



ABBOTTABAD UNIVERSITY OF SCIENCE AND TECHNOLOGY

Department of Computer Science



NAME: MUHAMMAD HASEEB

CLASS: BSCS

ROLL NO: 14861

Sir: JAMAL ABDUAL AHUD

SUBJECT: DSA

DATE: 01-02-2025

PROJECT PROPOSAL

DSA

HASH TABLE IMPLEMENTATION

GitHub link

1. Introduction **Error! Bookmark not defined.**
 - 1.1 Purpose **Error! Bookmark not defined.** 1.2
 - Document Conventions **Error! Bookmark not defined.**
 - 1.3 Project Scope **Error! Bookmark not defined.**
 - 1.1.1 Scope Definition **Error! Bookmark not defined.** 1.3.2
 - Core Features **Error! Bookmark not defined.** 1.3.3
 - Subsequent Releases **Error! Bookmark not defined.**
 - 1.3.4 Alignment with User and Business Goals **Error! Bookmark not defined.**
 - 1.4 References **Error! Bookmark not defined.**
2. over all Description **Error! Bookmark not defined.**
 - 2.1 Product Perspective..... **Error! Bookmark not defined.**
 - 2.1.1 Product Context **Error! Bookmark not defined.** 2.1.2
 - Product Origin **Error! Bookmark not defined.**
 - 2.1.3 Product Relationship to Existing Systems **Error! Bookmark not defined.**
 - 2.1.4 Product Ecosystem **Error! Bookmark not defined.**
 - 2.2 User Classes and Characteristics **Error! Bookmark not defined.**

2.2.1 Tech Enthusiasts	Error! Bookmark not defined.	2.2.2
Casual Shoppers	Error! Bookmark not defined.	2.2.3.
Favored User Class	Error! Bookmark not defined.	
2.2.4. Alignment with User Needs	Error! Bookmark not defined.	
2.3 Operating Environment	Error! Bookmark not defined.	
2.3.1 Hardware Platform	Error! Bookmark not defined.	
2.3.2 Operating Systems and Versions	Error! Bookmark not defined.	
2.4 Design and Implementation Constraints.....	Error! Bookmark not defined.	
2.4.2 Database Technology	Error! Bookmark not defined.	2.4.4
Third-Party Integrations	Error! Bookmark not defined.	
2.4.5 User Interface Design.....	Error! Bookmark not defined.	
2.5 Assumptions and Dependencies	Error! Bookmark not defined.	
2.5.1 Assumptions	Error! Bookmark not defined.	
2.5.2 Dependencies	Error! Bookmark not defined.	
3. System Features	Error! Bookmark not defined.	
5. External interface requirements	Error! Bookmark not defined.	
5.1 User Interfaces	Error! Bookmark not defined.	
5.1.1 Design Standards and Guidelines	Error! Bookmark not defined.	5.1.2
Screen Layout and Resolution	Error! Bookmark not defined.	
5.1.3 Standard Interface Elements.....	Error! Bookmark not defined.	
5.2 Software Interfaces	Error! Bookmark not defined.	
5.2.4 Non-Functional Requirements	Error! Bookmark not defined.	
5.3 Hardware Interfaces	Error! Bookmark not defined.	
5.3.1 Supported Device Types	Error! Bookmark not defined.	

1. Introduction

1.1 Purpose

The purpose of this document is to define the requirements for the implementation of a **Hash Table** using a **Linked List (Chaining Method) in Python**. This implementation provides efficient insertion, deletion, and lookup operations for managing key-value pairs.

1.2 Document Conventions

- Code is written in **Python 3.x**.
- Comments are used to explain functionalities.
- CamelCase is used for class names, and snake case is used for function and variable names.

1.3 Project Scope

1.3.1 Scope Definition

This project aims to implement a **Hash Table** data structure with the following features:

- **Chaining Method:** Handles collisions using linked lists.
- **Dynamic Resizing:** Automatically expands when the load factor exceeds **0.7**.
- **CRUD Operations:** Supports **Insertion, Search, and Deletion** of key-value pairs.

1.3.2 Core Features

- **Custom Hash Function** using modulo operation.
- **Load Factor Monitoring** to determine when to resize the table.
- **Linked List-based Chaining** to handle collisions.

1.3.3 Future Enhancements

- Implement **Open Addressing** as an alternative to chaining.
- Add **Graphical User Interface (GUI)** for visualization.

1.4 References

- **Data Structures and Algorithms in Python** – Michael T. Goodrich
- Python Documentation: <https://docs.python.org/3/>

2. Overall Description

2.1 Product Perspective

The Hash Table implementation is an **in-memory** data structure used for quick lookup operations. It serves as a foundational component for databases, caches, and other applications requiring fast data retrieval.

2.1.1 Product Context

This implementation is a standalone module that can be integrated into larger projects requiring **efficient key-value storage**.

2.2 User Classes and Characteristics

- **Developers:** Use it as a module in applications.
- **Students:** Learn hash tables and chaining in Python.

2.3 Operating Environment

- **Programming Language:** Python 3.x
- **Hardware:** Works on standard machines (1GB RAM, 1GHz CPU)
- **Operating System:** Windows/Linux/Mac

2.4 Design and Implementation Constraints

- Uses **linked lists** for collision handling.
- Load factor threshold: **0.7** (resizes when exceeded).

3. System Features

3.1 Key Functionalities

- **Insert (key, value):** Adds key-value pairs.
- **Search(key):** Retrieves value for a given key.
- **Delete(key):** Removes a key-value pair.
- **Display ():** Prints the entire hash table.

4. External Interface Requirements

4.1 User Interface

- **Command-Line Interface (CLI)** for input/output.

4.2 Software Interfaces

- Python 3 Standard Library

5. Quality Attributes

5.1 Performance

- **Average Time Complexity:** $O(1)$ for search, insert, and delete.
- **Worst Case Complexity:** $O(n)$ (when all keys hash to the same bucket).

5.2 Reliability

- Ensures accurate **insertion, retrieval, and deletion**.

5.3 Usability

- Simple API with easy-to-use functions.

5.4 Security

- **Handles collisions** to prevent data loss.

CODE:

```
class Node:
```

```
    """Node for storing key-value pairs in a linked list (for chaining)."""
```

```
    def __init__(self, key, value):
```

```
        self.key = key
```

```
        self.value = value
```

```
        self.next = None
```

```

class HashTable:
    """Custom Hash Table with Chaining and Dynamic
    Resizing."""
    def __init__(self, size=10):
        self.size = size
        self.table = [None] * self.size
        self.count = 0 # Track number of elements
        self.load_factor_threshold = 0.7 # Resize if exceeded

    def _hash(self, key):
        """Hash function using modulo operation."""
        return hash(key) % self.size

    def _resize(self):
        """Doubles the table size and rehashes all elements."""
        new_size = self.size * 2
        new_table = [None] * new_size
        old_table = self.table
        self.table = new_table
        self.size = new_size
        self.count = 0 # Reset count and reinsert

        for node in old_table:
            while node:
                self.insert(node.key, node.value)
                node = node.next

    def insert(self, key, value):
        """Inserts a key-value pair into the hash table."""
        index = self._hash(key)
        node = self.table[index]

```

```
while node:
    if node.key == key:
        node.value = value # Update existing key
        return
    node = node.next

# Insert new node at the head (chaining)
new_node = Node(key, value)
new_node.next = self.table[index]
self.table[index] = new_node
self.count += 1

# Check if resizing is needed
if self.count / self.size > self.load_factor_threshold:
    self._resize()

def search(self, key):
    """Searches for a key in the hash table."""
    index = self._hash(key)
    node = self.table[index]

    while node:
        if node.key == key:
            return node.value
        node = node.next

    return None # Key not found

def delete(self, key):
    """Deletes a key-value pair from the hash table."""
```

```

index = self._hash(key)
node = self.table[index]
prev = None

while node:
    if node.key == key:
        if prev:
            prev.next = node.next
        else:
            self.table[index] = node.next
        self.count -= 1
        return True # Key deleted
    prev = node
    node = node.next

```

```

return False # Key not found

```

```

def display(self):
    """Displays the hash table contents."""
    for i, node in enumerate(self.table):
        print(f"Index {i}: ", end="")
        while node:
            print(f"({node.key}: {node.value}) -> ", end="")
            node = node.next
        print("None")

```

```

# Example Usage

```

```

ht = HashTable()
ht.insert("apple", 100)
ht.insert("banana", 200)
ht.insert("orange", 300)

```



```
ht.insert("grape", 400)
```

```
ht.display()
```

```
print("Search for 'banana':", ht.search("banana")) #
```

```
Output: 200
```

```
ht.delete("banana")
```

```
ht.display()
```

CODE PICTURE:

```

1 class Node:
2     """Node for storing key-value pairs in a linked list (for chaining)."""
3     def __init__(self, key, value):
4         self.key = key
5         self.value = value
6         self.next = None
7
8 class HashTable:
9     """Custom Hash Table with Chaining and Dynamic Resizing."""
10    def __init__(self, size=10):
11        self.size = size
12        self.table = [None] * self.size
13        self.count = 0 # Track number of elements
14        self.load_factor_threshold = 0.7 # Resize if exceeded
15
16    def _hash(self, key):
17        """Hash function using modulo operation."""
18        return hash(key) % self.size
19
20    def _resize(self):
21        """Doubles the table size and rehashes all elements."""
22        new_size = self.size * 2
23        new_table = [None] * new_size
24        old_table = self.table
25        self.table = new_table
26        self.size = new_size
27        self.count = 0 # Reset count and reinsert
28
29        for node in old_table:
30            while node:
31                self.insert(node.key, node.value)
32                node = node.next
33
34    def insert(self, key, value):
35        """Inserts a key-value pair into the hash table."""
36        index = self._hash(key)
37        node = self.table[index]

```

```

8 class HashTable:
9     def insert(self, key, value):
10         node = self.table[index]
11
12         while node:
13             if node.key == key:
14                 node.value = value # Update existing key
15                 return
16             node = node.next
17
18         # Insert new node at the head (chaining)
19         new_node = Node(key, value)
20         new_node.next = self.table[index]
21         self.table[index] = new_node
22         self.count += 1
23
24         # Check if resizing is needed
25         if self.count / self.size > self.load_factor_threshold:
26             self._resize()
27
28     def search(self, key):
29         """Searches for a key in the hash table."""
30         index = self._hash(key)
31         node = self.table[index]
32
33         while node:
34             if node.key == key:
35                 return node.value
36             node = node.next
37
38         return None # Key not found
39
40     def delete(self, key):
41         """Deletes a key-value pair from the hash table."""
42         index = self._hash(key)
43         node = self.table[index]
44         prev = None

```

```

45         while node:
46             if node.key == key:
47                 if prev:
48                     prev.next = node.next
49                 else:
50                     self.table[index] = node.next
51                 self.count -= 1
52                 return True # Key deleted
53             prev = node
54             node = node.next
55
56         return False # Key not found
57
58     def display(self):
59         """Displays the hash table contents."""
60         for i, node in enumerate(self.table):
61             print(f"Index {i}: ", end="")
62             while node:
63                 print(f"({node.key}: {node.value}) -> ", end="")
64                 node = node.next
65             print("None")
66
67     # Example Usage
68     ht = HashTable()
69     ht.insert("apple", 100)
70     ht.insert("banana", 200)
71     ht.insert("orange", 300)
72     ht.insert("grape", 400)
73     ht.display()
74
75     print("Search for 'banana':", ht.search("banana")) # Output: 200
76     ht.delete("banana")
77     ht.display()

```

OUT PUT:

The screenshot shows a Python IDE with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The code editor displays a Python script for a HashTable implementation. The terminal shows the output of the script, which includes the insertion of 'apple', 'banana', 'orange', and 'grape' into the hash table, followed by a search for 'banana' and a delete operation.

```
File Edit Selection View Go Run Terminal Help Documents
EXPLORER
DOCUMENTS
  Custom Office Templ...
  GitHub
  desktop-tutorial
  muhammad-haseeb...
  name
  KingsoftData
  WPS Cloud Files
  -$sued document 1...
  desktop.ini
  fdfgldldldldfcoodh...
  fdfgldldldldfcoodh...
  muhammad_haseeb...
  muhammad_haseeb...
  muhammad_haseeb...
  muhammad_haseeb...
  Rescued document 1...
  Rescued document.doc
  Untitled-1.html
  Untitled-1.py
  Untitled-2.py

C:\Users\hp> OneDrive> Desktop> Untitled-1.py
97 ht.insert("apple", 100)
98 ht.insert("banana", 200)
99 ht.insert("orange", 300)
100 ht.insert("grape", 400)
101 ht.display()
102
103 print("Search for 'banana':", ht.search("banana")) # Output: 200
104 ht.delete("banana")
105 ht.display()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\hp\OneDrive\Documents> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe c:/Users/hp/OneDrive/Desktop/Untitled-1.py
Index 0: (banana: 200) -> None
Index 1: (orange: 300) -> (apple: 100) -> None
Index 2: None
Index 3: None
Index 4: None
Index 5: None
Index 6: (grape: 400) -> None
Index 7: None
Index 8: None
Index 9: None
Search for 'banana': 200
Index 0: None
Index 1: (orange: 300) -> (apple: 100) -> None
Index 2: None
Index 3: None
Index 4: None
Index 5: None
Index 6: (grape: 400) -> None
Index 7: None
Index 8: None
Index 9: None
PS C:\Users\hp\OneDrive\Documents>
```