AISSMS
INSTITUTE OF INFORMATION TECHNOLOGY
ADDING VALUE TO ENGINEERING

Department Of Computer Engineering

Data Structure And Algorithms Lab

Group-A

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AISSMS IOIT

SE COMPTER ENGINEERING

SUBMITTED BY

Kaustubh S Kabra
ERP No.- 34
Teams No.-20

2020 -2021

Experiment Number:-1

- ***Experiment Name:-*** *Chaining method and Addressing method on Hash Table.*
- ***Aim:-*** *Consider telephone book database of N clients. Make use of a hash table implementation to quickly look up client's telephone number. Make use of two collision handling techniques and compare them using number of comparisons required to find a set of telephone numbers.*
- ***Objective:-***
    1) *To understand working of hash table.*
    2) *To implement program using Chaining method and Addressing method.*
- ***Theory:-***

Hash Table:

The Hash table data structure stores elements in key-value pairs where

- *Key- unique integer that is used for indexing the values*
- *Value - data that are associated with keys.*

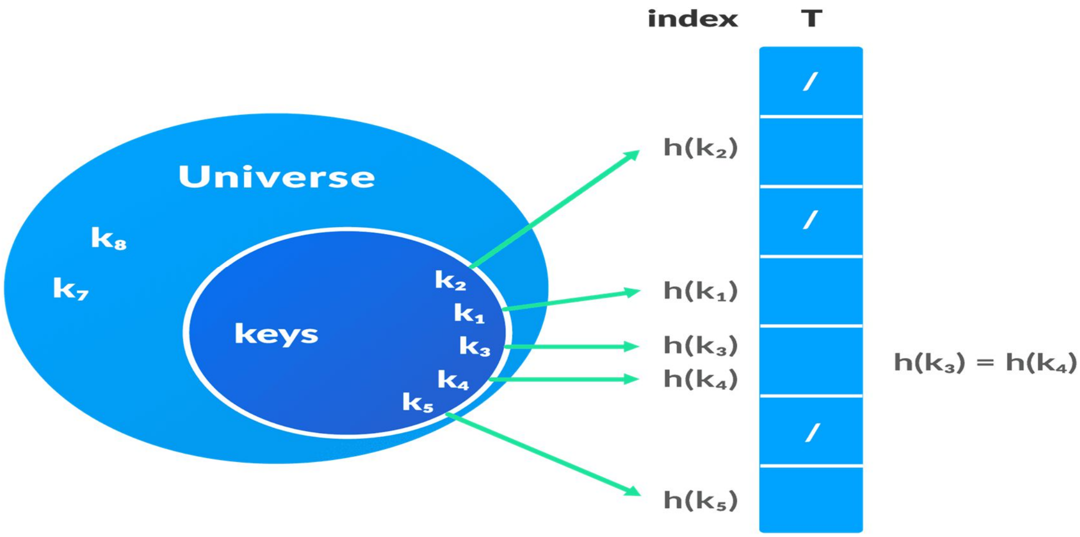
Hashing (Hash Function):

In a hash table, a new index is processed using the keys. And, the element corresponding to that key is stored in the index. This process is called hashing.

Let k be a key and h(x) be a hash function.

Here, h(k) will give us a new index to store the element linked with k.

Hashing is a technique of mapping a large set of arbitrary data to tabular indexes using a hash function. It is a method for representing dictionaries for large datasets.



Hash Collision:

When the hash function generates the same index for multiple keys, there will be a conflict (what value to be stored in that index). This is called a hash collision.

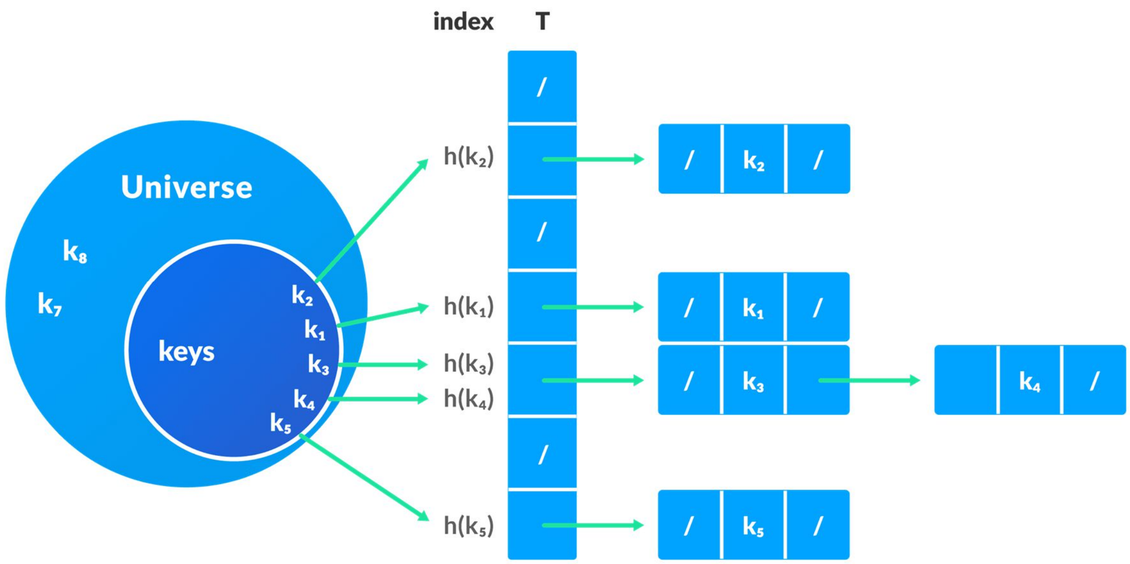
We can resolve the hash collision using one of the following techniques.

1. *Collision resolution by chaining*
2. *Open Addressing: Linear/Quadratic Probing and Double Hashing*

Collision resolution by chaining

In chaining, if a hash function produces the same index for multiple elements, these elements are stored in the same index by using a doubly-linked list.

If j is the slot for multiple elements, it contains a pointer to the head of the list of elements. If no element is present, j contains NIL.



Open Addressing

Unlike chaining, open addressing doesn't store multiple elements into the same slot. Here, each slot is either filled with a single key or left NIL.

Different techniques used in open addressing are:

- ***Linear Probing***

In linear probing, collision is resolved by checking the next slot.

h(k, i) = (h′(k) + i) % m

wherei = {0, 1, ….}, h′(k) is a new hash function

If a collision occurs at h(k, 0), then h(k, 1) is checked. In this way, the value of i is incremented linearly.

The problem with linear probing is that a cluster of adjacent slots is filled. When inserting a new element, the entire cluster must be traversed. This adds to the time required to perform operations on the hash table.

- ***Quadratic Probing***

It works similar to linear probing but the spacing between the slots is increased (greater than one) by using the following relation.

h(k, i) = (h′(k) + c1i + c2i2) % m

where,c1 and c2 are positive auxiliary constants,i = {0, 1, ….}

- ***Double hashing***

If a collision occurs after applying a hash function h(k), then another hash function is calculated for finding the next slot.

h(k, i) = (h1(k) + ih2(k)) % m

Applications of Hash Table:

Hash tables are implemented where

1. *constant time lookup and insertion is required*
2. *cryptographic applications indexing data is required*

- ## *Algorithm:-*

  1. *Start / Run*
  2. *selecting Chaining method .*
  3. *Declare an array of a linked list with the hash table size.*

4.Initialize an array of a linked list to NULL.
/// inserting :
5. Find hash key.
6. If chain[key] == NULL
Make chain[key] points to the key node.
7.Otherwise(collision),
Insert the key node at the end of the chain[key].
/// searching :
8. Get the value
9. Compute the hash key.
10. Search the value in the entire chain. i.e. chain[key].
11. If found, print "Search Found"
12. Otherwise, print "Search Not Found"
///Removing :
13. Get the value
14. Compute the key.
15. Using linked list deletion algorithm, delete the element from the chain[key].
Linked List Deletion Algorithm: Deleting a node in the linked list
16. If unable to delete, print "Value Not Found"
17.selecting linear probing (addressing):
/// inserting
18.use hash function to find index for a record
19.If that spot is already in use, we use next available spot in a "higher" index.
20. Treat the hash table as if it is round, if you hit the end of the hash table, go back to the front
///searching
21.use hash function to find index of where an item should be.

22.If it isn't there search records that records after that hash location (remember to treat table as cicular) until either it found, or until an empty record is found. If there is an empty spot in the table before record is found, it means that the the record is not there.

/// removing :

23.determine the hash index of the record

24.find record and remove it making the spot empty

25. Stop

- ***Program:-***

```
class hashing_chaining:
    arr = []

    def __init__(self, num):
        self.MAX = num
        self.arr = [[] for i in range(self.MAX)]

    def get_hash(self, key):
        h = 0
        for char in key:
            h += ord(char)
        return h % self.MAX

    def __setitem__(self, key, val):
        h = self.get_hash(key)
        found = False

        for idx, elem in enumerate(self.arr[h]):
            if len(elem) == 2 and elem[0] == key:
                self.arr[h][idx] = (key, val)
                found = True
                break

        if not found:
            self.arr[h].append((key, val))  # not exist

    def __getitem__(self, key):
        h = self.get_hash(key)
        for ele in self.arr[h]:
            if ele[0] == key:
                return ele[1]
        return None

    def __delitem__(self, key):
        h = self.get_hash(key)
        for index, key_val in enumerate(self.arr[h]):
            if key_val[0] == key:
                del self.arr[h][index]
                print('Item Deleted ')
```

```python
                break
        else:
            print('Item Not Found ')


class hashing_addressing:
    def __init__(self, num):
        self.MAX = num
        self.arr = [None for i in range(self.MAX)]
        print(len(self.arr))

    def get_hash(self, key):
        hash = 0
        for char in key:
            hash += ord(char)
        return hash % self.MAX

    def __getitem__(self, key):
        count_jump = 0
        h = self.get_hash(key)
        if self.arr[h] is None:
            return
        prob_range = self.get_prob_range(h)
        for prob_index in prob_range:
            count_jump += 1
            element = self.arr[prob_index]
            if element is None:
                return
            if element[0] == key:
                return element[1], count_jump

    def __setitem__(self, key, val):
        h = self.get_hash(key)
        if self.arr[h] is None:
            self.arr[h] = (key, val)
        else:
            new_h = self.find_slot(key, h)
            self.arr[new_h] = (key, val)
        # print(self.arr)

    def get_prob_range(self, index):
        return [*range(index, len(self.arr))] + [*range(0, index)]

    def find_slot(self, key, index):
        prob_range = self.get_prob_range(index)
        for prob_index in prob_range:
            if self.arr[prob_index] is None:
                return prob_index
            if self.arr[prob_index][0] == key:
                return prob_index
        raise Exception("Hashmap full")

    def __delitem__(self, key):
        h = self.get_hash(key)
        prob_range = self.get_prob_range(h)
        for prob_index in prob_range:
            if self.arr[prob_index] is None:
```

```
                print('Item Not Found ')
                break  # item not found so return. You can also throw
exception
            if self.arr[prob_index][0] == key:
                self.arr[prob_index] = None
                print('Item Deleted ')
                break
        # print(self.arr)


if __name__ == '__main__':
    try:
        while True:
            userinput1 = int(input(
                '\nWelcome to Hashing Table \nEnter which hashing table
type to apply\n1.CHAINING \n2.ADDRESSING \n3.EXIT\n==>'))

            if userinput1 == 1:
                size1 = int(input("ENTER SIZE OF TABLE :"))
                ch = hashing_chaining(size1)
                while True:
                    userinput2 = int(input("\n1.INSERT
\n2.SEARCH\n3.Display HASHING TABLE\n4.DELETE\n5.EXIT\n>>>> "))
                    if userinput2 == 1:
                        numb = int(input("ENTER NUMBER OF CLIENT:"))
                        for i in range(numb):
                            name1 = input("ENTER NAME OF CLIENT:")
                            number1 = int(input("ENTER PHONE NUMBER OF
CLIENT:"))
                            ch[name1] = number1
                    elif userinput2 == 2:
                        name1 = input("ENTER NAME OF CLIENT:")
                        print(ch[name1])
                    elif userinput2 == 3:
                        print(ch.arr)
                    elif userinput2 == 4:
                        name1 = input("ENTER NAME OF CLIENT:")
                        del ch[name1]

                    elif userinput2 == 5:
                        break
                    else:
                        print("WRONG INPUT")

            elif userinput1 == 2:
                size2 = int(input("ENTER SIZE OF TABLE :"))
                ah = hashing_addressing(size2)
                while True:
                    userinput2 = int(
                        input("\n1.INSERT\n2.SEARCH\n3.Display HASHING
TABLE\n4.DELETE\n5.EXIT\n>>>> "))

                    if userinput2 == 1:
                        numb = int(input("ENTER NUMBER OF CLIENT:"))
                        for i in range(numb):
                            name1 = input("ENTER NAME OF CLIENT:")
                            number1 = int(input("ENTER PHONE NUMBER OF
```

```
CLIENT:"))
                            ah[name1] = number1

                    elif userinput2 == 2:
                        name1 = input("ENTER NAME OF CLIENT:")
                        element, count = ah[name1]
                        print('Number is :', element, '\tRequired 
number of jumps:', count)

                    elif userinput2 == 3:
                        print(ah.arr)

                    elif userinput2 == 4:
                        name1 = input("ENTER NAME OF CLIENT:")
                        del ah[name1]

                    elif userinput2 == 5:
                        break
                    else:
                        print("WRONG INPUT")
                elif userinput1 == 3:
                    break
                else:
                    print('Wrong Input')
        except Exception as e:
            print('\nWrong input :', e)
```

- ***Output:-***

C:\Users\asus\AppData\Local\Programs\Python\Python38\python.exe "F:/4th Sem/DSAL/Group A/hashing_Kaustubh34(Teams-20).py"

Welcome to Hashing Table

Enter which hashing table type to apply

1.CHAINING

2.ADDRESSING

3.EXIT

==>1

ENTER SIZE OF TABLE :10

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 1

ENTER NUMBER OF CLIENT:7

ENTER NAME OF CLIENT:Kaustubh

ENTER PHONE NUMBER OF CLIENT:9168100204

ENTER NAME OF CLIENT:Harsh

ENTER PHONE NUMBER OF CLIENT:9874563210

ENTER NAME OF CLIENT:Onasvee

ENTER PHONE NUMBER OF CLIENT:7410852963

ENTER NAME OF CLIENT:Akash

ENTER PHONE NUMBER OF CLIENT:3214056789

ENTER NAME OF CLIENT:KK1

ENTER PHONE NUMBER OF CLIENT:9875632104

ENTER NAME OF CLIENT:Lonewolf

ENTER PHONE NUMBER OF CLIENT:7531598240

ENTER NAME OF CLIENT:Orion

ENTER PHONE NUMBER OF CLIENT:9517538264

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2

ENTER NAME OF CLIENT:KK1

9875632104

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2

ENTER NAME OF CLIENT:Lonewolf

7531598240

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2

ENTER NAME OF CLIENT:Orion

9517538264

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2

ENTER NAME OF CLIENT:acbd

None

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 3

[[], [('Onasvee', 7410852963)], [('Harsh', 9874563210)], [], [], [], [], [], [('Akash', 3214056789), ('Lonewolf', 7531598240)], [('Kaustubh', 9168100204), ('KK1', 9875632104), ('Orion', 9517538264)]]

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 4

ENTER NAME OF CLIENT:KK1

Item Deleted

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 3

[[], [('Onasvee', 7410852963)], [('Harsh', 9874563210)], [], [], [], [], [], [('Akash', 3214056789), ('Lonewolf', 7531598240)], [('Kaustubh', 9168100204), ('Orion', 9517538264)]]

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 4

ENTER NAME OF CLIENT:Lonewolf

Item Deleted

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 3

[[], [('Onasvee', 7410852963)], [('Harsh', 9874563210)], [], [], [], [], [], [('Akash', 3214056789)], [('Kaustubh', 9168100204), ('Orion', 9517538264)]]

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 5

Welcome to Hashing Table

Enter which hashing table type to apply

1.CHAINING

2.ADDRESSING

3.EXIT

==>2

ENTER SIZE OF TABLE :10

10

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 1

ENTER NUMBER OF CLIENT:7

ENTER NAME OF CLIENT:Kaustubh

ENTER PHONE NUMBER OF CLIENT:9168100204

ENTER NAME OF CLIENT:Harsh

ENTER PHONE NUMBER OF CLIENT:7894561230

ENTER NAME OF CLIENT:Onasvee

ENTER PHONE NUMBER OF CLIENT:7410852963

ENTER NAME OF CLIENT:Akash

ENTER PHONE NUMBER OF CLIENT:9517538264

ENTER NAME OF CLIENT:KK1

ENTER PHONE NUMBER OF CLIENT:9875632144

ENTER NAME OF CLIENT:Lonewolf

ENTER PHONE NUMBER OF CLIENT:8527413330

ENTER NAME OF CLIENT:Orion

ENTER PHONE NUMBER OF CLIENT:2583571593

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 3

[('KK1', 9875632144), ('Onasvee', 7410852963), ('Harsh', 7894561230), ('Lonewolf', 8527413330), ('Orion', 2583571593), None, None, None, ('Akash', 9517538264), ('Kaustubh', 9168100204)]

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2

ENTER NAME OF CLIENT:Kaustubh

Number is : 9168100204 Required number of jumps: 1

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2

ENTER NAME OF CLIENT:Lonewolf

Number is : 8527413330 Required number of jumps: 6

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2

ENTER NAME OF CLIENT:KK1

Number is : 9875632144 Required number of jumps: 2

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 2

ENTER NAME OF CLIENT:abcd

None

1.INSERT

2.SEARCH

3.Display HASHING TABLE

4.DELETE

5.EXIT

>>>> 5

Process finished with exit code 0

- ## *Analysis:-*

Worst-Case Time Complexity (Linear Probing)
Find: O(n)
Insert: O(n)
Remove: O(n)

Average-Case Time Complexity (Linear Probing)
Find: O(1)
Insert: O(1)
Remove: O(1)

Best-Case Time Complexity (Linear Probing)
Find: O(1) — No collisions
Insert: O(1) — No collisions
Remove: O(1) — No collisions

Space Complexity (Linear Probing)
O(n)

///////
Worst-Case Time Complexity (Separate Chaining)
Find: O(n) — If all the keys mapped to the same index (assuming Linked List)
Insert: O(n) — If all the keys mapped to the same index (assuming Linked List) and we check for duplicates
Remove: O(n) — If all the keys mapped to the same index (assuming Linked List)

Average-Case Time Complexity (Separate Chaining)
Find: O(1)
Insert: O(1)
Remove: O(1)

Best-Case Time Complexity (Separate Chaining)
Find: O(1) — No collisions
Insert: O(1) — No collisions
Remove: O(1) — No collisions

Space Complexity (Separate Chaining)
O(n) — Hash Tables typically have a capacity that is at most some constant multiplied by n
(the constant is predetermined), and each of our n nodes occupies O(1) space

- ***Conclusion:-*** *Hence, we have studied and implemented hashing, We have studied different collision handling techniques on our telephone book database and have successfully implemented them.*

Experiment Number:-2

- ***Experiment Name:-*** *Chaining with and without replacement method*

- ***Aim:-*** *Implement all the functions of a dictionary (ADT) using hashing and handle collisions using chaining with / without replacement.*

Data: Set of (key, value) pairs, keys are mapped to values, keys must be comparable, keys must be unique.
Standard operations: insert (key, value), Find (key), Delete (key).

- ***Objective:-***
  3) *To understand working of hash table.*
  4) *To implement program using Chaining with and without replacement method .*

- ***Theory:-***

Hash Table:

The Hash table data structure stores elements in key-value pairs where

- *Key- unique integer that is used for indexing the values*
- *Value - data that are associated with keys.*

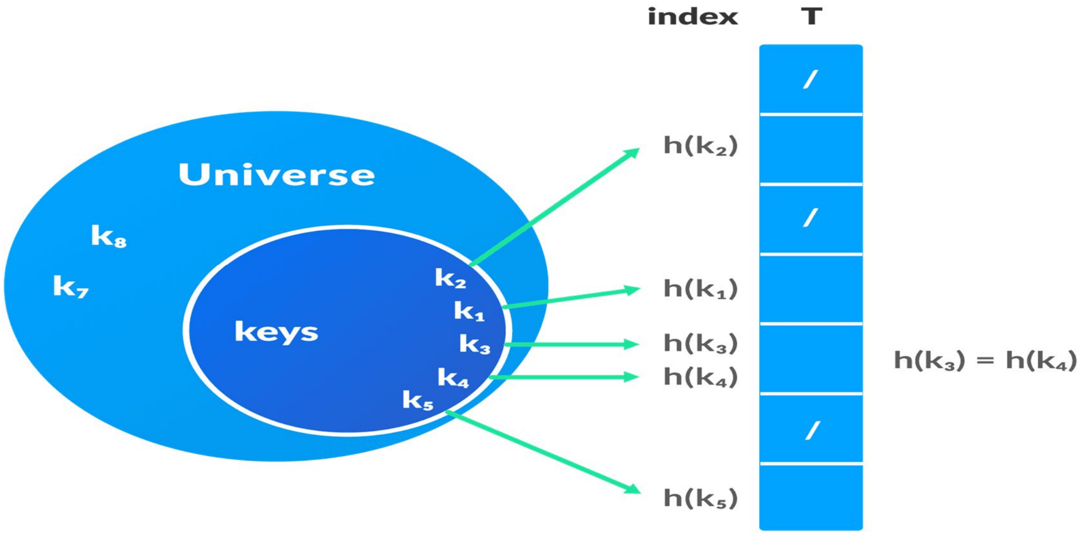
Hashing (Hash Function):

In a hash table, a new index is processed using the keys. And, the element corresponding to that key is stored in the index. This process is called hashing.

Let k be a key and h(x) be a hash function.

Here, h(k) will give us a new index to store the element linked with k.

Hashing is a technique of mapping a large set of arbitrary data to tabular indexes using a hash function. It is a method for representing dictionaries for large datasets.



<u>Hash Collision:</u>

When the hash function generates the same index for multiple keys, there will be a conflict (what value to be stored in that index). This is called a hash collision.

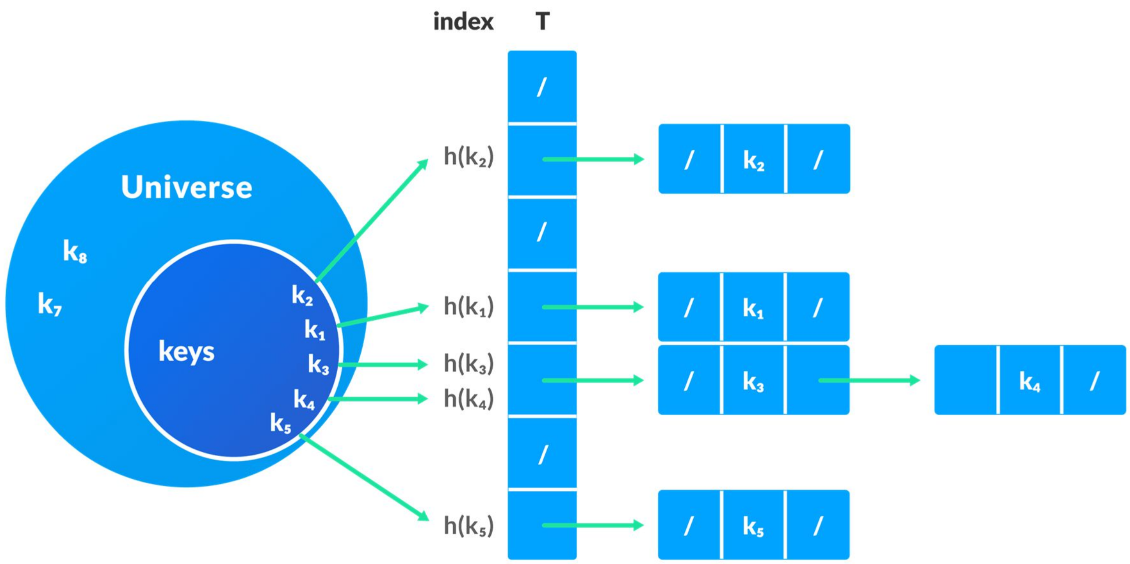
We can resolve the hash collision using one of the following techniques.

3. *Collision resolution by chaining*
4. *Open Addressing: Linear/Quadratic Probing and Double Hashing*

Collision resolution by chaining

In chaining, if a hash function produces the same index for multiple elements, these elements are stored in the same index by using a doubly-linked list.

If j is the slot for multiple elements, it contains a pointer to the head of the list of elements. If no element is present, j contains NIL.



Applications of Hash Table:

Hash tables are implemented where

3. *constant time lookup and insertion is required*
4. *cryptographic applications indexing data is required*

- ## ***Algorithm:-***

1.Start

2. Taking user input

3.setting dictionary size and creating empty list

4.

/// inserting :

5. Find hash key.

6. If chain[key] == NULL

Make chain[key] points to the key node.

7.Otherwise(collision),

Insert the key node at the end of the chain[key].

/// searching :

8. Get the value

9. Compute the hash key.

10. Search the value in the entire chain. i.e. chain[key].

11. If found, print "Search Found"

12. Otherwise, print "Search Not Found"

///Removing :

13. Get the value

14. Compute the key.

15. Using linked list deletion algorithm, delete the element from the chain[key].

Linked List Deletion Algorithm: Deleting a node in the linked list

16. If unable to delete, print "Value Not Found"

17. using for loop for printing the dict

18. Stop

- ***Program:-***

```
class Dictionary(object):
    def __init__(self, size):
        self.dict = [None] * size
        self.length = 0

    def __del__(self):
        pass

    def insert(self, key, value):
        hashCode = hash(key) % size
        lis = [(key, value)]

        if self.dict[hashCode] is None:
            self.dict[hashCode] = lis
            self.length += 1

        else:
            tup = (key, value)
            i = 0
            for j in range(size):
                if i == len(self.dict[hashCode]):
                    self.length += 1
                    self.dict[hashCode].append(tup)
                    return

                if self.dict[hashCode][i][0] == key:
                    self.dict[hashCode][i] = tup
```

```python
                    return
                i += 1

            else:
                self.length += 1
                self.dict[hashCode].append(tup)

    def find(self, key):
        hashCode = hash(key) % size
        i = 0

        if self.dict[hashCode] == None:
            return -1

        elif self.dict[hashCode][i][0] == key:
            return self.dict[hashCode][i][1]

        else:
            i += 1
            for j in range(size):
                if i == len(self.dict[hashCode]):
                    return -1

                if self.dict[hashCode][i][0] == key:
                    return self.dict[hashCode][i][1]
                i += 1

            else:
                return -1

    def delete(self, key):
        hashCode = hash(key) % size
        i = 0

        if self.dict[hashCode] == None:
            return -1

        elif self.dict[hashCode][i][0] == key:
            if len(self.dict[hashCode]) == 1:
                self.length -= 1
                self.dict[hashCode] = None

            else:
                self.length -= 1
                self.dict[hashCode].remove(self.dict[hashCode][i])

        else:
            i += 1
            for j in range(size):
                if i == len(self.dict[hashCode]):
                    return -1

                if self.dict[hashCode][i][0] == key:
                    if len(self.dict[hashCode]) == 1:
                        self.length -= 1
                        self.dict[hashCode] = None
```

```
                        else:
                            self.length -= 1
  
self.dict[hashCode].remove(self.dict[hashCode][i])
                            return
                    i += 1

            else:
                return -1

    def printDict(self):
        print("\n{ ", end="")
        i = 1
        for lis in self.dict:
            if lis != None:
                for ele in lis:
                    if i == self.length:
                        print(f"{ele[0]} : {ele[1]}", end="")
                    else:
                        print(f"{ele[0]} : {ele[1]}, ", end="")
                    i += 1
        print(" }")


if __name__ == '__main__':
    try:
        size = int(input("\nEnter The Size of Dictionary: "))
        dt = Dictionary(size)

        while True:
            userInput = int(input("\n1. Press 1 To Insert(Key, 
Value)\n2. Press 2 To Find(Key)\n3. Press 3 To Delete("
                                  "Key)\n4. Press 4 To Print 
Dictionary\n5. Press 5 To Exit\n>>>> "))
            if userInput == 1:
                while True:
                    selectKey = int(input("\nSelect The Data Type of 
Key: \n1. String\n2. Integer\n3. Float\n>>>> "))
                    selectValue = int(
                        input("\nSelect The Data Type of Value: \n1. 
String\n2. Integer\n3. Float\n>>>> "))

                    if selectKey == 1 and selectValue == 1:
                        key = input("\nEnter The Key: ")
                        value = input("\nEnter The Value: ")
                        dt.insert(key, value)
                        break

                    elif selectKey == 1 and selectValue == 2:
                        key = input("\nEnter The Key: ")
                        value = int(input("\nEnter The Value: "))
                        dt.insert(key, value)
                        break

                    elif selectKey == 1 and selectValue == 3:
                        key = input("\nEnter The Key: ")
                        value = float(input("\nEnter The Value: "))
```

```
                        dt.insert(key, value)
                        break

                    elif selectKey == 2 and selectValue == 1:
                        key = int(input("\nEnter The Key: "))
                        value = input("\nEnter The Value: ")
                        dt.insert(key, value)
                        break

                    elif selectKey == 2 and selectValue == 2:
                        key = int(input("\nEnter The Key: "))
                        value = int(input("\nEnter The Value: "))
                        dt.insert(key, value)
                        break

                    elif selectKey == 2 and selectValue == 3:
                        key = int(input("\nEnter The Key: "))
                        value = float(input("\nEnter The Value: "))
                        dt.insert(key, value)
                        break

                    elif selectKey == 3 and selectValue == 1:
                        key = float(input("\nEnter The Key: "))
                        value = input("\nEnter The Value: ")
                        dt.insert(key, value)
                        break

                    elif selectKey == 3 and selectValue == 2:
                        key = float(input("\nEnter The Key: "))
                        value = int(input("\nEnter The Value: "))
                        dt.insert(key, value)
                        break

                    elif selectKey == 3 and selectValue == 3:
                        key = float(input("\nEnter The Key: "))
                        value = float(input("\nEnter The Value: "))
                        dt.insert(key, value)
                        break

                    else:
                        print("\nPlease Enter Correct Input!!")
                        continue

            elif userInput == 2:
                selectKey = int(input("\nSelect The Data Type of Key:
\n1. String\n2. Integer\n3. Float\n>>>> "))
                while True:
                    if selectKey == 1:
                        key = input("\nEnter The Key: ")
                        if dt.find(key) != -1:
                            print("\n{", f"{key} : {dt.find(key)}",
"}")
                        else:
                            print(dt.find(key))
                        break

                    elif selectKey == 2:
```

```
                        key = int(input("\nEnter The Key: "))
                        if dt.find(key) != -1:
                            print("\n{", f"{key} : {dt.find(key)}",
"}")
                        else:
                            print(dt.find(key))
                        break

                    elif selectKey == 3:
                        key = float(input("\nEnter The Key: "))
                        if dt.find(key) != -1:
                            print("\n{", f"{key} : {dt.find(key)}",
"}")
                        else:
                            print(dt.find(key))
                        break

                    else:
                        print("\nPlease Enter Correct Input!!")
                        continue

            elif userInput == 3:
                selectKey = int(input("\nSelect The Data Type of Key:
\n1. String\n2. Integer\n3. Float\n>>>> "))
                while True:
                    if selectKey == 1:
                        key = input("\nEnter The Key: ")
                        dt.delete(key)
                        break

                    elif selectKey == 2:
                        key = int(input("\nEnter The Key: "))
                        dt.delete(key)
                        break

                    elif selectKey == 3:
                        key = float(input("\nEnter The Key: "))
                        dt.delete(key)
                        break

                    else:
                        print("\nPlease Enter Correct Input!!")
                        continue

            elif userInput == 4:
                dt.printDict()

            elif userInput == 5:
                exit()

            else:
                print("\nPlease Enter Correct Input!!")

    except Exception as e:
        print("\nWrong Input,", e)
```

- ***Output:-***

Enter The Size of Dictionary: 10

1. Press 1 To Insert(Key, Value)

2. Press 2 To Find(Key)

3. Press 3 To Delete(Key)

4. Press 4 To Print Dictionary

5. Press 5 To Exit

>>>> 1

Select The Data Type of Key:

1. String

2. Integer

3. Float

>>>> 1

Select The Data Type of Value:

1. String

2. Integer

3. Float

>>>> 2

Enter The Key: orion

Enter The Value: 22

1. Press 1 To Insert(Key, Value)

2. Press 2 To Find(Key)

3. Press 3 To Delete(Key)

4. Press 4 To Print Dictionary

5. Press 5 To Exit

>>>> 1

Select The Data Type of Key:

1. String

2. Integer

3. Float

>>>> 1

Select The Data Type of Value:

1. String

2. Integer

3. Float

>>>> 3

Enter The Key: kk

Enter The Value: 22.5

1. Press 1 To Insert(Key, Value)

2. Press 2 To Find(Key)

3. Press 3 To Delete(Key)

4. Press 4 To Print Dictionary

5. Press 5 To Exit

>>>> 1

Select The Data Type of Key:

1. String

2. Integer

3. Float

>>>> 3

Select The Data Type of Value:

1. String

2. Integer

3. Float

>>>> 1

Enter The Key: 12345

Enter The Value: hS

1. Press 1 To Insert(Key, Value)

2. Press 2 To Find(Key)

3. Press 3 To Delete(Key)

4. Press 4 To Print Dictionary

5. Press 5 To Exit

>>>> 2

Select The Data Type of Key:

1. String

2. Integer

3. Float

>>>> 1

Enter The Key: orion

{ orion : 22 }

1. Press 1 To Insert(Key, Value)

2. Press 2 To Find(Key)

3. Press 3 To Delete(Key)

4. Press 4 To Print Dictionary

5. Press 5 To Exit

>>>> 2

Select The Data Type of Key:

1. String

2. Integer

3. Float

>>>> 2

Enter The Key: 33

-1

1. Press 1 To Insert(Key, Value)

2. Press 2 To Find(Key)

3. Press 3 To Delete(Key)

4. Press 4 To Print Dictionary

5. Press 5 To Exit

>>>> 3

Select The Data Type of Key:

1. String

2. Integer

3. Float

>>>> 1

Enter The Key: orion

1. Press 1 To Insert(Key, Value)

2. Press 2 To Find(Key)

3. Press 3 To Delete(Key)

4. Press 4 To Print Dictionary

5. Press 5 To Exit

>>>> 2

Select The Data Type of Key:

1. String

2. Integer

3. Float

>>>> 1

Enter The Key: orion

-1

1. Press 1 To Insert(Key, Value)

2. Press 2 To Find(Key)

3. Press 3 To Delete(Key)

4. Press 4 To Print Dictionary

5. Press 5 To Exit

>>>> 4

{ kk : 22.5, 12345.0 : hS }

1. Press 1 To Insert(Key, Value)

2. Press 2 To Find(Key)

3. Press 3 To Delete(Key)

4. Press 4 To Print Dictionary

5. Press 5 To Exit

>>>> 5

- ***Analysis:-***

  Time complexity

  Worst-Case Time Complexity (Separate Chaining)

  Find: O(n) — If all the keys mapped to the same index (assuming Linked List)

  Insert: O(n) — If all the keys mapped to the same index (assuming Linked List) and we check for duplicates

Remove: O(n) — If all the keys mapped to the same index (assuming Linked List)

Average-Case Time Complexity (Separate Chaining)
Find: O(1)
Insert: O(1)
Remove: O(1)

Best-Case Time Complexity (Separate Chaining)
Find: O(1) — No collisions
Insert: O(1) — No collisions
Remove: O(1) — No collisions

Space Complexity (Separate Chaining)
O(n) — Hash Tables typically have a capacity that is at most some constant multiplied by n (the constant is predetermined), and each of our n nodes occupies O(1) space

- ***Conclusion:-*** *Hence, we have studied and implemented hashing, We have studied collision handling techniques using chanining with and without replacement and have successfully implemented them.*