Rajalakshmi Engineering College

Name: SARAVANA PERUMAL B
Email: 241901103@rajalakshmi.edu.in
Roll no: 241901103
Phone: 9342922599
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_MCQ_Updated

Attempt : 1
Total Mark : 20
Marks Obtained : 18

Section 1 : MCQ

1. What is the worst-case time complexity for inserting an element in a hash table with linear probing?

Answer

O(n)

Status : Correct ***Marks : 1/1***

2. Which data structure is primarily used in linear probing?

Answer

Array

Status : Correct ***Marks : 1/1***

3. In division method, if key = 125 and m = 13, what is the hash index?

Answer

8

Status : Correct ***Marks : 1/1***

4. In the folding method, what is the primary reason for reversing alternate parts before addition?

Answer

To reduce the chance of collisions caused by similar digit patterns

Status : Correct ***Marks : 1/1***

5. In the division method of hashing, the hash function is typically written as:

Answer

h(k) = k % m

Status : Correct ***Marks : 1/1***

6. What does a deleted slot in linear probing typically contain?

Answer

A special "deleted" marker

Status : Correct ***Marks : 1/1***

7. What happens if we do not use modular arithmetic in linear probing?

Answer

Segmentation fault

Status : Wrong ***Marks : 0/1***

8. Which of the following values of 'm' is recommended for the division method in hashing?

Answer

A prime number

Status : Correct ***Marks : 1/1***

9. Which of these hashing methods may result in more uniform distribution with small keys?

Answer

Mid-Square

Status : Correct ***Marks : 1/1***

10. Which folding method divides the key into equal parts, reverses some of them, and then adds all parts?

Answer

Folding reversal method

Status : Correct ***Marks : 1/1***

11. What is the primary disadvantage of linear probing?

Answer

Clustering

Status : Correct ***Marks : 1/1***

12. Which of the following best describes linear probing in hashing?

Answer

Resolving collisions by linearly searching for the next free slot

Status : Correct ***Marks : 1/1***

13. What would be the result of folding 123456 into three parts and summing: (12 + 34 + 56)?

Answer

102

Status : Correct ***Marks : 1/1***

14. In C, how do you calculate the mid-square hash index for a key k, assuming we extract two middle digits and the table size is 100?

Answer

((k * k) / 100) % 100

Status : Correct ***Marks : 1/1***

15. What is the output of the mid-square method for a key k = 123 if the hash table size is 10 and you extract the middle two digits of k * k?

Answer

1

Status : Correct ***Marks : 1/1***

16. Which of the following statements is TRUE regarding the folding method?

Answer

It divides the key into parts and adds them.

Status : Correct ***Marks : 1/1***

17. In linear probing, if a collision occurs at index i, what is the next index checked?

Answer

(i + 1) % table_size

Status : Correct ***Marks : 1/1***

18. Which situation causes clustering in linear probing?

Answer

Sequential key insertion

Status : Wrong ***Marks : 0/1***

19. What is the initial position for a key k in a linear probing hash table?

Answer

k % table_size

Status : Correct ***Marks : 1/1***

20. Which C statement is correct for finding the next index in linear probing?

Answer

index = (index + 1) % size;

Status : Correct ***Marks : 1/1***

Rajalakshmi Engineering College

Name: SARAVANA PERUMAL B
Email: 241901103@rajalakshmi.edu.in
Roll no: 241901103
Phone: 9342922599
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

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++)
{
        int pos = linearProbe(table, size, arr[i]);
        table[pos] = arr[i];
}
}
void printTable(int table[], int size)
{
    for (int i = 0; i < size; i++)
    {
        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: SARAVANA PERUMAL B
Email: 241901103@rajalakshmi.edu.in
Roll no: 241901103
Phone: 9342922599
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

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++) {
        int idx = linearProbe(table, size, arr[i]);
        table[idx] = 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; // Found
        }
        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
        printf("Value %d: Not Found\n", x);
}

return 0;
}
```

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

Rajalakshmi Engineering College

Name: SARAVANA PERUMAL B
Email: 241901103@rajalakshmi.edu.in
Roll no: 241901103
Phone: 9342922599
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 : 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).

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>

#define TABLE_SIZE 101  // Prime number bigger than max n (50)
#define MAX_STR_LEN 20
```

```
typedef struct {
    char key[MAX_STR_LEN];
    char value[MAX_STR_LEN];
    int occupied;   // 0 = empty, 1 = occupied, 2 = deleted
} Entry;

Entry hashTable[TABLE_SIZE];
char insertionOrder[50][MAX_STR_LEN];
int count = 0;

unsigned int hash(char *str) {
    unsigned int h = 0;
    for (int i = 0; str[i]; i++) {
        h = (h * 31 + str[i]) % TABLE_SIZE;
    }
    return h;
}

// Insert key-value pair; assumes no duplicate keys in input
void insert(char *key, char *value) {
    unsigned int idx = hash(key);
    while (hashTable[idx].occupied == 1) {
        idx = (idx + 1) % TABLE_SIZE;
    }
    strcpy(hashTable[idx].key, key);
    strcpy(hashTable[idx].value, value);
    hashTable[idx].occupied = 1;

    strcpy(insertionOrder[count++], key);
}

// Search key in hash table; returns index or -1 if not found
int search(char *key) {
    unsigned int idx = hash(key);
    int startIdx = idx;
    while (hashTable[idx].occupied != 0) {
        if (hashTable[idx].occupied == 1 && strcmp(hashTable[idx].key, key) == 0) {
            return idx;
        }
        idx = (idx + 1) % TABLE_SIZE;
        if (idx == startIdx) break;
```

```
    }
    return -1;
}

// Remove key by marking slot deleted and removing from insertion order
int removeKey(char *key) {
    int idx = search(key);
    if (idx == -1) return 0;

    hashTable[idx].occupied = 2;  // Mark deleted

    // Remove key from insertionOrder
    int foundAt = -1;
    for (int i = 0; i < count; i++) {
        if (strcmp(insertionOrder[i], key) == 0) {
            foundAt = i;
            break;
        }
    }
    if (foundAt != -1) {
        for (int i = foundAt; i < count - 1; i++) {
            strcpy(insertionOrder[i], insertionOrder[i + 1]);
        }
        count--;
    }
    return 1;
}

int main() {
    int n;
    scanf("%d", &n);
    // Clear hash table
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable[i].occupied = 0;
    }

    for (int i = 0; i < n; i++) {
        char key[MAX_STR_LEN], value[MAX_STR_LEN];
        scanf("%s %s", key, value);
        insert(key, value);
    }
```

```
    char k[MAX_STR_LEN];
    scanf("%s", k);

    if (removeKey(k)) {
        printf("The given key is removed!\n");
    } else {
        printf("The given key is not found!\n");
    }

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

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: SARAVANA PERUMAL B
Email: 241901103@rajalakshmi.edu.in
Roll no: 241901103
Phone: 9342922599
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
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define TABLE_SIZE 31  // A prime number greater than max N (15) for better hashing

// Structure to hold fruit data
typedef struct {
    char key[50];  // fruit name
    int value;     // score
    int occupied;  // flag to indicate if slot is occupied
} Entry;

Entry hashTable[TABLE_SIZE];
```

```
// Simple hash function for strings
unsigned int hash(char *str) {
    unsigned int hash = 0;
    for (int i = 0; str[i]; i++) {
        hash = (hash * 31) + str[i];
    }
    return hash % TABLE_SIZE;
}

// Insert key-value pair into the hash table using linear probing
void insert(char *key, int value) {
    unsigned int idx = hash(key);
    while (hashTable[idx].occupied) {
        // If same key, update the value
        if (strcmp(hashTable[idx].key, key) == 0) {
            hashTable[idx].value = value;
            return;
        }
        idx = (idx + 1) % TABLE_SIZE;
    }
    strcpy(hashTable[idx].key, key);
    hashTable[idx].value = value;
    hashTable[idx].occupied = 1;
}

// Search for a key and return pointer to the entry, or NULL if not found
Entry* search(char *key) {
    unsigned int idx = hash(key);
    int startIdx = idx;

    while (hashTable[idx].occupied) {
        if (strcmp(hashTable[idx].key, key) == 0) {
            return &hashTable[idx];
        }
        idx = (idx + 1) % TABLE_SIZE;
        if (idx == startIdx) break;  // back to start means not found
    }
    return NULL;
}

int main() {
```

```
    int N;
    scanf("%d", &N);
    // Clear the hash table initially
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable[i].occupied = 0;
    }

    for (int i = 0; i < N; i++) {
        char fruit[50];
        int score;
        scanf("%s %d", fruit, &score);
        insert(fruit, score);
    }

    char query[50];
    scanf("%s", query);

    Entry *res = search(query);
    if (res != NULL) {
        printf("Key \"%s\" exists in the dictionary.\n", query);
    } else {
        printf("Key \"%s\" does not exist in the dictionary.\n", query);
    }

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: SARAVANA PERUMAL B
Email: 241901103@rajalakshmi.edu.in
Roll no: 241901103
Phone: 9342922599
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
#include <stdio.h>
#include <stdlib.h>

#define HASH_SIZE 100

// Node for linked list to handle collisions in the hash table
typedef struct Node {
    int key;          // original element
    int count;        // occurrence count
    struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int key) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->key = key;
    newNode->count = 1;
    newNode->next = NULL;
    return newNode;
```

```
}

// Mid-square hash function: (key * key) % 100
int midSquareHash(int key) {
    return (key * key) % HASH_SIZE;
}

// Insert or update the count of key in hash table
void insert(Node* hashTable[], int key) {
    int index = midSquareHash(key);
    Node* head = hashTable[index];
    Node* curr = head;

    // Search if key already exists in chain
    while (curr != NULL) {
        if (curr->key == key) {
            curr->count++;  // update count
            return;
        }
        curr = curr->next;
    }

    // If not found, create new node and insert at head
    Node* newNode = createNode(key);
    newNode->next = head;
    hashTable[index] = newNode;
}

// Find and return the element with odd occurrences; else -1
int findOddOccurrence(Node* hashTable[]) {
    for (int i = 0; i < HASH_SIZE; i++) {
        Node* curr = hashTable[i];
        while (curr != NULL) {
            if (curr->count % 2 == 1) {
                return curr->key;
            }
            curr = curr->next;
        }
    }
    return -1;
}
```

```
// Free memory allocated for hash table
void freeHashTable(Node* hashTable[]) {
    for (int i = 0; i < HASH_SIZE; i++) {
        Node* curr = hashTable[i];
        while (curr != NULL) {
            Node* temp = curr;
            curr = curr->next;
            free(temp);
        }
    }
}

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

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

    // Initialize hash table
    Node* hashTable[HASH_SIZE] = { NULL };

    // Insert all elements into hash table
    for (int i = 0; i < N; i++) {
        insert(hashTable, arr[i]);
    }

    // Find the element with odd occurrences
    int result = findOddOccurrence(hashTable);

    printf("%d\n", result);

    // Free memory
    freeHashTable(hashTable);

    return 0;
}
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**