VIETNAM NATIONAL UNIVERSITY OF
HO CHI MINH CITY

UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

Report

Exercise 2: Implementing Hash Table from scratch

Course name: Data Structures and Algorithms

CSC10004_23CLC09

Students:
Nguyen Le Ho Anh Khoa - 23127211

Teacher:
Bui Duy Dang
Truong Tan Khoa
Nguyen Thanh Tinh

August 13, 2024

Contents

1 Student Information **2**

2 How I implemented the requirements **2**

3 Detailed Experiments **3**
3.1 Linear Probing . . . 3
3.1.1 Add key: . . . 3
3.1.2 Search key (Use the Hash Table added as above 3.1.1): . . . 4
3.1.3 Remove key (Use the Hash Table added as above 3.1.1): . . . 4
3.1.4 Experiments Linear Probing and Linear Search Algorithms . . . 5
3.2 Quadratic Probing . . . 6
3.2.1 Add key: . . . 6
3.2.2 Search key (Use the Hash Table added as above 3.2.1): . . . 7
3.2.3 Remove key (Use the Hash Table added as above 3.2.1): . . . 7
3.2.4 Experiments Quadratic Probing and Linear Search Algorithms . . . 8
3.3 Chaining using Linked List . . . 9
3.3.1 Add key: . . . 9
3.3.2 Search key (Use the Hash Table added as above 3.3.1): . . . 10
3.3.3 Remove key (Use the Hash Table added as above 3.3.1): . . . 10
3.3.4 Experiments Chaining Using Linked List and Linear Search Algorithms . . . 11
3.4 Chaining using AVL Tree . . . 12
3.4.1 Add key: . . . 12
3.4.2 Search key (Use the Hash Table added as above 3.4.1): . . . 13
3.4.3 Remove key (Use the Hash Table added as above 3.4.1): . . . 13
3.4.4 Experiments Chaining using AVL Tree and Linear Search Algorithms . . . 14
3.5 Double Hashing . . . 15
3.5.1 Add key: . . . 15
3.5.2 Search key (Use the Hash Table added as above 3.5.1): . . . 16
3.5.3 Remove key (Use the Hash Table added as above 3.5.1): . . . 16
3.5.4 Experiments Double Hashing and Linear Search Algorithms . . . 17

4 Self - Evaluation **18**

5 Exercise Feedback **18**
5.1 What have I learned . . . 18
5.2 What was my difficult . . . 19

1 Student Information

Class: 23CLC09
Student ID: 23127211
Full name: Nguyen Le Ho Anh Khoa

2 How I implemented the requirements

To hash a string, you used polynomial rolling hash function as the hints in requirements. The formula is:

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

where:

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

To hash the keys, I used the following hash functions:

- Linear Probing: $h(k, i) = (h'(k) + i) \mod \textit{capacity}$. $h'(k)$ is the hash value of key k.
- Double Hashing: $h(k, i) = (h_1(k) + i \times h_2(k)) \mod \textit{capacity}$. $h_1(k)$ is the first hash value of key k, and $h_2(k)$ is the second hash value of key k. $h_2(k) = 1 + h(k) \mod (\textit{capacity} - 1)$.
- Quadratic Probing: $h(k, i) = (h'(k) + i^2) \mod \textit{capacity}$. $h'(k)$ is the hash value of key k.
- Chaining using Linked List: $h(k, i) = (h'(k) + i) \mod \textit{capacity}$. $h'(k)$ is the hash value of key k. Each cell of the hash table is a linked list. When a collision occurs, the new element is inserted at the end of the linked list.
- Chaining using AVL Tree: $h(k, i) = (h'(k) + i) \mod \textit{capacity}$. $h'(k)$ is the hash value of key k. Each cell of the hash table is an AVL tree. When a collision occurs, the new element is inserted into the AVL tree.

Quadratic probing and **Double hashing** frequently encounter collisions when the hash table size is small or when keys hash to the same index. This can quickly lead to revisiting the same indices, especially if the capacity is a prime number.

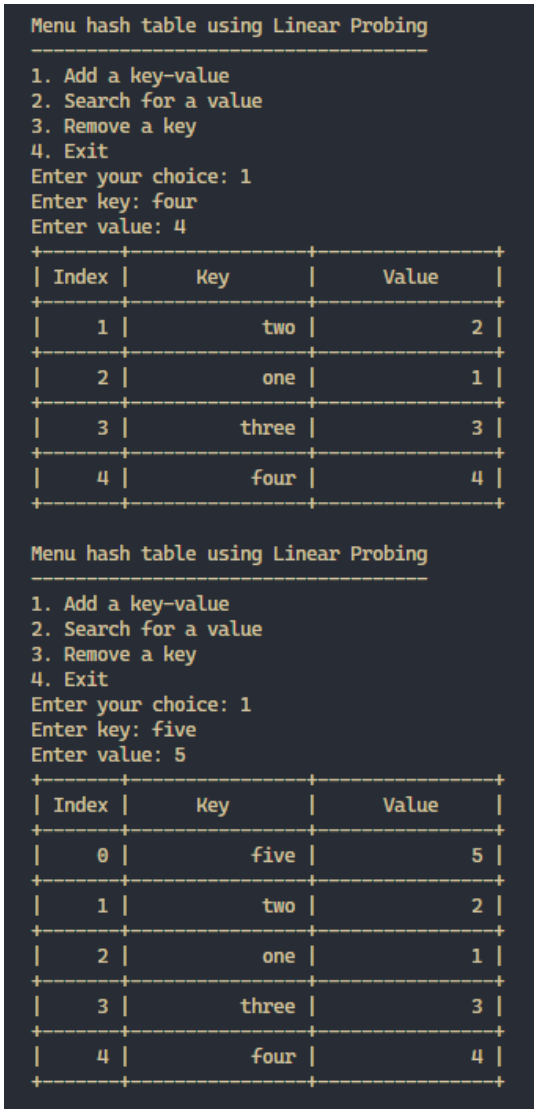
To address this issue, a rehash function needs to be implemented to increase the size of the hash table and redistribute the elements. The rehash function creates a new hash table with double size and transfers all elements from the old hash table to the new one. This helps to minimize collisions and improve the performance of the hash table.

3 Detailed Experiments

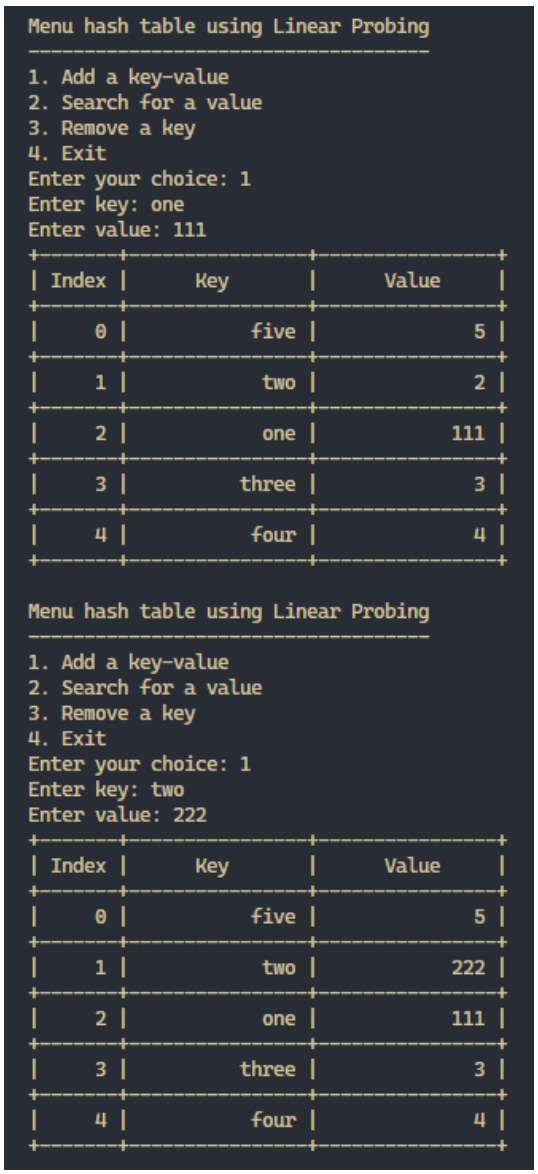
3.1 Linear Probing

3.1.1 Add key:

The order insert key to the Hash Table by Linear Probing is as follows: [one; 1], [two; 2], [three; 3], [four; 4], [five; 5], [one; 111], [two; 222]. The size of the Hash Table is 5. The Hash Table is shown in the figure below:



(a) Add new



(b) Update

Figure 1: Add new by key by Linear Probing

3.1.2 Search key (Use the Hash Table added as above 3.1.1):

```
Menu hash table using Linear Probing
-------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 2
Enter key: one
Value: 111
+-------+---------------+----------------+
| Index |      Key      |      Value     |
+-------+---------------+----------------+
|     0 |          five |              5 |
+-------+---------------+----------------+
|     1 |           two |            222 |
+-------+---------------+----------------+
|     2 |           one |            111 |
+-------+---------------+----------------+
|     3 |         three |              3 |
+-------+---------------+----------------+
|     4 |          four |              4 |
+-------+---------------+----------------+
```

(a) Found

```
Menu hash table using Linear Probing
-------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 2
Enter key: six
Key not found
+-------+---------------+----------------+
| Index |      Key      |      Value     |
+-------+---------------+----------------+
|     0 |          five |              5 |
+-------+---------------+----------------+
|     1 |           two |            222 |
+-------+---------------+----------------+
|     2 |           one |            111 |
+-------+---------------+----------------+
|     3 |         three |              3 |
+-------+---------------+----------------+
|     4 |          four |              4 |
+-------+---------------+----------------+
```

(b) Not Found

Figure 2: Search value by key by Linear Probing (size=5)

3.1.3 Remove key (Use the Hash Table added as above 3.1.1):

```
Menu hash table using Linear Probing
-------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 3
Enter key: one
+-------+---------------+----------------+
| Index |      Key      |      Value     |
+-------+---------------+----------------+
|     0 |          five |              5 |
+-------+---------------+----------------+
|     1 |           two |            222 |
+-------+---------------+----------------+
|     3 |         three |              3 |
+-------+---------------+----------------+
|     4 |          four |              4 |
+-------+---------------+----------------+
```

(a) Removed

```
Menu hash table using Linear Probing
-------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 3
Enter key: six
Key six not found
+-------+---------------+----------------+
| Index |      Key      |      Value     |
+-------+---------------+----------------+
|     0 |          five |              5 |
+-------+---------------+----------------+
|     1 |           two |            222 |
+-------+---------------+----------------+
|     3 |         three |              3 |
+-------+---------------+----------------+
|     4 |          four |              4 |
+-------+---------------+----------------+
```

(b) Not Found

Figure 3: Remove value by key by Linear Probing

3.1.4 Experiments Linear Probing and Linear Search Algorithms

```
$ g++ experiment.cpp && ./a
Data loaded successfully

--------Linear Probing--------
Searching for the first title:
Searching in vector:            Mark P. O. Morford
Time taken:                     0 microseconds
Searching in hash table:        Mark P. O. Morford
Time taken:                     0 microseconds

Searching for the middle title:
Searching in vector:            BEVERLY CLEARY
Time taken:                     2963 microseconds
Searching in hash table:        BEVERLY CLEARY
Time taken:                     1001 microseconds

Searching for the last title:
Searching in vector:            Christopher Biffle
Time taken:                     4998 microseconds
Searching in hash table:        Christopher Biffle
Time taken:                     1001 microseconds

Searching title not exist:
Searching in vector:            Not Found
Time taken:                     6001 microseconds
Searching in hash table:        Not Found
Time taken:                     1000 microseconds
```

Figure 4: Compare time Linear Probing Search and Linear Search Algorithms

- The time complexity of Linear Probing Search is O(n)
- The time complexity of Linear Search Algorithms is O(n).

But in the most cases, using Linear Probing Search in Hash Table is faster than using Linear Search Algorithms in normal vector.

3.2 Quadratic Probing

3.2.1 Add key:

The order insert key to the Hash Table by Quadratic Probing is as follows: [one; 1], [two; 2], [three; 3], [four; 4], [five; 5], [one; 111], [two; 222]. The size of the Hash Table is 5. But after added the key [three; 3], the size of the Hash Table is 10 by rehashing. However, this condition can be met even if there are still empty slots available, due to the nature of quadratic probing. The Hash Table is shown in the figure below:

```
Menu hash table using Quadratic Probing
---------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 1
Enter key: four
Enter value: 4
+-------+---------------+---------------+
| Index |      Key      |     Value     |
+-------+---------------+---------------+
|     0 |         three |             3 |
+-------+---------------+---------------+
|     1 |           two |             2 |
+-------+---------------+---------------+
|     2 |           one |             1 |
+-------+---------------+---------------+
|     4 |          four |             4 |
+-------+---------------+---------------+

Menu hash table using Quadratic Probing
---------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 1
Enter key: five
Enter value: 5
Table is full. Rehashing...
+-------+---------------+---------------+
| Index |      Key      |     Value     |
+-------+---------------+---------------+
|     0 |          five |             5 |
+-------+---------------+---------------+
|     2 |           one |             1 |
+-------+---------------+---------------+
|     4 |          four |             4 |
+-------+---------------+---------------+
|     6 |         three |             3 |
+-------+---------------+---------------+
|     7 |           two |             2 |
+-------+---------------+---------------+
```

(a) Add new

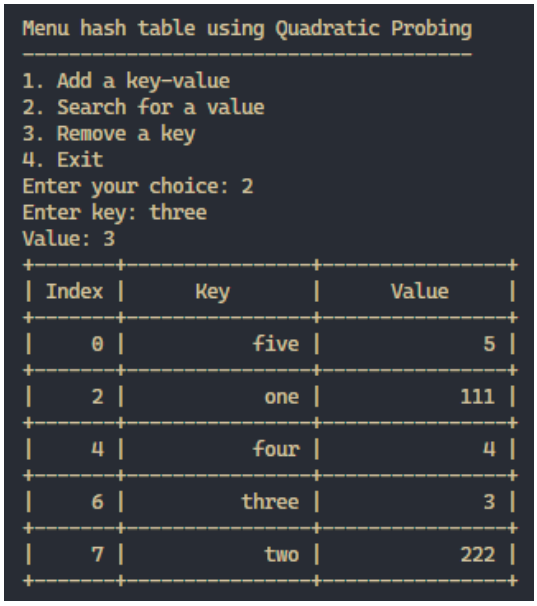
```
Menu hash table using Quadratic Probing
---------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 1
Enter key: one
Enter value: 111
+-------+---------------+---------------+
| Index |      Key      |     Value     |
+-------+---------------+---------------+
|     0 |          five |             5 |
+-------+---------------+---------------+
|     2 |           one |           111 |
+-------+---------------+---------------+
|     4 |          four |             4 |
+-------+---------------+---------------+
|     6 |         three |             3 |
+-------+---------------+---------------+
|     7 |           two |             2 |
+-------+---------------+---------------+

Menu hash table using Quadratic Probing
---------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 1
Enter key: two
Enter value: 222
+-------+---------------+---------------+
| Index |      Key      |     Value     |
+-------+---------------+---------------+
|     0 |          five |             5 |
+-------+---------------+---------------+
|     2 |           one |           111 |
+-------+---------------+---------------+
|     4 |          four |             4 |
+-------+---------------+---------------+
|     6 |         three |             3 |
+-------+---------------+---------------+
|     7 |           two |           222 |
+-------+---------------+---------------+
```

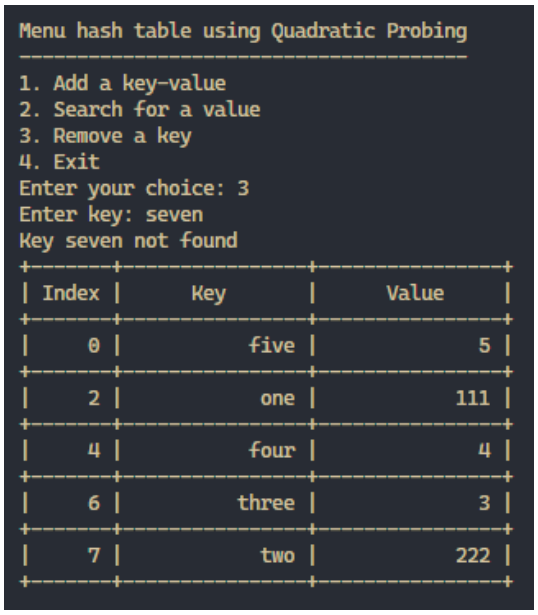
(b) Update

Figure 5: Add new key by Quadratic Probing (size=5)

3.2.2 Search key (Use the Hash Table added as above 3.2.1):



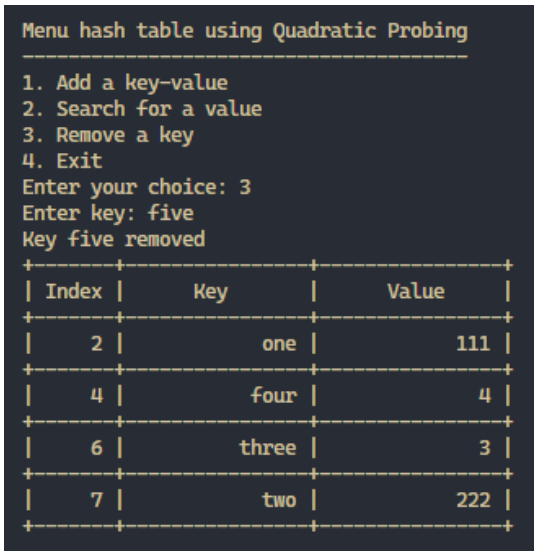
(a) Found



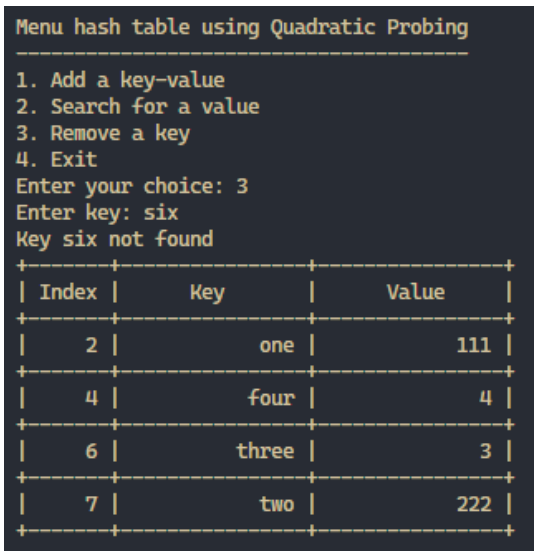
(b) Not Found

Figure 6: Search value by key by Quadratic Probing (size=5)

3.2.3 Remove key (Use the Hash Table added as above 3.2.1):



(a) Removed



(b) Not Found

Figure 7: Remove value by key by Quadratic Probing

3.2.4 Experiments Quadratic Probing and Linear Search Algorithms

```
$ g++ experiment.cpp && ./a
Data loaded successfully

--------Quadratic Probing--------
Searching for the first title:
Searching in vector:            Mark P. O. Morford
Time taken:                     1005 microseconds
Searching in hash table:        Mark P. O. Morford
Time taken:                     990 microseconds

Searching for the middle title:
Searching in vector:            BEVERLY CLEARY
Time taken:                     5484 microseconds
Searching in hash table:        BEVERLY CLEARY
Time taken:                     1000 microseconds

Searching for the last title:
Searching in vector:            Christopher Biffle
Time taken:                     5583 microseconds
Searching in hash table:        Christopher Biffle
Time taken:                     996 microseconds

Searching title not exist:
Searching in vector:            Not Found
Time taken:                     3080 microseconds
Searching in hash table:        Not Found
Time taken:                     998 microseconds
```

Figure 8: Compare time Quadratic Probing and Linear Search Algorithms

- The time complexity of Quadratic Probing Search is O(n)
- The time complexity of Linear Search Algorithms is O(n).

But in the most cases, using Quadratic Probing Search in Hash Table is faster than using Linear Search Algorithms in normal vector.

3.3 Chaining using Linked List

3.3.1 Add key:

The order insert key to the Hash Table by Chaining using Linked List is as follows: [one; 1], [two; 2], [three; 3], [four; 4], [five; 5], [one; 111], [two; 222]. The size of the Hash Table is 5. The Hash Table is shown in the figure below:

```
Menu hash table using Chaining by Linked List
-----------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 1
Enter key: four
Enter value: 4
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [two,2] -> [three,3]
2         [one,1]
3
4         [four,4]

Menu hash table using Chaining by Linked List
-----------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 1
Enter key: five
Enter value: 5
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [two,2] -> [three,3] -> [five,5]
2         [one,1]
3
4         [four,4]
```

(a) Add new

```
Menu hash table using Chaining by Linked List
-----------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 1
Enter key: one
Enter value: 111
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [two,2] -> [three,3] -> [five,5]
2         [one,111]
3
4         [four,4]

Menu hash table using Chaining by Linked List
-----------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 1
Enter key: two
Enter value: 222
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [two,222] -> [three,3] -> [five,5]
2         [one,111]
3
4         [four,4]
```

(b) Update

Figure 9: Add new by key by Chaining using Linked List

3.3.2 Search key (Use the Hash Table added as above 3.3.1):

```
Menu hash table using Chaining by Linked List
-----------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 2
Enter key: one
Value: 111
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [two,222] -> [three,3] -> [five,5]
2         [one,111]
3
4         [four,4]
```

(a) Found

```
Menu hash table using Chaining by Linked List
-----------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 2
Enter key: six
Key not found
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [two,222] -> [three,3] -> [five,5]
2         [one,111]
3
4         [four,4]
```

(b) Not Found

Figure 10: Search value by key by Chaining using Linked List

3.3.3 Remove key (Use the Hash Table added as above 3.3.1):

```
Menu hash table using Chaining by Linked List
-----------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 2
Enter key: one
Value: 111
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [two,222] -> [three,3] -> [five,5]
2         [one,111]
3
4         [four,4]
```

(a) Removed

```
Menu hash table using Chaining by Linked List
-----------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 3
Enter key: seven
Key not found
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [two,222] -> [five,5]
2         [one,111]
3
4         [four,4]
```

(b) Not Found

Figure 11: Remove value by key by Chaining Using Linked List

3.3.4 Experiments Chaining Using Linked List and Linear Search Algorithms

```
$ g++ experiment.cpp && ./a
Data loaded successfully

--------Chaining by Linked List--------
Searching for the first title:
Searching in vector:            Mark P. O. Morford
Time taken:                     1002 microseconds
Searching in hash table:        Mark P. O. Morford
Time taken:                     997 microseconds

Searching for the middle title:
Searching in vector:            BEVERLY CLEARY
Time taken:                     2003 microseconds
Searching in hash table:        BEVERLY CLEARY
Time taken:                     1001 microseconds

Searching for the last title:
Searching in vector:            Christopher Biffle
Time taken:                     6125 microseconds
Searching in hash table:        Christopher Biffle
Time taken:                     1001 microseconds

Searching title not exist:
Searching in vector:            Not Found
Time taken:                     5998 microseconds
Searching in hash table:        Not Found
Time taken:                     1001 microseconds
```

Figure 12: Compare time Chaining using Linked List Search and Linear Search Algorithms

- The time complexity of Chaining using Linked List Search is O(n)
- The time complexity of Linear Search Algorithms is O(n).

But in the most cases, using Chaining using Linked List Search in Hash Table is faster than using Linear Search Algorithms in normal vector.

3.4 Chaining using AVL Tree

3.4.1 Add key:

The order insert key to the Hash Table by Chaining using AVL Tree is as follows: [one; 1], [two; 2], [three; 3], [four; 4], [five; 5], [one; 111], [two; 222]. The size of the Hash Table is 5. The Hash Table is shown in the figure below:

```
Menu hash table using Chaining by AVL
-------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 1
Enter key: four
Enter value: 4
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [three,3] [two,2]
2         [one,1]
3
4         [four,4]

Menu hash table using Chaining by AVL
-------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 1
Enter key: five
Enter value: 5
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [five,5] [three,3] [two,2]
2         [one,1]
3
4         [four,4]
```

(a) Add new

```
Menu hash table using Chaining by AVL
-------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 1
Enter key: one
Enter value: 111
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [five,5] [three,3] [two,2]
2         [one,111]
3
4         [four,4]

Menu hash table using Chaining by AVL
-------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 1
Enter key: two
Enter value: 222
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [five,5] [three,3] [two,222]
2         [one,111]
3
4         [four,4]
```

(b) Update

Figure 13: Add new by key by Chaining using AVL Tree

3.4.2 Search key (Use the Hash Table added as above 3.4.1):

```
Menu hash table using Chaining by AVL
-------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 2
Enter key: two
Value: 222
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [five,5] [three,3] [two,222]
2         [one,111]
3
4         [four,4]
```

(a) Found

```
Menu hash table using Chaining by AVL
-------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 2
Enter key: nine
Key not found
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [five,5] [three,3] [two,222]
2         [one,111]
3
4         [four,4]
```

(b) Not Found

Figure 14: Search value by key by CChaining Using AVL Tree

3.4.3 Remove key (Use the Hash Table added as above 3.4.1):

```
Menu hash table using Chaining by AVL
-------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 3
Enter key: five
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [three,3] [two,222]
2         [one,111]
3
4         [four,4]
```

(a) Removed

```
Menu hash table using Chaining by AVL
-------------------------------------
1. Add a key-value
2. Search for a value
3. Remove a key
4. Exit
Enter your choice: 3
Enter key: nine
Key not found
Hash table:
Bucket    [Key,Value]
---------------------------------------
0
1         [three,3] [two,222]
2         [one,111]
3
4         [four,4]
```

(b) Not Found

Figure 15: Remove value by key by Chaining Using AVL Tree

3.4.4 Experiments Chaining using AVL Tree and Linear Search Algorithms

```
--------Chaining by AVL--------
Searching for the first title:
Searching in vector:            Mark P. O. Morford
Time taken:                     1002 microseconds
Searching in hash table:        Mark P. O. Morford
Time taken:                     997 microseconds

Searching for the middle title:
Searching in vector:            BEVERLY CLEARY
Time taken:                     3297 microseconds
Searching in hash table:        BEVERLY CLEARY
Time taken:                     1000 microseconds

Searching for the last title:
Searching in vector:            Christopher Biffle
Time taken:                     4999 microseconds
Searching in hash table:        Christopher Biffle
Time taken:                     998 microseconds

Searching title not exist:
Searching in vector:            Not Found
Time taken:                     5999 microseconds
Searching in hash table:        Not Found
Time taken:                     1001 microseconds
```

Figure 16: Compare time Chaining using AVL Tree Search and Linear Search Algorithms

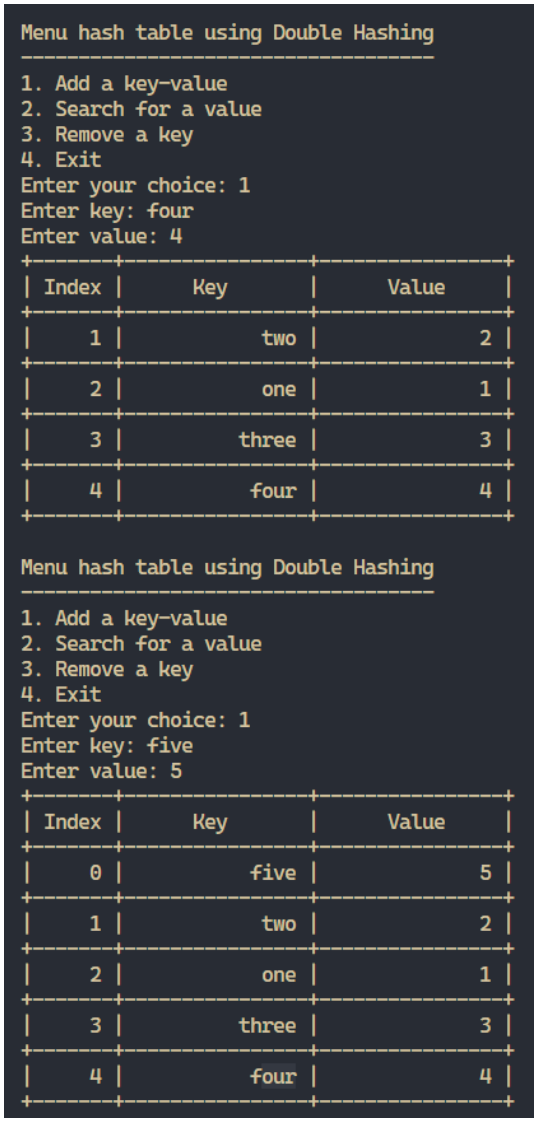
- The time complexity of Chaining using AVL Tree Search is O(logn)
- The time complexity of Linear Search Algorithms is O(n).

So in the most cases, using Chaining using AVL Tree Search in Hash Table is faster than using Linear Search Algorithms in normal vector.

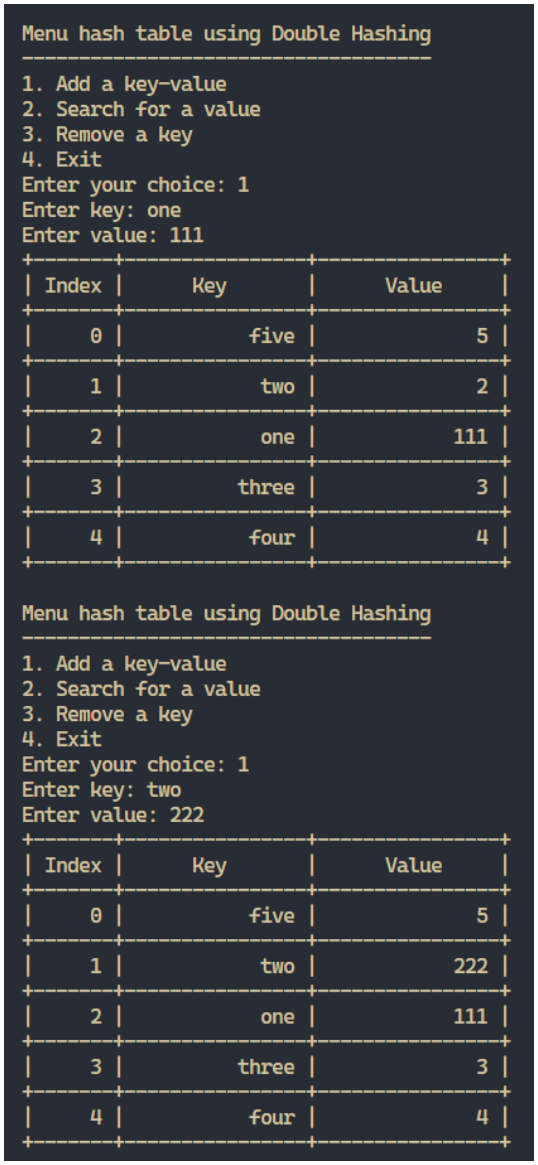
3.5 Double Hashing

3.5.1 Add key:

The order insert key to the Hash Table size = 5 by Double Hashing is as follows: [one; 1], [two; 2], [three; 3], [four; 4], [five; 5], [one; 111], [two; 222].The Hash Table is shown in the figure below:



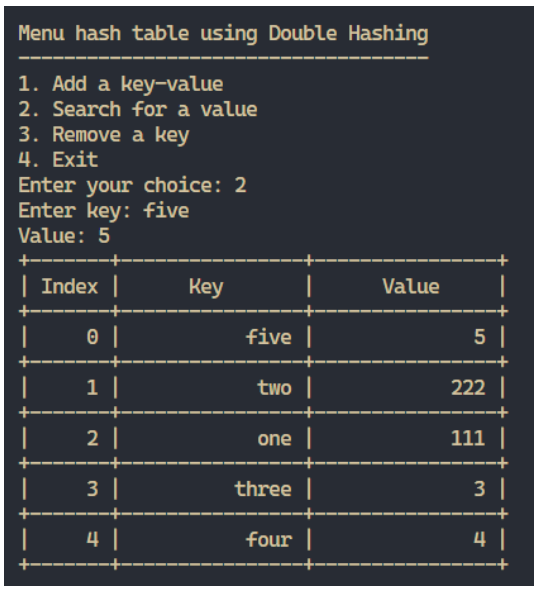
(a) Add new



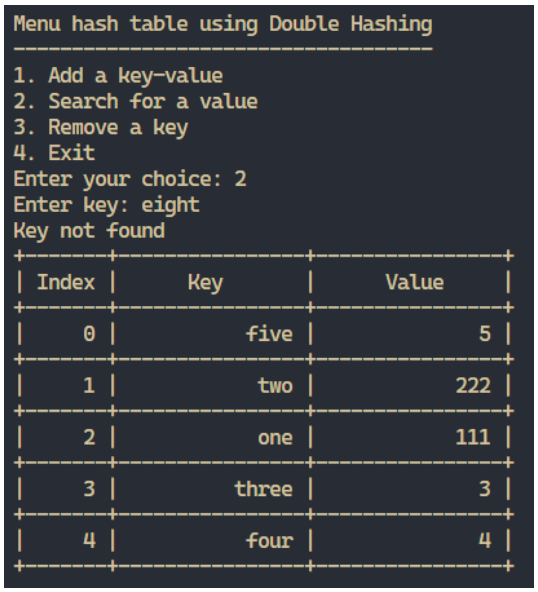
(b) Update

Figure 17: Add new key by Quadratic Probing (size=5)

3.5.2 Search key (Use the Hash Table added as above 3.5.1):



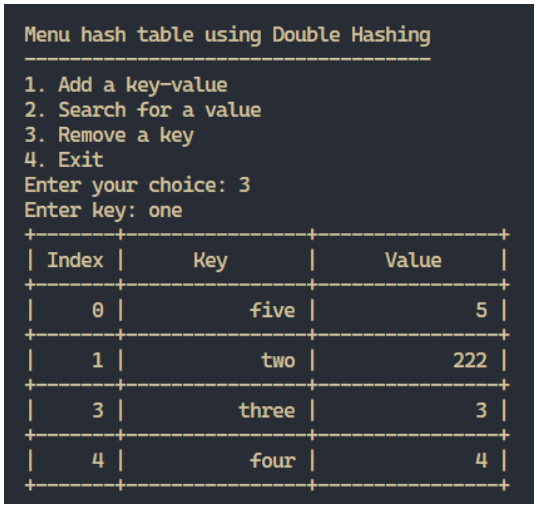
(a) Found



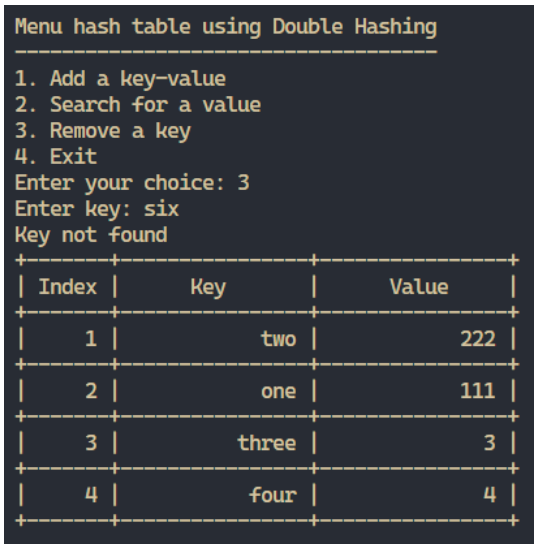
(b) Not Found

Figure 18: Search value by key by Quadratic Probing (size=5)

3.5.3 Remove key (Use the Hash Table added as above 3.5.1):



(a) Removed



(b) Not Found

Figure 19: Remove value by key by Double Hashing

3.5.4 Experiments Double Hashing and Linear Search Algorithms

```
$ g++ experiment.cpp &&./a
Table is full, rehashing...
Data loaded successfully

--------Double Hashing--------
Searching for the first title:
Searching in vector:            Mark P. O. Morford
Time taken:                     1003 microseconds
Searching in hash table:        Mark P. O. Morford
Time taken:                     991 microseconds

Searching for the middle title:
Searching in vector:            BEVERLY CLEARY
Time taken:                     1995 microseconds
Searching in hash table:        BEVERLY CLEARY
Time taken:                     995 microseconds

Searching for the last title:
Searching in vector:            Christopher Biffle
Time taken:                     5001 microseconds
Searching in hash table:        Christopher Biffle
Time taken:                     1000 microseconds

Searching title not exist:
Searching in vector:            Not Found
Time taken:                     5000 microseconds
Searching in hash table:        Not Found
Time taken:                     998 microseconds
```

Figure 20: Compare time Double Hashing Search and Linear Search Algorithms

- The time complexity of Double Hashing Search is O(n)
- The time complexity of Linear Search Algorithms is O(n).

But in the most cases, using Double Hashing Search in Hash Table is faster than using Linear Search Algorithms in normal vector.

When run with large dataset, Hash Table by Double Hashing need to rehash because the condition **if (i == capacity)** is used to determine if the table is full. However, this condition can be met even if there are still empty slots available, due to the nature of quadratic probing.

4 Self - Evaluation

In this exercise, I meticulously implemented Hash Table from scratch in C++. My evaluation encompassed several critical aspects.

- I verified the correctness of both implementations, ensuring that they adhered to the fundamental properties of Hash Table.
- I optimized the code for performance, using efficient algorithms and data structures. I also minimized the number of comparisons and avoided redundant operations.
- I paid close attention to memory usage, avoiding unnecessary wastage.
- I prioritized code readability and maintainability, using descriptive names and adding explanatory comments. Lastly, I thoroughly tested edge cases and implemented robust error handling. The code is modular, well-commented, and meets the specified criteria.

Table 1: Self - Evaluation about my Exercise

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%
	Total	**100%**

5 Exercise Feedback

5.1 What have I learned

- In the past, my approach was sequential programming, but this task has broadened my knowledge to include the fundamentals of object-oriented programming.
- I've acquired skills in creating reports with LaTeX.
- I've employed Github as a vault for my source code and reports, all of which are securely stored in my personal repository.

5.2 What was my difficult

- At the outset, my journey with programming was fraught with difficulties due to my unfamiliarity with object-oriented programming. However, my comprehension has been greatly enhanced after delving into a plethora of resources available on the internet. [3]
- I encountered some difficulties when writing function about AVL tree. However, after a period of debugging and contemplation, I successfully resolved the issues. [2]
- I had a hard time understanding the concept of hash tables, but after a period of research and experimentation, I was able to grasp the fundamentals.
- Measuring the time complexity of the algorithms was a challenging task, but I was able to overcome it by using the Chrono library in C++. [1]
- Lastly, my English proficiency isn't quite up to par yet, so there's a possibility that this report contains some grammatical errors.

References

[1] GeeksforGeeks. Measure execution time of a function in C++. 2023.

[2] GeeksforGeeks. AVL Tree — Set 1 (Insertion). 2024.

[3] GeeksforGeeks. Object Oriented Programming in C++. 2024.