would contain the hash prefix and pointer to the bucket sitting in "Simulated Secondary Memory (detailed below)."

Simulated Secondary Memory:

Following tips would help you achieve this.

(a) The secondary memory can be simulated through an array of the abstract data-type "bucket". You can fix a very large size for this array. **Simulated secondary memory must be an array only**. No arraylist, vector and dynamic arrays would be allowed.
(b) The bucket capacity is fixed in terms of number of records it can contain. Do not hard code this number as it would be varied in the experiments.
(c) Locations (indices) in this array form our "bucket address / hardware address."
(d) Here, the bucket abstract data-type would have the following information:
    a. Number of empty spaces
    b. An array of structures to store the records. Length of this array is fixed according to the parameter "bucket-size" specified.
    c. Link to the next bucket (valid only if this bucket is overflowing)
    d. All buckets in the overflow chain must be linked. The last bucket of the overflow chain must have a special character denoting that it is the end of the overflow chain.
(e) Note that in your entire code, there should be only one abstract data type for bucket for all the purposes, viz., records, overflow buckets for both records and directory entries.
(f) It is advisable to keep a separate area in the secondary memory for storing the overflow buckets.
(g) There should be only one instance of secondary memory running in your code. In other words, your normal data buckets, directory table buckets (detailed later) and the all-overflows must be in the same secondary memory.

Main Memory

(a) You can assume to have enough "main memory" for "bringing" in the buckets to be rehashed.
(b) In addition to item (a) "Main memory" can hold upto 1024 directory entries. The rest resides in your "Simulated Secondary memory."
(c) Directory entries overflowing into secondary memory would be using the same bucket abstract data type which was previously declared in item "Simulated Secondary Memory".

Other Details:

(a) The most significant bits are extracted to find the directory entry.
(b) Only one directory expansion is allowed per record insertion. Following the directory expansion, you may attempt to resolve the collision (if it still persists) by increasing the local depth (if local depth < global depth). In case the collision is still not resolved, just create an overflow bucket.
(c) "Main memory" can hold upto 1024 directory entries. The rest resides in "Simulated Secondary Memory."

Evaluation Details:

(1) TAs would use a small dataset to check the running of your code. For this, they may give you a file containing 15-20 records. Your code should read this file and then load the index records in the extendible hash.
(2) Extendible hash must have a separate insert function which would insert any given arbitrary "index record" into the extendible hash.
(3) Extendible hash must have a seperate intuitive visualize function for checking the correctness of the code.