```
# 21118 - Shubham (E-1)
class Entry:
    def __init__(self, word=" ", meaning=" ", chain=-1):
        self.word = word
        self.meaning = meaning
        self.chain = chain


class HashTable():
    def __init__(self, n):
        self.SIZE = 0
        self.ht = []
        self.MAX_SIZE = n
        for i in range(0, n):
            self.ht.append(Entry())

    def hashFun(self, key):
        res = 0
        for i in range(0, len(key)):
            res += i*i
        return res % self.MAX_SIZE

    def insertWithoutReplacement(self, key, val):
        if self.SIZE == self.MAX_SIZE:  # Avoid Overflow
            print("*Dictionary is Full*\n")
            return None

        idx = self.hashFun(key)
        if self.ht[idx].word == " ":
            self.ht[idx] = Entry(key, val)
            print("*Inserted Successfully*\n")
            self.SIZE += 1
        else:
            inti = idx
            while self.hashFun(self.ht[idx].word) != inti and self.ht[idx].wor
d != " ":
                idx = (idx+1) % self.MAX_SIZE
            if self.ht[idx].word == " ":
                self.ht[idx] = Entry(key, val)
                print("*Inserted Successfully*\n")
                self.SIZE += 1
            else:
                while self.ht[idx].chain != -1:
                    if self.ht[idx].word == key:
                        print("*Word Already Exist in Dictionary*\n")
                        return None
                    idx = self.ht[idx].chain
                if self.ht[idx].word == key:
                    print("*Word Already Exist in Dictionary*\n")
                    return None
                prevIdx = idx
```

```
                while self.ht[idx].word != " ":
                    idx = (idx+1) % self.MAX_SIZE
                self.ht[prevIdx].chain = idx
                self.ht[idx] = Entry(key, val)
                print("*Word inserted successfully*\n")
                self.SIZE += 1

    def insertWithReplacement(self, key, val):
        if self.SIZE == self.MAX_SIZE:  # Avoid Overflow
            print("*Dictionary is Full*\n")
            return None

        idx = self.hashFun(key)
        if self.ht[idx].word == " ":
            self.ht[idx] = Entry(key, val)
            print("*Inserted Successfully*\n")
            self.SIZE += 1
        else:
            if(self.hashFun(self.ht[idx].word) == idx):
                while self.ht[idx].chain != -1:
                    if self.ht[idx].word == key:
                        print("*Word Already Exist in Dictionary*\n")
                        return None
                    idx = self.ht[idx].chain
                if self.ht[idx].word == key:
                    print("*Word Already Exist in Dictionary*\n")
                    return None
                prevIdx = idx
                while self.ht[idx].word != " ":
                    idx = (idx+1) % self.MAX_SIZE
                self.ht[prevIdx].chain = idx
                self.ht[idx] = Entry(key, val)
                print("*Word inserted successfully*\n")
                self.SIZE += 1
            else:
                init = idx
                idx = (idx+1) % self.MAX_SIZE
                while(self.ht[idx].chain != init):
                    idx = (idx+1) % self.MAX_SIZE
                prevIdx = idx
                idx = init
                while(self.ht[idx].word != " "):
                    idx = (idx+1) % self.MAX_SIZE
                self.ht[prevIdx].chain = idx
                self.ht[idx] = Entry(
                    self.ht[init].word, self.ht[init].meaning, self.ht[init].c
hain)
                self.ht[init] = Entry(key, val)
                print("*Word inserted successfully*\n")
                self.SIZE += 1
```

```
    def search(self, key):
        idx = self.hashFun(key)
        compa = 0
        if self.hashFun(self.ht[idx].word) == idx:
            while self.ht[idx].chain != -1:
                compa += 1
                if self.ht[idx].word == key:
                    print("Word found after ", compa-1, "collisions")
                    print("Word: ", self.ht[idx].word,
                          "\nMeaning: ", self.ht[idx].meaning)
                    return None
                idx = self.ht[idx].chain
            if self.ht[idx].word == key:
                compa += 1
                print("Word Found After ", compa-1, "collisions")
                print("Word: ", self.ht[idx].word,
                      "\nMeaning: ", self.ht[idx].meaning)
            else:
                print("*Word does not exist in dictionary*\n")
        else:
            init = idx
            while self.hashFun(self.ht[idx].word) != init:
                compa += 1
                idx = (idx+1) % self.MAX_SIZE
                if idx == init:
                    print("Word does not exist in dictionary")
                    return None
            while self.ht[idx].chain != -1:
                compa += 1
                if self.ht[idx].word == key:
                    print("Word Found After ", compa-1, "collisions")
                    print("Word: ", self.ht[idx].word,
                          "Meaning: ", self.ht[idx].meaning)
                    return None
                idx = self.ht[idx].chain
            if self.ht[idx].word == key:
                compa += 1
                print("Word found after ", compa-1, "collisions")
                print("Word: ", self.ht[idx].word,
                      "\nMeaning: ", self.ht[idx].meaning)
            else:
                print("*Word does not exist in dictionary*\n")

    def searchIndex(self, key):
        idx = self.hashFun(key)
        if self.hashFun(self.ht[idx].word) == idx:
            while self.ht[idx].chain != -1:
                if self.ht[idx].word == key:
                    return idx
                idx = self.ht[idx].chain
            if self.ht[idx].word == key:
```

```
                return idx
            else:
                return -1
        else:
            init = idx
            while self.hashFun(self.ht[idx].word) != init:
                idx = (idx+1) % self.MAX_SIZE
                if idx == init:
                    return -1
            while self.ht[idx].chain != -1:
                if self.ht[idx].word == key:
                    return idx
                idx = self.ht[idx].chain
            if self.ht[idx].word == key:
                return idx
            else:
                return -1

    def delete(self, key):
        if(self.searchIndex(key) == -1):
            print("*Word does not exist in dictionary*\n")
            return None

        self.SIZE -= 1
        i = self.hashFun(key)
        if(key == self.ht[i].word):
            if(self.ht[i].chain != -1):
                init = self.ht[i].chain
                self.ht[i] = Entry(self.ht[init].word,
                                   self.ht[init].meaning, self.ht[init].chain)
                self.ht[init] = Entry()
            else:
                self.ht[i] = Entry()
                init = i
                i = (i+1) % self.MAX_SIZE
                while(i != init):
                    if(self.hashFun(self.ht[i].word) == self.hashFun(key)):
                        while(self.ht[i].chain != init):
                            i = self.ht[i].chain
                        self.ht[i].chain = -1
                        print(f"{key} word deleted successfully")
                        return None
                    i = (i+1) % self.MAX_SIZE
        else:
            i = self.searchIndex(key)
            if(self.ht[i].chain != -1):
                init = self.ht[i].chain
                self.ht[i] = Entry(self.ht[init].word,
                                   self.ht[init].meaning, self.ht[init].chain)
                self.ht[init] = Entry()
            else:
```

```
                self.ht[i] = Entry()
                init = i
                i = (i+1) % self.MAX_SIZE
                while(i != init):
                    if(self.hashFun(self.ht[i].word) == self.hashFun(key)):
                        while(self.ht[i].chain != init):
                            i = self.ht[i].chain
                        self.ht[i].chain = -1
                        print(f"{key} word deleted successfully.\n")
                        return None
                    i = (i+1) % self.MAX_SIZE
        print(f"{key} word deleted successfully.\n")

    def display(self):
        print("Bucket.No", "\t", "Word    ", "\t", "meaning   ", "\t", "chain"
)
        for i in range(0, self.MAX_SIZE):
            print(i, " " * 5, "\t", self.ht[i].word, " " * (8 - len(self.ht[i]
.word)),
                  "\t", self.ht[i].meaning, " " * (10 - len(self.ht[i].word)),
 "\t", self.ht[i].chain)


def main():
    ht = HashTable(7)

    while True:
        print("\n\nEnter\n\t1 to Insert \n\t2 to Search \n\t3 to Display \n\t4
 to Delete \n\t0 to Exit")
        choice = int(input(": "))

        if (choice == 0):
            break

        if choice == 1:
            while True:
                print(
                    "Choose:\n\t1 to Insert Without Replacement \n\t2 to Inser
t With Replacement \n\t0 to Exit")
                choice2 = int(input(": "))
                if (choice2 == 0):
                    break

                key = input("Enter the Word: ")
                val = input("Enter meaning: ")

                if(choice2 == 1):
                    ht.insertWithoutReplacement(key, val)
                elif (choice2 == 2):
                    ht.insertWithReplacement(key, val)
                else:
```

```
                    print("INVALID CHOICE. Try Again.")

        elif choice == 2:
            word = input("Enter Word: ")
            ht.search(word)

        elif choice == 3:
            ht.display()

        elif choice == 4:
            key = input("Enter Word: ")
            ht.delete(key)

        else:
            print("INVALID CHOICE. Try Again.")


main()
```

Testcase1:

Enter

1 to Insert

2 to Search

3 to Display

4 to Delete

0 to Exit

: 1

Choose:

1 to Insert Without Replacement

2 to Insert With Replacement

0 to Exit

: 1

Enter the Word: Mango

Enter meaning: Yellow

Inserted Successfully

Choose:

1 to Insert Without Replacement

2 to Insert With Replacement

0 to Exit

: 1

Enter the Word: Orange

Enter meaning: Orange

Inserted Successfully

Choose:

1 to Insert Without Replacement

2 to Insert With Replacement

0 to Exit

: 1

Enter the Word: Banana

Enter meaning: Yellow

Word inserted successfully

Choose:

1 to Insert Without Replacement

2 to Insert With Replacement

0 to Exit

: 1

Enter the Word: Strawberry

Enter meaning: Red

Inserted Successfully

Choose:

1 to Insert Without Replacement

2 to Insert With Replacement

0 to Exit

: 1

Enter the Word: Apple

Enter meaning: Red

Word inserted successfully

Choose:

1 to Insert Without Replacement

2 to Insert With Replacement

0 to Exit

Enter

1 to Insert

2 to Search

3 to Display

4 to Delete

0 to Exit

: 2

Enter Word: Apple

Word Found After 1 collisions

Word: Apple

Meaning: Red

Enter

1 to Insert

2 to Search

3 to Display

4 to Delete

0 to Exit

: 2

Enter Word: Mango

Word found after 0 collisions

Word: Mango

Meaning: Yellow

Enter

1 to Insert

2 to Search

3 to Display

4 to Delete

0 to Exit

: 0

Testcase2:

Enter

1 to Insert

2 to Search

3 to Display

4 to Delete

0 to Exit

: 1

Choose:

1 to Insert Without Replacement

2 to Insert With Replacement

0 to Exit

: 2

Enter the Word: Mango

Enter meaning: Yellow

Inserted Successfully

Choose:

1 to Insert Without Replacement

2 to Insert With Replacement

0 to Exit

: 2

Enter the Word: Orange

Enter meaning: Orange

Inserted Successfully

Choose:

1 to Insert Without Replacement

2 to Insert With Replacement

0 to Exit

: 2

Enter the Word: Banana

Enter meaning: Yellow

Word inserted successfully

Choose:

1 to Insert Without Replacement

2 to Insert With Replacement

0 to Exit

: 2

Enter the Word: Strawberry

Enter meaning: Red

Word inserted successfully

Choose:

1 to Insert Without Replacement

2 to Insert With Replacement

0 to Exit

: 2

Enter the Word: Apple

Enter meaning: Red

Word inserted successfully

Choose:

1 to Insert Without Replacement

2 to Insert With Replacement

0 to Exit

: 0

Enter

1 to Insert

2 to Search

3 to Display

4 to Delete

0 to Exit

: 2

Enter Word: Apple

Word Found After 1 collisions

Word: Apple

Meaning: Red

Enter

1 to Insert

2 to Search

3 to Display

4 to Delete

0 to Exit

: 2

Enter Word: Mango

Word found after 0 collisions

Word: Mango

Meaning: Yellow

Enter

1 to Insert

2 to Search

3 to Display

4 to Delete

0 to Exit

: 4

Enter Word: Apple

Apple word deleted successfully.

Enter

1 to Insert

2 to Search

3 to Display

4 to Delete

0 to Exit

: 0