Lab 3

Hash Table, Sparse Table

In this lab session, we will focus on implementing data structures that can query data very quickly: Hash table and Sparse table.

1 Hash Table

Students are required to implement a hash table with 4 different collision handling: Linear Probing, Quadratic Probing, Chaining using Linked List, and Double Hashing.

1.1 Linear Probing

Implement a `struct` to represent hash table using **linear probing** with the following variables:

- `vector<hashNode*> table`: This is an array containing the data for the hash table, where each item is a `hashNode` storing a key-value pair. For example:

```cpp
struct hashNode
{
    string key;
    int value;
}
```

- `int capacity`: size of the hash table.

and functions:

- `void init(unsigned int hashSize)`: initialize an empty hash table with the given size.
- `void release()`: free all dynamically allocated memory in the hash table.
- `unsigned int hashFunctions(...)`: hash functions to compute the index for a given key.
- `void add(string key, int value)`: add element. If the key existed, update the old value.
- `int* searchValue(string key)`: search element in the table. If not existed, return `NULL`.
- `void removeKey(string key)`: remove an element from the hash table.

And other variables, functions if necessary.

Here is example code:

```
struct hashTable
{
    // Variables (Attributes)
    struct hashNode
    {
        string key;
        int value;
    };

    int capacity;
    vector<hashNode*> table;

    // Functions (Methods)
    void init(unsigned int hashSize) {...}
    void release() {...}
    unsigned int hashFunction(string key) {...}
    void add(string key, int value) {...}
    int* searchValue(string key) {...}
    void removeKey(string key) {...}
}
```

To hash a string in `hashFunction(string key)`, you can use polynomial rolling hash function. The formula is:

$$hash(s) = \left(\sum_{i=0}^{n-1} (s[i] \times p^i) \right) \mod m$$

which:

- s: The key as a string of length n.
- $s[i]$: ASCII code of the character at position i from s.
- $p = 31$.
- $m = 10^9 + 9$.

In `main.cpp`, try using hash table. You can initialize the hash table with a small size, then perform add, search, and remove operations on it before releasing the memory and ending the program. **No menu is required.**

1.2 Quadratic Probing

Implement a hash table using **quadratic probing** with the same variables and functions as in Linear Probing.

1.3 Chaining using Linked List

Implement a hash table using **chaining** with the same variables and functions as in Linear Probing. However, the `hashNode` will need to be modified a bit to implement a **linked list**, as shown below:

```
struct hashNode
{
    string key;
    int value;
    hashNode* next; // Add this line
};
```

Also, you will need to implement some additional functions to work with linked lists.

1.4 Double Hashing

Implement a hash table using **double hashing** with the same variables as in Linear Probing, and the same functions but adding a second hash function.

2 Sparse Table

Given an integer array, find the **maximum**, **minimum**, and **greatest common divisor** (GCD) of the elements within a specified range using **Sparse Table**.

Input:

- An integer `n` representing the number of elements in the array.
- An array of `n` integers.
- Two integers `L` and `R` denoting the range $[L, R]$ (index starts from 0).

Output:

- Three integers representing the maximum, minimum, and GCD of the elements in the range $[L, R]$ on the same line.

Submission

Your source code must be contributed in the form of a compressed file and named your submission according to the format `StudentID.zip`. Here is a detail of the directory organization:

```
StudentID
├── Exercise_1_1.cpp
├── Exercise_1_2.cpp
├── Exercise_1_3.cpp
├── Exercise_1_4.cpp
└── Exercise_2.cpp
```

The end.