Vietnam National University,
Ho Chi Minh City

University of Science

Faculty of Information Technology

Implementing Hash Table from scratch

CSC10004 – Data Structures and Algorithms

Ngo Nguyen The Khoa – 23127065 – 23CLC09

August 6, 2024

Contents

1 How I implemented the requirements

I implemented the requirements as follows:

- The second hash function I have used for *'Double Hashing'* is $h_2(k) = 1 + (k \mod (m-1))$, where m is the size of the hash table.
- With *'Double Hashing'* and *'Quadratic Probing'* methods, I have maintained a hash table which its size is a prime number larger than the real size to avoid collisions as much as possible.
- I didn't implement rehashing for *'Double Hashing'* and *'Quadratic Probing'* because the size of the hash table is fixed, so it could cause 'not-found' situation in some cases. (You can find those in the *'Double Hashing'* and *'Quadratic Probing'* experiment sections in this report.)
- Although there was no rehashing implementation in my source, I tested using rehashing for *'Double Hashing'* and *'Quadratic Probing'* methods to solve the collisions.
  - **Rehashing using *'double the size'*:** I have to rehash 8 times, which means the size is a prime number larger than $3e5 \cdot 256$ to avoid missing some records.
  - **Rehashing using *'prime number sizing'*:** I have to resize the hash table to $58500011 \approx 3e5 \cdot 195$ to avoid missing some records.

2 Result Screenshots

2.1 Linear Probing Operations

```cpp
int main() {
    HashTable<int, string> table;
    table.init(8);

    cout << "1. add operations\n";
    table.add(2, "two");
    table.add(4, "four");
    table.add(6, "six");
    table.add(10, "ten");
    table.add(3, "three");
    table.add(5, "five");

    for (auto *node : table.table) {…

    cout << ">>> Update key 2\n";
    table.add(2, "TWO");
    printSearchValue(table, 2);

    cout << "\n2. search operations\n";
    printSearchValue(table, 1);
    printSearchValue(table, 6);
    printSearchValue(table, 10);

    cout << "\n3. remove operations\n";
    table.removeKey(4);
    printSearchValue(table, 4);

    table.release();
```

```
Linear Probing
)· cpp main
1. add operations
null
null
2 two
10 ten
4 four
3 three
6 six
5 five
>>> Update key 2
Key 2 has value "TWO"

2. search operations
Not found value of 1
Key 6 has value "six"
Key 10 has value "ten"

3. remove operations
Not found value of 4
Linear Probing
)·
```

Figure 1: Screenshot of Linear Probing operations.

2.2 Quadratic Probing Operations

```cpp
int main() {
    HashTable<int, string> table;
    table.init(8);

    cout << "1. add operations\n";
    table.add(2, "two");
    table.add(4, "four");
    table.add(6, "six");
    table.add(10, "ten");
    table.add(3, "three");
    table.add(5, "five");

    for (auto *node : table.table) {…

    cout << ">>> Update key 2\n";
    table.add(2, "TWO");
    printSearchValue(table, 2);

    cout << "\n2. search operations\n";
    printSearchValue(table, 1);
    printSearchValue(table, 6);
    printSearchValue(table, 10);

    cout << "\n3. remove operations\n";
    table.removeKey(4);
    printSearchValue(table, 4);

    table.release();
```

```
Quadratic Probing
❯ cpp main
1. add operations
null
null
2 two
3 three
4 four
5 five
6 six
null
null
null
10 ten
>>> Update key 2
Key 2 has value "TWO"

2. search operations
Not found value of 1
Key 6 has value "six"
Key 10 has value "ten"

3. remove operations
Not found value of 4
Quadratic Probing
❯
```

Figure 2: Screenshot of Quadratic Probing operations.

2.3 Chaining AVL Operations

```cpp
int main() {
    HashTable<int, string> table;
    table.init(8);

    cout << "1. add operations\n";
    table.add(2, "two");
    table.add(4, "four");
    table.add(6, "six");
    table.add(10, "ten");
    table.add(3, "three");
    table.add(5, "five");

    for (auto *node : table.table) {…

    cout << ">>> Update key 2\n";
    table.add(2, "TWO");
    printSearchValue(table, 2);

    cout << "\n2. search operations\n";
    printSearchValue(table, 1);
    printSearchValue(table, 6);
    printSearchValue(table, 10);

    cout << "\n3. remove operations\n";
    table.removeKey(4);
    printSearchValue(table, 4);

    table.release();
```

```
Chaining AVL
) cpp main
1. add operations
null
null
2 two, 10 ten,
3 three,
4 four,
5 five,
6 six,
null
>>> Update key 2
Key 2 has value "TWO"

2. search operations
Not found value of 1
Key 6 has value "six"
Key 10 has value "ten"

3. remove operations
Not found value of 4
Chaining AVL
)
```

Figure 3: Screenshot of Chaining AVL operations.

2.4 Chaining Linked List Operations

```cpp
int main() {
    HashTable<int, string> table;
    table.init(8);

    cout << "1. add operations\n";
    table.add(2, "two");
    table.add(4, "four");
    table.add(6, "six");
    table.add(10, "ten");
    table.add(3, "three");
    table.add(5, "five");

    for (auto *node : table.table) {…

    cout << ">>> Update key 2\n";
    table.add(2, "TWO");
    printSearchValue(table, 2);

    cout << "\n2. search operations\n";
    printSearchValue(table, 1);
    printSearchValue(table, 6);
    printSearchValue(table, 10);

    cout << "\n3. remove operations\n";
    table.removeKey(4);
    printSearchValue(table, 4);

    table.release();
```

```
Chaining Linked List
❯ cpp main
1. add operations
null
null
2 two, 10 ten,
3 three,
4 four,
5 five,
6 six,
null
>>> Update key 2
Key 2 has value "TWO"

2. search operations
Not found value of 1
Key 6 has value "six"
Key 10 has value "ten"

3. remove operations
Not found value of 4
Chaining Linked List
❯
```

Figure 4: Screenshot of Chaining Linked List operations.

2.5 Double Hashing Operations

```cpp
int main() {
    HashTable<int, string> table;
    table.init(8);

    cout << "1. add operations\n";
    table.add(2, "two");
    table.add(4, "four");
    table.add(6, "six");
    table.add(10, "ten");
    table.add(3, "three");
    table.add(5, "five");

    for (auto *node : table.table) {…

    cout << ">>> Update key 2\n";
    table.add(2, "TWO");
    printSearchValue(table, 2);

    cout << "\n2. search operations\n";
    printSearchValue(table, 1);
    printSearchValue(table, 6);
    printSearchValue(table, 10);

    cout << "\n3. remove operations\n";
    table.removeKey(4);
    printSearchValue(table, 4);

    table.release();
```

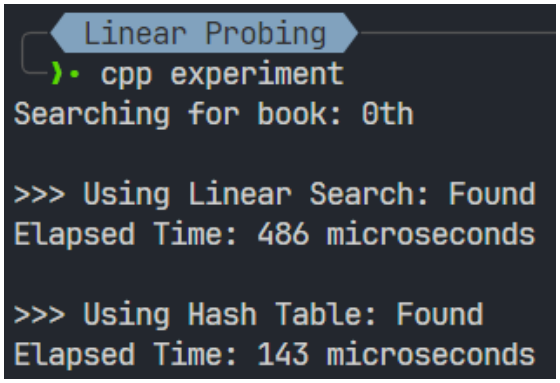
```
Double Hashing
❯ cpp main
1. add operations
null
null
2 two
3 three
4 four
5 five
6 six
null
null
null
10 ten
>>> Update key 2
Key 2 has value "TWO"

2. search operations
Not found value of 1
Key 6 has value "six"
Key 10 has value "ten"

3. remove operations
Not found value of 4
Double Hashing
❯
```

Figure 5: Screenshot of Double Hashing operations.

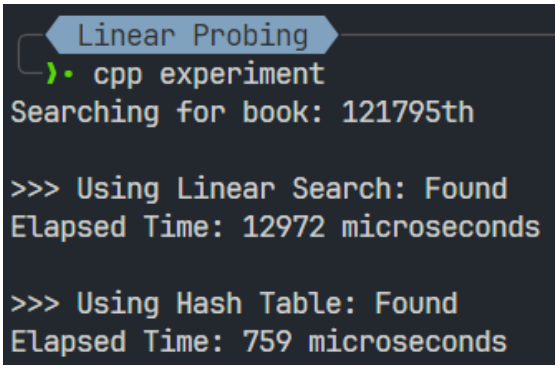
3 Experiments

3.1 Linear Probing and Linear Searching Algorithm

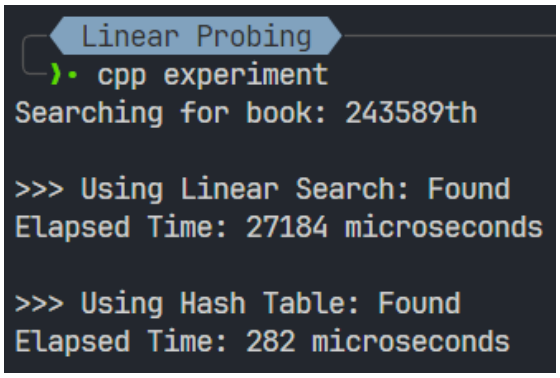
- **Theoretical Time Complexity:**
  - The time complexity of searching using Linear Probing is $O(n)$.
  - The time complexity of searching using Linear Searching Algorithm is $O(n)$.
- **Actual execution time:**
  - Searching using Linear Probing is faster than using Linear Searching Algorithm in most cases (except there are too many collisions).
- **Case Screenshots:**



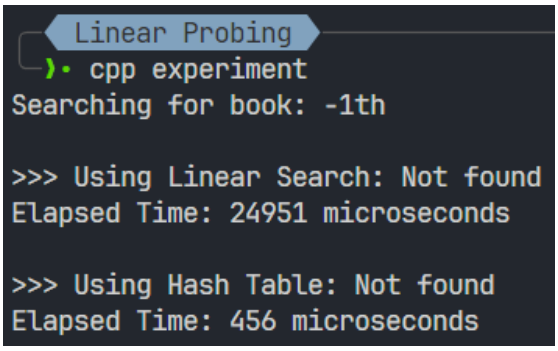
(a) When the key is at the begining of Hash Table



(b) When the key is at the middle of Hash Table



(c) When the key is at the end of Hash Table

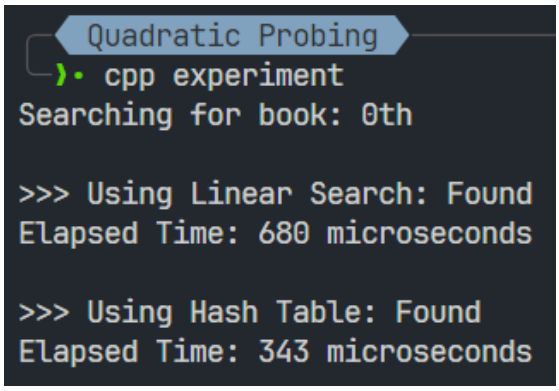


(d) When the key is 'not' in Hash Table

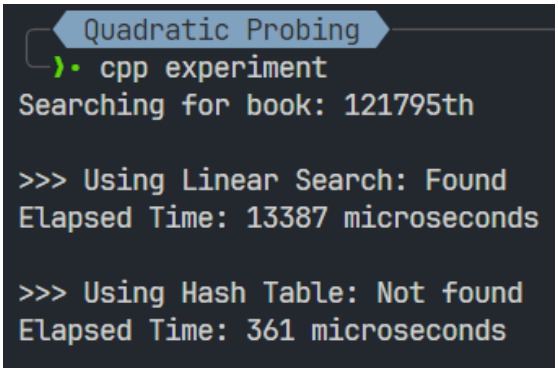
Figure 6: Screenshots of Searching Case with Linear Probing.

3.2 Quadratic Probing and Linear Searching Algorithm

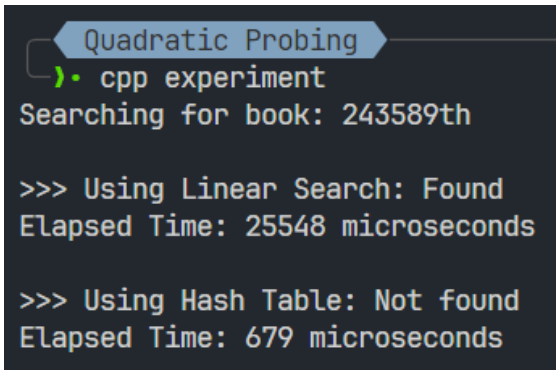
- **Theoretical Time Complexity:**
  - The time complexity of searching using Quadratic Probing is $O(n)$.
  - The time complexity of searching using Linear Searching Algorithm is $O(n)$.
- **Actual execution time:**
  - Searching using Quadratic Probing is faster than using Linear Searching Algorithm in most cases (except there are too many collisions).
- **Case Screenshots:**



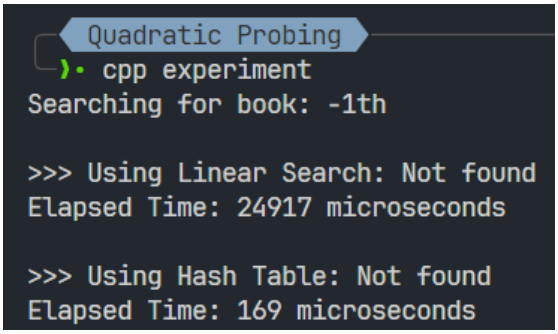
(a) When the key is at the begining of Hash Table



(b) When the key is at the middle of Hash Table



(c) When the key is at the end of Hash Table

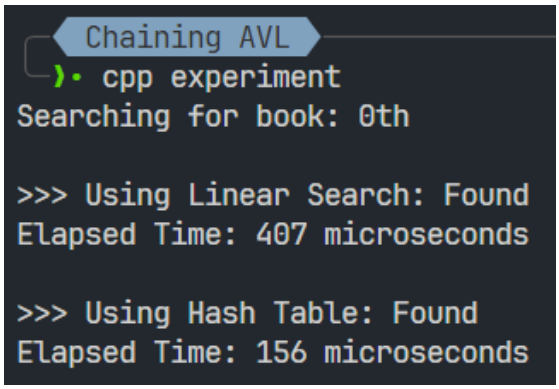


(d) When the key is 'not' in Hash Table

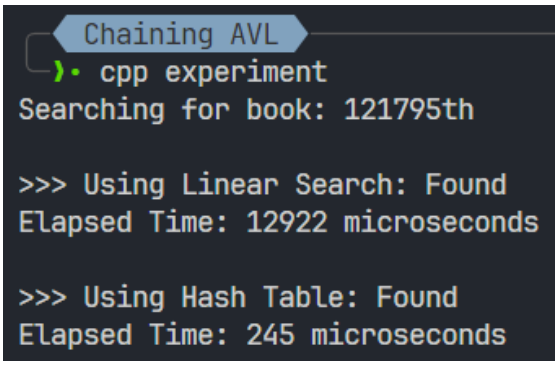
Figure 7: Screenshots of Searching Case with Quadratic Probing.

3.3 Chaining AVL and Linear Searching Algorithm

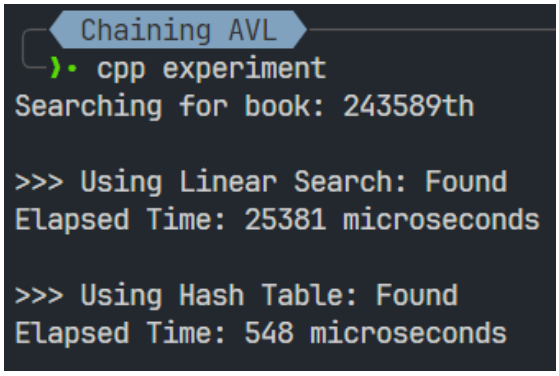
- **Theoretical Time Complexity:**
  - The time complexity of searching using Chaining AVL is $O(\log(n))$.
  - The time complexity of searching using Linear Searching Algorithm is $O(n)$.
- **Actual execution time:**
  - Searching using Chaining AVL is almost faster than using Linear Searching Algorithm.
- **Case Screenshots:**



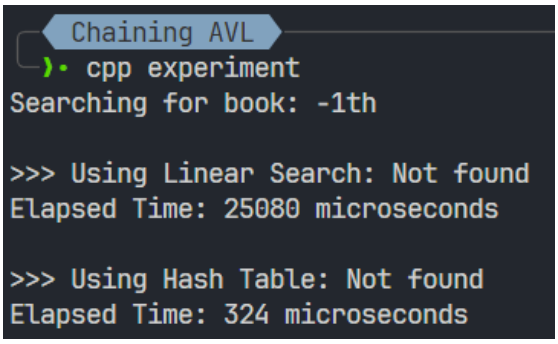
(a) When the key is at the begining of Hash Table



(b) When the key is at the middle of Hash Table



(c) When the key is at the end of Hash Table

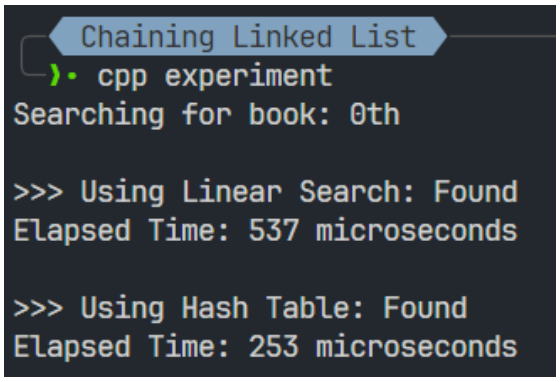


(d) When the key is 'not' in Hash Table

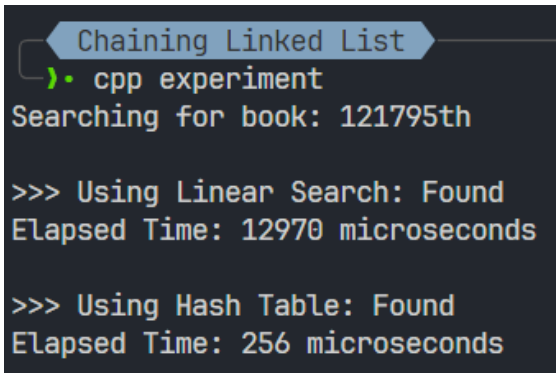
Figure 8: Screenshots of Searching Case with Chaining AVL.

3.4 Chaining Linked List and Linear Searching Algorithm

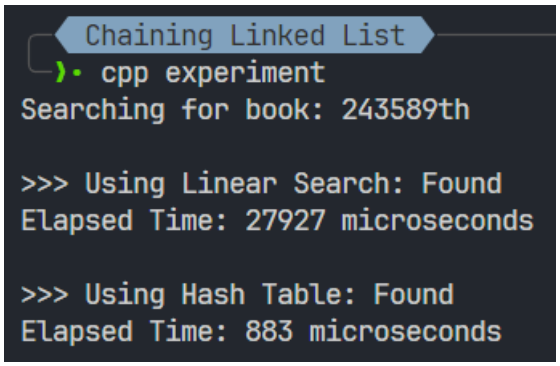
- **Theoretical Time Complexity:**
  - The time complexity of searching using Chaining Linked List is $O(n)$.
  - The time complexity of searching using Linear Searching Algorithm is $O(n)$.
- **Actual execution time:**
  - Searching using Chaining Linked List is faster than using Linear Searching Algorithm in most cases (except there are too many collisions).
- **Case Screenshots:**



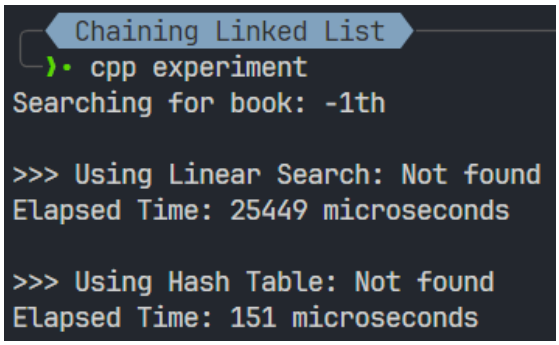
(a) When the key is at the begining of Hash Table



(b) When the key is at the middle of Hash Table



(c) When the key is at the end of Hash Table

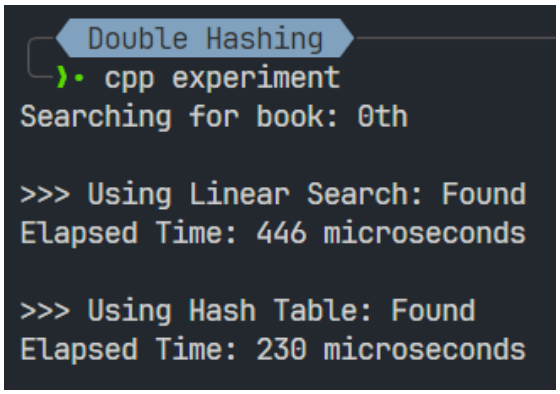


(d) When the key is 'not' in Hash Table

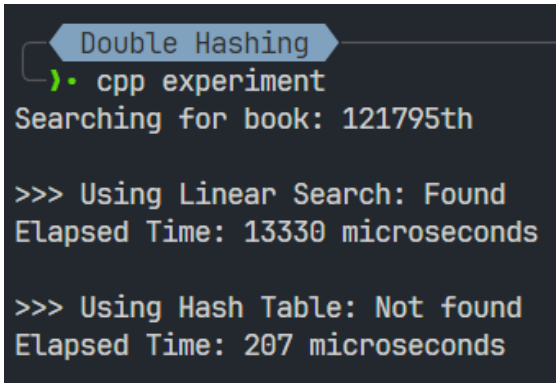
Figure 9: Screenshots of Searching Case with Chaining Linked List.

3.5 Double Hashing and Linear Searching Algorithm

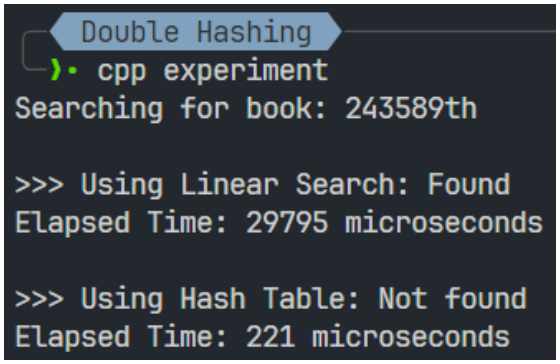
- **Theoretical Time Complexity:**
  - The time complexity of searching using Double Hashing is $O(n)$.
  - The time complexity of searching using Linear Searching Algorithm is $O(n)$.
- **Actual execution time:**
  - Searching using Double Hashing is faster than using Linear Searching Algorithm in most cases (except there are too many collisions).
- **Case Screenshots:**



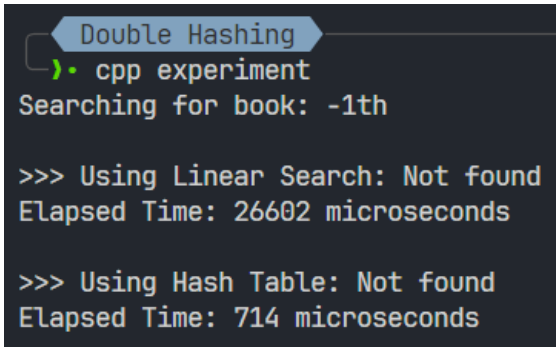
(a) When the key is at the begining of Hash Table



(b) When the key is at the middle of Hash Table



(c) When the key is at the end of Hash Table



(d) When the key is 'not' in Hash Table

Figure 10: Screenshots of Searching Case with Double Hashing.

4 Self-evaluation

No.	Details	Score
1	Linear Probing	100%
2	Quadratic Probing	100%
3	Chaining using Linked List	100%
4	Chaining using AVL Tree	100%
5	Double Hashing	100%
6	Experiments	100%
7	Report	100%

5 Exercise Feedback

5.1 What I have learned from this Exercise

Because almost the things could be done easily, I have learned a few things new from this exercise.

- I have learned how to implement the hash table using different methods of collision handling.
- I have learned how to use rehashing to solve the collisions.
- I also know how to test the hash table using different test cases.
- Have a strong understanding of the hash table and its methods is the big thing I got after finishing this Exercise.

5.2 What I found challenging

This exercise is quite simple and easy, so I have no difficulties to finish this task.

5.3 What I have used in this exercise

- I used C++ to implement the data structures and the test cases.
- I also used LaTeX to write this report.