# Author : Shubham (Roll No.: 21118)

# DSA Assignment 04 : Hashing Techniques (Linear Probing without replacement)

class Record():

    def \_\_init\_\_(self, num, name):

        self.ph\_num, self.name = num, name

    def \_\_str\_\_(self):

        return "Phone Num: {}, Name: {}".format(self.ph\_num, self.name)

class HashTable():

    def \_\_init\_\_(self, D):

        self.MAX\_SIZE = D

        self.SIZE = 0

        self.table = [Record(-1, "DUMMY") for \_ in range(D)]

        self.is\_del = [0 for \_ in range(D)]

    def Print(self):

        i = 0

        for entry in self.table:

            print(i, entry)

            i += 1

    def getHashVal(self, key):

        return key % self.MAX\_SIZE

    # Search for key in HashTable, if found return comparisons needed else return -1

    def search(self, key):

        if self.SIZE == 0:

            return -1

        idx = self.getHashVal(key)

        init = idx

        comp = 0

        while True:

            comp += 1

            if (self.table[idx].ph\_num == key):  # key found

                print(self.table[idx])

                return comp

            if (self.table[idx].ph\_num == -1 and not self.is\_del[idx]):  # empty slot

                return -1

            idx = (idx+1) % self.MAX\_SIZE

            if (idx == init):  # if hashtable is full

                return -1

    # Successful insertion returns True

    def insert(self, key, val):

        if self.SIZE == self.MAX\_SIZE:

            print("Hashtable is full.")

            return 0

        if (self.search(key) != -1):

            print("Record is already Present.")

            return 0

        idx = self.getHashVal(key)

        # linear probing

        while self.table[idx].ph\_num != -1:

            idx = (idx+1) % self.MAX\_SIZE

        self.table[idx] = Record(key, val)

        self.SIZE += 1

        return 1

    # Successful deletion returns True

    def delete(self, key):

        if (self.search(key) == -1):

            print("Record is not Present.")

            return 0

        idx = self.getHashVal(key)

        while True:

            if (self.table[idx].ph\_num == key):

                break

            idx = (idx+1) % self.MAX\_SIZE

        self.is\_del[idx] = 1

        self.table[idx] = Record(-1, "DUMMY")

        self.SIZE -= 1

        return 1

def main():

    ht = HashTable(10)

    while True:

        print('''Choose:

        \t1 for insertion

        \t2 for searching

        \t3 for deletion.

        \t0 to Exit.''')

        choice = int(input("Enter Choice: "))

        if not choice in [0, 1, 2, 3]:

            print("INVALID CHOICE.\n")

        elif choice == 0:

            break

        else:

            print("\n\nEnter Details")

            key = int(input("Enter Phone Number: "))

            if choice == 1:

                val = input("Enter Name: ")

            if choice == 1:

                if ht.insert(key, val):

                    print("INSERTION SUCCESSFUL.")

                else:

                    print("INSERTION FAILED.")

            elif choice == 2:

                comp = ht.search(key)

                if (comp == -1):

                    print("NOT FOUND.")

                else:

                    print ("FOUND. ({} comparisons needed)".format(comp))

            elif choice == 3:

                if ht.delete(key):

                    print("DELETION SUCCESSFUL.")

                else:

                    print("DELETION FAILED.")

        print()

main()

TESTCASES:

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 1

Enter Details

Enter Phone Number: 22

Enter Name: A

INSERTION SUCCESSFUL.

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 1

Enter Details

Enter Phone Number: 33

Enter Name: B

INSERTION SUCCESSFUL.

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 1

Enter Details

Enter Phone Number: 42

Enter Name: C

INSERTION SUCCESSFUL.

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 1

Enter Details

Enter Phone Number: 44

Enter Name: D

INSERTION SUCCESSFUL.

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 2

Enter Details

Enter Phone Number: 44

Phone Num: 44, Name: D

FOUND. (2 comparisons needed)

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 3

Enter Details

Enter Phone Number: 42

Phone Num: 42, Name: C

DELETION SUCCESSFUL.

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 2

Enter Details

Enter Phone Number: 42

NOT FOUND.

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 0

# Author : Shubham (Roll No.: 21118)

# DSA Assignment 04 : Hashing Techniques (Linear Probing with replacement)

class Record():

    def \_\_init\_\_(self, num, name):

        self.ph\_num, self.name = num, name

    def \_\_str\_\_(self):

        return "Phone Num: {}, Name: {}".format(self.ph\_num, self.name)

class HashTable():

    def \_\_init\_\_(self, D):

        self.MAX\_SIZE = D

        self.SIZE = 0

        self.table = [Record(-1, "DUMMY") for \_ in range(D)]

        self.is\_del = [0 for \_ in range(D)]

    def Print(self):

        i = 0

        for entry in self.table:

            print(i, entry)

            i += 1

    def getHashVal(self, key):

        return key % self.MAX\_SIZE

    # Search for key in HashTable, if found return comparisons needed else return -1

    def search(self, key):

        if self.SIZE == 0:

            return -1

        idx = self.getHashVal(key)

        init = idx

        comp = 0

        while True:

            comp += 1

            if (self.table[idx].ph\_num == key):  # key found

                print(self.table[idx])

                return comp

            if (self.table[idx].ph\_num == -1 and not self.is\_del[idx]):  # empty slot

                return -1

            idx = (idx+1) % self.MAX\_SIZE

            if (idx == init):  # if hashtable is full

                return -1

    # Successful insertion returns True

    def insert(self, key, val):

        if self.SIZE == self.MAX\_SIZE:

            print("Hashtable is full.")

            return 0

        if (self.search(key) != -1):

            print("Record is already Present.")

            return 0

        idx = self.getHashVal(key)

        # empty slot

        if self.table[idx].ph\_num == -1:

            self.table[idx] = Record(key, val)

            self.SIZE += 1

        # slot occupied by element of same chain

        elif idx == self.getHashVal(self.table[idx].ph\_num):

            while self.table[idx].ph\_num != -1:

                idx = (idx+1) % self.MAX\_SIZE

            self.table[idx] = Record(key, val)

            self.SIZE += 1

        # slot occupied by element of other chain -> case of replacement

        else:

            tmp = self.table[idx]

            self.table[idx] = Record(key, val)

            self.SIZE += 1

            while self.table[idx].ph\_num != -1:

                idx = (idx+1) % self.MAX\_SIZE

            self.table[idx] = tmp

        return 1

    # Successful deletion returns True

    def delete(self, key):

        if (self.search(key) == -1):

            print("Record is not Present.")

            return 0

        idx = self.getHashVal(key)

        while True:

            if (self.table[idx].ph\_num == key):

                break

            idx = (idx+1) % self.MAX\_SIZE

        self.is\_del[idx] = 1

        self.table[idx] = Record(-1, "DUMMY")

        self.SIZE -= 1

        return 1

def main():

    ht = HashTable(10)

    while True:

        print('''Choose:

        \t1 for insertion

        \t2 for searching

        \t3 for deletion.

        \t0 to Exit.''')

        choice = int(input("Enter Choice: "))

        if not choice in [0, 1, 2, 3]:

            print("INVALID CHOICE.\n")

        elif choice == 0:

            break

        else:

            print("\n\nEnter Details")

            key = int(input("Enter Phone Number: "))

            if choice == 1:

                val = input("Enter Name: ")

            if choice == 1:

                if ht.insert(key, val):

                    print("INSERTION SUCCESSFUL.")

                else:

                    print("INSERTION FAILED.")

            elif choice == 2:

                comp = ht.search(key)

                if (comp == -1):

                    print("NOT FOUND.")

                else:

                    print ("FOUND. ({} comparisons needed)".format(comp))

            elif choice == 3:

                if ht.delete(key):

                    print("DELETION SUCCESSFUL.")

                else:

                    print("DELETION FAILED.")

        print()

main()

TESTCASES:

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 1

Enter Details

Enter Phone Number: 22

Enter Name: A

INSERTION SUCCESSFUL.

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 1

Enter Details

Enter Phone Number: 33

Enter Name: B

INSERTION SUCCESSFUL.

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 1

Enter Details

Enter Phone Number: 42

Enter Name: C

INSERTION SUCCESSFUL.

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 1

Enter Details

Enter Phone Number: 44

Enter Name: D

INSERTION SUCCESSFUL.

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 2

Enter Details

Enter Phone Number: 44

Phone Num: 44, Name: D

FOUND. (1 comparisons needed)

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 3

Enter Details

Enter Phone Number: 44

Phone Num: 44, Name: D

DELETION SUCCESSFUL.

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 2

Enter Details

Enter Phone Number: 44

NOT FOUND.

Choose:

1 for insertion

2 for searching

3 for deletion.

0 to Exit.

Enter Choice: 0