Rajalakshmi Engineering College

Name: Devyesh Chellappan
Email: 241801049@rajalakshmi.edu.in
Roll no: 241801049
Phone: 7708811709
Branch: REC
Department: I AI & DS FB
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Ravi is building a basic hash table to manage student roll numbers for quick lookup. He decides to use Linear Probing to handle collisions.

Implement a hash table using linear probing where:

The hash function is: index = roll_number % table_sizeOn collision, check subsequent indexes (i+1, i+2, ...) until an empty slot is found.

You need to:

Insert a list of n student roll numbers into the hash table.Print the final state of the hash table. If a slot is empty, print -1.

Input Format

The first line of the input contains two integers n and table_size, where n is the

241801049

number of roll numbers to be inserted, and table_size is the size of the hash table.

The second line contains n space-separated integers — the roll numbers to insert into the hash table.

Output Format

The output should print a single line with table_size space-separated integers representing the final state of the hash table after all insertions.

If any slot remains unoccupied, it should be represented as -1.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4 7
50 700 76 85

Output: 700 50 85 -1 -1 -1 76

Answer

```
#include <stdio.h>

#define MAX 100

void initializeTable(int table[], int size) {
    for (int i = 0; i < size; i++)
        table[i] = -1;
}

int linearProbe(int table[], int size, int num) {
    int index = num % size;
    while (table[index] != -1) {
        index = (index + 1) % size;
    }
    return index;
}

void insertIntoHashTable(int table[], int size, int arr[], int n) {
    for (int i = 0; i < n; i++) {
```

241801049

```
        int index = linearProbe(table, size, arr[i]);
        table[index] = arr[i];
    }
}

void printTable(int table[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", table[i]);
    }
}

int main() {
    int n, table_size;
    scanf("%d %d", &n, &table_size);

    int arr[MAX];
    int table[MAX];

    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    initializeTable(table, table_size);
    insertIntoHashTable(table, table_size, arr, n);
    printTable(table, table_size);

    return 0;
}
```

Status : Correct ***Marks : 10/10***

241801049

Rajalakshmi Engineering College

Name: Devyesh Chellappan
Email: 241801049@rajalakshmi.edu.in
Roll no: 241801049
Phone: 7708811709
Branch: REC
Department: I AI & DS FB
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Priya is developing a simple student management system. She wants to store roll numbers in a hash table using Linear Probing, and later search for specific roll numbers to check if they exist.

Implement a hash table using linear probing with the following operations:

Insert all roll numbers into the hash table.For a list of query roll numbers, print "Value x: Found" or "Value x: Not Found" depending on whether it exists in the table.

Input Format

The first line contains two integers, n and table_size — the number of roll numbers to insert and the size of the hash table.

241801049

The second line contains n space-separated integers — the roll numbers to insert.

The third line contains an integer q — the number of queries.

The fourth line contains q space-separated integers — the roll numbers to search for.

Output Format

The output print q lines — for each query value x, print: "Value x: Found" or "Value x: Not Found"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5 10
21 31 41 51 61
3
31 60 51

Output: Value 31: Found
Value 60: Not Found
Value 51: Found

Answer

```
#include <stdio.h>

#define MAX 100



void initializeTable(int table[], int size) {
   for (int i = 0; i < size; i++)
      table[i] = -1;
}

int linearProbe(int table[], int size, int num) {
   int index = num % size;
   while (table[index] != -1) {
      index = (index + 1) % size;
```

241801049

```
    }
    return index;
}

void insertIntoHashTable(int table[], int size, int arr[], int n) {
    for (int i = 0; i < n; i++) {
        int index = linearProbe(table, size, arr[i]);
        table[index] = arr[i];
    }
}

int searchInHashTable(int table[], int size, int num) {
    int index = num % size;
    int start = index;

    while (table[index] != -1) {
        if (table[index] == num)
            return 1;
        index = (index + 1) % size;
        if (index == start)
            break;
    }
    return 0;
}

int main() {
    int n, table_size;
    scanf("%d %d", &n, &table_size);

    int arr[MAX], table[MAX];
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    initializeTable(table, table_size);
    insertIntoHashTable(table, table_size, arr, n);

    int q, x;
    scanf("%d", &q);
    for (int i = 0; i < q; i++) {
        scanf("%d", &x);
        if (searchInHashTable(table, table_size, x))
            printf("Value %d: Found\n", x);
        else
```

241801049

```
        printf("Value %d: Not Found\n", x);
    }

    return 0;
}
```

Status : Correct ***Marks : 10/10***

241801049

Rajalakshmi Engineering College

Name: Devyesh Chellappan
Email: 241801049@rajalakshmi.edu.in
Roll no: 241801049
Phone: 7708811709
Branch: REC
Department: I AI & DS FB
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

In a messaging application, users maintain a contact list with names and corresponding phone numbers. Develop a program to manage this contact list using a dictionary implemented with hashing.

The program allows users to add contacts, delete contacts, and check if a specific contact exists. Additionally, it provides an option to print the contact list in the order of insertion.

Input Format

The first line consists of an integer n, representing the number of contact pairs to be inserted.

Each of the next n lines consists of two strings separated by a space: the name of the contact (key) and the corresponding phone number (value).

241801049

The last line contains a string k, representing the contact to be checked or removed.

Output Format

If the given contact exists in the dictionary:

1. The first line prints "The given key is removed!" after removing it.
2. The next n - 1 lines print the updated contact list in the format: "Key: X; Value: Y" where X represents the contact's name and Y represents the phone number.

If the given contact does not exist in the dictionary:

1. The first line prints "The given key is not found!".
2. The next n lines print the original contact list in the format: "Key: X; Value: Y" where X represents the contact's name and Y represents the phone number.

Refer to the sample outputs for the formatting specifications.

Sample Test Case

```
Input: 3
Alice 1234567890
Bob 9876543210
Charlie 4567890123
Bob
Output: The given key is removed!
Key: Alice; Value: 1234567890
Key: Charlie; Value: 4567890123
```

Answer

```
// You are using GCC
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define TABLE_SIZE 101  // Size of the hash table (a prime number to reduce
collisions)
```

241801049

```
#define MAX_CONTACTS 50
#define MAX_LEN 20      // Max length for names and numbers

typedef struct {
   char name[MAX_LEN];
   char phone[MAX_LEN];
   bool occupied;
} Contact;

// Hash function: simple polynomial rolling hash
int hash(char *str) {
   int hash = 0;
   for (int i = 0; str[i]; i++) {
      hash = (hash * 31 + str[i]) % TABLE_SIZE;
   }
   return hash;
}

// Insert into hash table using linear probing
void insert(Contact table[], char *name, char *phone) {
   int index = hash(name);
   while (table[index].occupied) {
      index = (index + 1) % TABLE_SIZE;
   }
   strcpy(table[index].name, name);
   strcpy(table[index].phone, phone);
   table[index].occupied = true;
}

// Search for a contact by name
int search(Contact table[], char *name) {
   int index = hash(name);
   int start = index;
   while (table[index].occupied) {
      if (strcmp(table[index].name, name) == 0) {
         return index;
      }
      index = (index + 1) % TABLE_SIZE;
      if (index == start) break; // Avoid infinite loop
   }
   return -1;
}
```

241801049

```
// Main function
int main() {
   int n;
   scanf("%d", &n);

   Contact table[TABLE_SIZE] = {0};
   char insertionOrder[MAX_CONTACTS][MAX_LEN]; // For printing in insertion
order
   int count = 0;

   // Read and insert contacts
   for (int i = 0; i < n; i++) {
      char name[MAX_LEN], phone[MAX_LEN];
      scanf("%s %s", name, phone);
      insert(table, name, phone);
      strcpy(insertionOrder[count++], name);
   }

   // Read key to remove
   char key[MAX_LEN];
   scanf("%s", key);

   int index = search(table, key);
   if (index != -1) {
      printf("The given key is removed!\n");
      table[index].occupied = false;
   } else {
      printf("The given key is not found!\n");
   }

   // Print contacts in insertion order
   for (int i = 0; i < count; i++) {
      int idx = search(table, insertionOrder[i]);
      if (idx != -1) {
         printf("Key: %s; Value: %s\n", table[idx].name, table[idx].phone);
      }
   }

   return 0;
}
```

241801049

Status : Correct *Marks : 10/10*

241801049

Rajalakshmi Engineering College

Name: Devyesh Chellappan
Email: 241801049@rajalakshmi.edu.in
Roll no: 241801049
Phone: 7708811709
Branch: REC
Department: I AI & DS FB
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

You are provided with a collection of numbers, each represented by an array of integers. However, there's a unique scenario: within this array, one element occurs an odd number of times, while all other elements occur an even number of times. Your objective is to identify and return the element that occurs an odd number of times in this arrangement.

Utilize mid-square hashing by squaring elements and extracting middle digits for hash codes. Implement a hash table for efficient integer occurrence tracking.

Note: Hash function: squared = key * key.

Example

241801049

Input:

7

2 2 3 3 4 4 5

Output:

5

Explanation

The hash function and the calculated hash indices for each element are as follows:

2 -> hash(2*2) % 100 = 4

3 -> hash(3*3) % 100 = 9

4 -> hash(4*4) % 100 = 16

5 -> hash(5*5) % 100 = 25

The hash table records the occurrence of each element's hash index:

Index 4: 2 occurrences

Index 9: 2 occurrences

Index 16: 2 occurrences

Index 25: 1 occurrence

Among the elements, the integer 5 occurs an odd number of times (1 occurrence) and satisfies the condition of the problem. Therefore, the program outputs 5.

Input Format

The first line of input consists of an integer N, representing the size of the array.

The second line consists of N space-separated integers, representing the elements of the array.

Output Format

The output prints a single integer representing the element that occurs an odd

241801049

number of times.

If no such element exists, print -1.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 7
2 2 3 3 4 4 5

Output: 5

Answer

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define MAX_SIZE 100


unsigned int hash(int key, int tableSize) {
   int squared = key * key;
   int numDigits = 0;
   int temp = squared;
   while (temp > 0) {
      temp /= 10;
      numDigits++;
   }
   int middleDigits = (numDigits / 2);
   int divisor = 1;
   for (int i = 0; i < middleDigits; i++) {
      divisor *= 10;
   }
   return (squared / divisor) % tableSize;
}

int getOddOccurrence(int arr[], int size) {
   int hashTable[MAX_SIZE] = {0};
```

241801049

```
    for (int i = 0; i < size; i++) {
        int index = hash(arr[i], MAX_SIZE);
        hashTable[index]++;
    }
    for (int i = 0; i < size; i++) {
        int index = hash(arr[i], MAX_SIZE);
        if (hashTable[index] % 2 != 0) {
            return arr[i];
        }
    }
    return -1;
}

int main() {
    int n;
    scanf("%d", &n);

    int arr[MAX_SIZE];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    printf("%d\n", getOddOccurrence(arr, n));

    return 0;
}
```

Status : Correct *Marks : 10/10*

241801049 241801049 241801049 241801049