Introduction to Database Systems

Individual Homework 2: Extendible Hashing

1. Introduction

In the lecture, you have learned what an index is in the database, the purpose of building an index, as well as a quick search in the database, which is hashing. In this homework, you are required to implement the extendible hashing and use it as a simple index for given data. Please continue reading for details.

2. Tasks

You need to write two files, "**hash.h**" and "**hash.cpp**", which are the header file and the cpp code of your hash-related classes. Please do not use the function like "map" or "unordered_map" to maintain the index without hash function:

The provided files are explained below:

- makefile
  - TA will use provided makefile to compile your code when grading
- main.cpp
  - contains the main function which will be used to test your homework
  - main function will read all files needed for you, as well as record time and memory pages used by each task
  - **DO NOT** modify this file
- utils.cpp
  - contains some utility functions for file reading, used time output
  - **DO NOT** modify this file
- utils.h
  - header file for utils.cpp
  - **DO NOT** modify this file
- data.txt

    - contains the data used in the homework
    - each line consists of a key and a value, both are integers and are separated with a “,”
    - the data is not sorted on key or value
    - will be loaded as two vectors of integer in the main function
    - if the key exist more than one, take the last recorded value as the value in hash table
    - note that the data TAs use to test your program will not be identical to the one provided
- key_query.txt
    - contains the queries to test your extendible hash
    - each line consists of a key, which is an integer
    - will be loaded as a vector of int in the main function
    - some of the keys may not exist in the data
    - note that the queries TAs use to test your program will not be identical to the one provided
- key_query_ans1.txt
    - answer to key_query.txt right after hash_table constructed
    - record the value and the local depth of the bucket where the key is, both of which are integers and are separated with a “,”
- key_query_ans2.txt
    - answer to key_query.txt after removing query
    - record the value and the local depth of the bucket where the key is, both of which are integers and are separated with a “,”
- remove_key_query.txt
    - contains the queries to be removed in extendible hash
    - each line consists of a keys, which is a integer

    - will be loaded as a vector of int in the main function
    - some of the key may not exist in the data
    - note that the queries TAs use to test your program will not be identical to the one provided

You need to write two files, the detail requirements are explained below:

- hash.cpp
    - the code for your index, this file must contains a class named “**hash_table**”
    - hash_table class is an implementation of extendible hash index as explained in the slide.
    - At least four function needed to be implemented (add more if you need)
        - constructor hash_table()
        - key_query
        - remove_query
        - clear
    - hash_table(table_size, bucket_size, num_rows, key, value) takes five inputs, which are an integer initializing the hash table size(fixed to 2), an integer constraint the size of the buckets(fixed to 2), an integer indicating the number of data rows, a vector of integer represent keys and a vector of integer represent values. You need to construct your hash index in this function by inserting the key, value pairs into the hash table **one by one**. While inserting, the global depth and the local depth should be maintained correctly.
    - key_query(query_keys, file_name) takes two inputs, which are a vector of integers indicating the key used for query and the name of the output file. In this function you need to **output a file named file_name**, each row consists of two integers. The former is the value corresponding to the keys in query_keys (output -1 if the key is not found), and the former is the local depth of the bucket with the hash index inferred from the keys(output the local depth the hash index need to be even if the key is not found).
    - remove_query takes one input, which is a vector of integers indicating the keys to be removed. In this function you need to remove the key, value pair in the

    specific bucket. While removing, the global depth and the local depth should be maintained correctly.
  - clear() takes no input, you need to free all the memory used by your extendible hash index in this function.
- hash.h
  - header file for hash.cpp

To summarize, you need to implement an extendible hash index class which is capable of inserting some key, value pairs into it, then processing a query with the index, a function to release used memory is also encouraged.

==

Briefly,

The aim of this homework is to develop students' understanding and implementation skills of data structures in database indexing with the extendible hash. The homework is split into three parts, requiring students to implement basic functions such as inserting data, querying, and deleting, and checking the correctness of the program through test data.

Tasks

1. The data.txt, which contains two fields, key and value, as the input file for the implementation of an extendible hash table in C++.
2. The key_query.txt, which contains a field key. Based on the key order in this file, generate the corresponding value and local depth and export it as key_query_ans1.txt. If the value corresponding to the key is not found in the hash table, display the value as -1. However, the local depth should still be displayed.
3. The remove_key_query.txt, which contains a field key, to delete the corresponding keys in the constructed extendible hash table. Next, create a new file key_query_ans2.txt by reusing key_query.txt as input to generate the value and local depth again. If the value corresponding to the key is not found in the hash table, display the value as -1. However, the local depth should still be displayed.

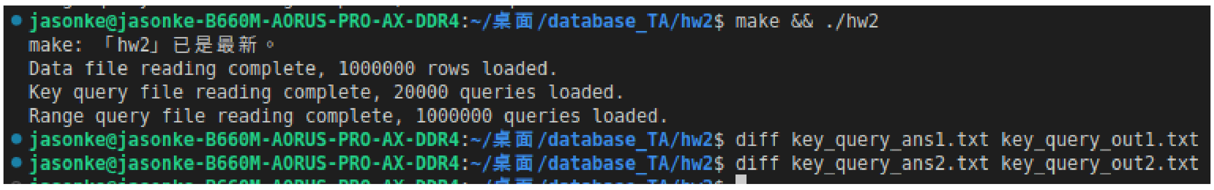
Requirements

1. Students must develop the extendible hash table in C++.
2. The extendible hash table should support fundamental operations like insertion, querying, and deletion while maintaining global depth and local depth.
3. Please verify the program's accuracy using test data.
4. Please submit the program source code and generate key_query_ans1.txt and key_query_ans2.txt files.

==

Please note that the data files TAs use to test your program will be different to the ones provided, even in the number of rows and queries. Remember to make your program work with these differences in data.

If you successfully run the code and do the tasks, your program will output 3 files, “key_query_out1.txt”, “key_query_out2.txt” (generated by functions you implemented) and “time_used.txt” (automatically generated, you do not need to implement this). Below figure is the example way to examine your correctness.



```
● jasonke@jasonke-B660M-AORUS-PRO-AX-DDR4:~/桌面/database_TA/hw2$ make && ./hw2
  make: 「hw2」已是最新。
  Data file reading complete, 1000000 rows loaded.
  Key query file reading complete, 20000 queries loaded.
  Range query file reading complete, 1000000 queries loaded.
● jasonke@jasonke-B660M-AORUS-PRO-AX-DDR4:~/桌面/database_TA/hw2$ diff key_query_ans1.txt key_query_out1.txt
● jasonke@jasonke-B660M-AORUS-PRO-AX-DDR4:~/桌面/database_TA/hw2$ diff key_query_ans2.txt key_query_out2.txt
```

3. Grading

In this homework, the correctness of your code only makes part of your score, to encourage you to optimize the index (e.g. concurrency, shrink), part of your score will be evaluated by the time used for the subtasks. The details are explained in the table below:

Description	Score(%)
Correctness of “key_query_out1.txt”(value + local depth)	10% + 20%
Correctness of “key_query_out2.txt”(value + local depth)	10% + 20%
Correctness of extendible hash implementation	20%
Time used to build index	10%
Time used to process remove query	10%

The key_query_out will be graded separately. Therefore, if you directly implement a big enough hash table with insert and remove but not extendible hash, you will still get 20% of the score. If you do not pass any "key_query_out1.txt" test data, you will not be able to get the score in "Time used to build index". In the same way, if you do not pass any "key_query_out2.txt" test data, you will not be able to get the score in "Time used to process remove query".

You will get at least 80 points if your output and implementation are correct, the other 40 points are based on performance of your code. For each subtask, we will rank all time used for the subtask of all students (wrong answer or no submission will not be included), then give score based on the PR(percentile rank) value:

- PR >= 90: you get 100% of the performance score
- 60 <= PR < 90: you get 75% of the performance score
- 20 <= PR < 60: you get 50% of the performance score
- PR <= 20: you get 25% of the performance score
- If your answer/implementation is wrong, you will not get any points in the performance score

With the exception of "correctness of extendible hash implementation", the homework will be revised automatically with a script. We will use your "hash.h" and "hash.cpp" with other provided codes to compile the program, we also provide the makefile we will use for your reference. The C++ version used will be C++17. Please make sure your code works with provided files and given settings.

4. Discussion

TAs had opened a channel **HW2 討論區** on Teams of the course, you can post questions about the homework on the forum. TAs will answer questions as soon as possible.

Discussion rules:

1. Do not ask for the answer to the homework.
2. Check if someone has asked the question you have before asking.
3. We encourage you to answer other students' questions, but again, do not give the answer to the homework. Reply the messages to answer questions.
4. Since we have this discussion forum, do not send email to ask questions about the homework unless the questions are personal and you do not want to ask publicly.
5. If you are not familiar with asking questions precisely, you can refer to **how to ask questions the smart way**.

5. Submission

1. The deadline of this homework is **4/28 (Fri.) 23:55:00**.

2. The submission only requires two files: hash.h and hash.cpp
3. You should put your “hash.h” and “hash.cpp” into one folder, the folder should be named as “**HW2_XXXXXXX**” where XXXXXXX is your student ID.
   ex: **HW2_123456**

   Then compress your folder into one `zip` file. Submit it to New E3 System with the format **HW2_XXXXXXX.zip** where XXXXXXX is your student ID.
   ex: **HW2_123456.zip**

   We **only accept one zip file**, each wrong format or naming format causes -10 points to your score (after considering late submission penalty).

4. Late submission lead to score of (original score)*0.85^{days}, for example, if you submit your homework right after the deadline, you get (original score)*0.85 points.
5. **0 points will be given to Plagiarism**. If the codes are similar to other people and you can’t explain your code properly, you will be identified as plagiarism. TAs will strictly examine your code. It is important that you must write your code by yourself.
6. If there is anything you are not sure about submission, ask in the discussion forum.