Introduction

- This is the second homework assignment for CSC3170, contributing 10% to the final course grade.
- The deadline for submission is **October 27th at 11:59 PM**.
- The assignment consists of two components: a writing part (40 points) and a coding part (60 points).
- Submitting AI-generated content or copying from other students will be considered as plagiarism and will be penalized accordingly.
- The teaching assistant responsible for this assignment is **Ruilizhen Hu**. For any questions or clarifications, please contact: ruilizhenhu@link.cuhk.edu.cn.

Writing Part

Problem 1 (10 points)**.** *Number of tuples in one page:*

1.1 (2 points)**.** *What is the smallest size, in bytes, of a record from the following schema in a slotted page? Assume that the record header is 4B, the boolean is 1B, and the date is 8B, disregard word-alignment and record directory metadata (e.g., record pointer and record length).*

```
name VARCHAR
student BOOLEAN
birthday DATE
state VARCHAR
```

1.2 (4 points)**.** *What is the maximum number of records that can be stored in a 2KB page, given the schema above? In addition to the record contents (including the 4B record header and the attribute values), each record requires a record pointer (8B) and a record length (8B) in the slot directory. At the page level, the slot count (8B) and free space pointer (8B) are stored once per page. Word-alignment applies to the record contents, where each word is 8B. Note that you are using the N-ary storage model (NSM) and need to use padding to conduct word-alignment where each word is 8B long.*

1.3 (4 points)**.** *If we are required to use reordering (and padding if needed) to conduct word-alignment, what is the maximum number of records that can be stored with the same setting in problem 1.2?*

Problem 2 (20 points). *Number of I/Os for query: Consider a database with a single table* `Class(id, name, instructor, size, credits)`*, where* `id` *is the primary key, and all attributes have the same fixed width. Suppose Class has 10000 tuples that fit into 500 pages, ignoring any additional storage overhead for the table (e.g., page headers, tuple headers). Additionally, you should make the following assumptions:*

- *The DBMS does not have any additional meta-data (e.g., sort order, zone maps).*
- `Class` *does not have any indexes (including for primary key* `id`*).*
- *None of* `Class`*'s pages are already in the buffer pool.*
- *Content-wise, the tuples of* `Class` *will make each query run the longest possible (this assumption is critical for solving problem 2.1).*
- *The tuples of* `Class` *can be in any order (this assumption is critical for solving problem 2.2 when you compute the minimum versus the maximum number of pages that the DBMS will potentially have to read)*

2.1 (10 points). *Consider the following query:*

```
SELECT MAX(credits) from Class
    WHERE size > 50
```

- *Suppose the DBMS uses the decomposition storage model (DSM) with implicit offsets. How many pages will the DBMS potentially have to read from disk to answer this query?*
- *Suppose the DBMS uses the N-ary storage model (NSM). How many pages will the DBMS potentially have to read from disk to answer this query?*

2.2 (10 points). *Consider another query:*

```
SELECT name, instructor, size from Class
    WHERE id = 114514 or id = 23333
```

- *Suppose the DBMS uses the decomposition storage model (DSM) with implicit offsets. What is the minimum and maximum number of pages that the DBMS will potentially have to read from disk to answer this query?*
- *Suppose the DBMS uses the N-ary storage model (NSM). What is the minimum and maximum number of pages that the DBMS will potentially have to read from disk to answer this query?*

Problem 3 (10 points). *Bloom Filter: A Bloom Filter is a space-efficient probabilistic data structure for set membership queries. It may return false positives (indicating that an element is in the set when it is not) but will never return a false negative. A Bloom Filter consists of:*

- *A bit array of length m (all initialized to 0).*
- *k independent hash functions.*

When inserting an element, we compute k positions and set them to 1. When querying, we check whether all k positions are set to 1.

3.1 (3 points)**.** *Suppose we have a Bloom Filter with a bit array of length 10 and two hash functions:*

- $h_1(x) = x \mod 10$
- $h_2(x) = (x/10) \mod 10$

Insert elements 7 and 27 in order. Draw the final bit array.

3.2 (3 points)**.** *Why can Bloom Filters produce false positives? Briefly explain.*

3.3 (4 points)**.** *Suppose we want to design a Bloom Filter for a set of $n = 1000$ elements with a false positive rate below 1%. (Note: We did not cover this in the lecture, and you need to do some research to give the answer.)*

- *Write down the approximate false positive probability formula.*
- *Briefly explain how to choose m (bit array size) and k (number of hash functions).*

Coding Part

Problem 4 (60 points)**.** *In this assignment, you are required to implement two different hash table data structures:* ***Cuckoo Hashing*** *and* ***Linear Probing Hashing****.*

Description of Cuckoo Hashing. *You will implement a hash table with two subtables of equal size m. Each key can be placed either at position $h_1(x)$ in Table 1 or at $h_2(x)$ in Table 2. The hash functions are defined as:*

$$h_1(x) = x \bmod m, \quad h_2(x) = \left\lfloor \frac{x}{m} \right\rfloor \bmod m$$

Insertion proceeds as follows: attempt to insert into $T_1[h_1(x)]$. If the slot is occupied, evict the existing element and move it to its alternative position in the other table. This process continues until:

- *an empty slot is found, or*
- *a cycle is detected (i.e., the displaced key returns to a previously visited position).*

If insertion fails due to a cycle, you must rehash by doubling the table size and reinserting all elements. Searching requires checking both positions $h_1(x)$ and $h_2(x)$. Deletion removes the key if present.

Description of Linear Probing Hashing. *You will implement a hash table with a single array of size m. The hash function is:*

$$h(x) = x \bmod m$$

On collision, the algorithm probes sequentially (linear probing) until it finds an empty slot. Deletion should be implemented with ***lazy deletion****, i.e., mark the slot as deleted but do not clear it. The* ***load factor*** *is defined as*

$$\alpha = \frac{\textit{number of occupied slots}}{m},$$

where "occupied slots" count only entries that currently store a key (deleted slots are not counted). If the load factor exceeds 0.7*, you must rehash by doubling the table size and reinserting all elements.*

Provided Skeleton Code. *We provide header files with class definitions for both hash tables. You must complete the implementation in the corresponding* `.cc` *files. The hash functions are stored as function pointers and are defined in the implementation files.*

Grading. *We will test your code using both public and* ***hidden*** *unit tests. Each hash table implementation will be evaluated by 6 tests (5 points each), totaling 60 points.*

- *Cuckoo Hashing: 6 tests, 5 points each (30 points).*
- *Linear Probing: 6 tests, 5 points each (30 points).*

Passing all tests earns full credit. Partial credit will be given if only basic functionality is correct.

Submission. *Submit only two implementation files and commit with the original file names (i.e.* `cuckoo_hash.cc` *and* `linear_probing.cc`*)*