Rajalakshmi Engineering College

Name: Navaneetha Krishnan
Email: 241901067@rajalakshmi.edu.in
Roll no: 241901067
Phone: 8939010233
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

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

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

// You are using GCC
void initializeTable(int table[], int size) {
    //Type your code here
      for (int i = 0; i < size; i++) {
        table[i] = -1;  // Using -1 to indicate an empty slot
    }

}

int linearProbe(int table[], int size, int num) {
    //Type your code here
     int index = num % size;     // Initial hash index
    int start_index = index;    // To check if we've come full circle

    // Continue until an empty slot is found
```

```
    while (table[index] != -1) {
        index = (index + 1) % size;
        // If we've checked all slots, return -1 (table full)
        if (index == start_index) {
            return -1;
        }
    }
    return index;

}

void insertIntoHashTable(int table[], int size, int arr[], int n) {
    //Type your code here
    for (int i = 0; i < n; i++) {
        int index = linearProbe(table, size, arr[i]);
        if (index != -1) {
            table[index] = arr[i];
        } else {
            // Optional: Print an error message if the table is full
            printf("Error: Hash table is full. Cannot insert %d.\n", arr[i]);
        }
    }

}

void printTable(int table[], int size) {
    //Type your code here
    for (int i = 0; i < size; i++) {
        if (table[i] != -1)
            printf("%d ", table[i]);
        else
            printf("%d ",table[i]);
    }
    printf("\n");

}

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**

Rajalakshmi Engineering College

Name: Navaneetha Krishnan
Email: 241901067@rajalakshmi.edu.in
Roll no: 241901067
Phone: 8939010233
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

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.

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

// You are using GCC
void initializeTable(int table[], int size) {
    //Type your code here
    for (int i = 0; i < size; i++) {
        table[i] = -1;
    }

}

int linearProbe(int table[], int size, int num) {
    //Type your code here
```

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

}

void insertIntoHashTable(int table[], int size, int arr[], int n) {
    //Type your code here
     for (int i = 0; i < n; i++) {
        int index = linearProbe(table, size, arr[i]);
        if (index != -1)
            table[index] = arr[i];
        else
            printf("Error: Hash table is full. Cannot insert %d.\n", arr[i]);
    }

}

int searchInHashTable(int table[], int size, int num) {
    //Type your code here
    int index = num % size;
    int start_index = index;
    while (table[index] != -1) {
        if (table[index] == num) {
            return 1;
        }
        index = (index + 1) % size;
        if (index == start_index)
            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
            printf("Value %d: Not Found\n", x);
    }

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Navaneetha Krishnan
Email: 241901067@rajalakshmi.edu.in
Roll no: 241901067
Phone: 8939010233
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_COD_Question 3

Attempt : 2
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).

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 <stdlib.h>
#include <string.h>

typedef struct {
```

```
    char key[50];
    char value[50];
} KeyValuePair;

typedef struct {
    KeyValuePair *pairs;
    int size;
    int capacity;
} Dictionary;

void initDictionary(Dictionary *dict) {
    dict->size = 0;
    dict->capacity = 10;
    dict->pairs = (KeyValuePair *)malloc(dict->capacity * sizeof(KeyValuePair));
    if (dict->pairs == NULL) {
        perror("Failed to allocate memory for dictionary pairs");
        exit(EXIT_FAILURE);
    }
}

void insertKeyValuePair(Dictionary *dict, const char *key, const char *value) {
    for (int i = 0; i < dict->size; i++) {
        if (strcmp(dict->pairs[i].key, key) == 0) {
            strcpy(dict->pairs[i].value, value);
            return;
        }
    }

    if (dict->size == dict->capacity) {
        dict->capacity *= 2;
        dict->pairs = (KeyValuePair *)realloc(dict->pairs, dict->capacity *
sizeof(KeyValuePair));
        if (dict->pairs == NULL) {
            perror("Failed to reallocate memory for dictionary pairs");
            exit(EXIT_FAILURE);
        }
    }

    strcpy(dict->pairs[dict->size].key, key);
    strcpy(dict->pairs[dict->size].value, value);
    dict->size++;
}
```

```
void removeKeyValuePair(Dictionary *dict, const char *key) {
    int found_index = -1;
    for (int i = 0; i < dict->size; i++) {
        if (strcmp(dict->pairs[i].key, key) == 0) {
            found_index = i;
            break;
        }
    }

    if (found_index != -1) {
        for (int i = found_index; i < dict->size - 1; i++) {
            dict->pairs[i] = dict->pairs[i + 1];
        }
        dict->size--;
    }
}

int doesKeyExist(Dictionary *dict, const char *key) {
    for (int i = 0; i < dict->size; i++) {
        if (strcmp(dict->pairs[i].key, key) == 0) {
            return 1;
        }
    }
    return 0;
}

void printDictionary(Dictionary *dict) {
    if (dict->size == 0) {
        return;
    }
    for (int i = 0; i < dict->size; i++) {
        printf("Key: %s; Value: %s\n", dict->pairs[i].key, dict->pairs[i].value);
    }
}

int main() {
    Dictionary dict;
    initDictionary(&dict);

    int numPairs;
    scanf("%d", &numPairs);
```

```
    char key[50], value[50];
    for (int i = 0; i < numPairs; i++) {
        scanf("%s %s", key, value);
        insertKeyValuePair(&dict, key, value);
    }

    scanf("%s", key);

    if (doesKeyExist(&dict, key)) {
        printf("The given key is removed!\n");
        removeKeyValuePair(&dict, key);
        printDictionary(&dict);
    } else {
        printf("The given key is not found!\n");
        printDictionary(&dict);
    }

    free(dict.pairs);
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Navaneetha Krishnan
Email: 241901067@rajalakshmi.edu.in
Roll no: 241901067
Phone: 8939010233
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Develop a program using hashing to manage a fruit contest where each fruit is assigned a unique name and a corresponding score. The program should allow the organizer to input the number of fruits and their names with scores.

Then, it should enable them to check if a specific fruit, identified by its name, is part of the contest. If the fruit is registered, the program should display its score; otherwise, it should indicate that it is not included in the contest.

Input Format

The first line consists of an integer N, representing the number of fruits in the contest.

The following N lines contain a string K and an integer V, separated by a space, representing the name and score of each fruit in the contest.

The last line consists of a string T, representing the name of the fruit to search for.

Output Format

If T exists in the dictionary, print "Key "T" exists in the dictionary.".

If T does not exist in the dictionary, print "Key "T" does not exist in the dictionary.".

Refer to the sample outputs for the formatting specifications.

Sample Test Case

Input: 2
banana 2
apple 1
Banana

Output: Key "Banana" does not exist in the dictionary.

Answer

```
// You are using GCC
int keyExists(KeyValuePair* dictionary, int size, const char* key) {
    //Type your code here
for (int i = 0; i < size; i++) {
        if (strcmp(dictionary[i].key, key) == 0) {
            return 1; // Key found
        }
    }
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Navaneetha Krishnan
Email: 241901067@rajalakshmi.edu.in
Roll no: 241901067
Phone: 8939010233
Branch: REC
Department: I CSE (CS) FB
Batch: 2028
Degree: B.E - CSE (CS)

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

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

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

// You are using GCC

unsigned int hash(int key, int tableSize) {
    //Type your code here
    return(unsigned int)(((key %tableSize)+tableSize)%tableSize);
}

int getOddOccurrence(int arr[], int size) {
 // We'll build a small open-addressing table of capacity MAX_SIZE
    int   keys[MAX_SIZE];
    int   counts[MAX_SIZE];
    bool  used[MAX_SIZE];

    // Initialize
    for (int i = 0; i < MAX_SIZE; i++) {
        used[i]   = false;
        counts[i] = 0;
    }

    // Insert/count each array element
```

```
    for (int i = 0; i < size; i++) {
        int k = arr[i];
        unsigned int idx = hash(k, MAX_SIZE);

        // Linear probe until we find either an empty slot or the same key
        while (used[idx] && keys[idx] != k) {
            idx = (idx + 1) % MAX_SIZE;
        }

        if (!used[idx]) {
            // New key
            used[idx]   = true;
            keys[idx]   = k;
            counts[idx] = 1;
        } else {
            // Existing key: increment its count
            counts[idx]++;
        }
    }

    // Scan for the key with an odd count
    for (int i = 0; i < MAX_SIZE; i++) {
        if (used[i] && (counts[i] % 2 != 0)) {
            return keys[i];
        }
    }

    // None found
    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***