**Problem Statement**

Write a program to count distinct elements in an array within the window size of W, using Hashing technique.

**Note:** Use HashMap

**Input format :**

The first line of input consists of integer **n** denoting the size of the array.

The second line of input consists of **n** space-separated integers representing the array elements.

The third line of input consists of an integer **W**, representing the window size.

**Output format :**

The output displays the number of distinct elements in each window in the new line.

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

1 ≤ n ≤ 100

1 ≤ array elements ≤ 10

1 ≤ W ≤ 10

**Sample test cases :**

**Input 1 :**

7

1 2 1 3 4 2 3

4

**Output 1 :**

3

4

4

3

**Input 2 :**

7

1 2 1 3 4 2 3

2

**Output 2 :**

2

2

2

2

2

2

// You are using Java

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

class DistinctElementsInWindow {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        // Input the size of the array

        int n = sc.nextInt();

        // Input the array elements

        int[] array = new int[n];

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextInt();

        }

        // Input the window size

        int W = sc.nextInt();

        // Calculate distinct elements in each window

        countDistinctInWindow(array, W);

        sc.close();

    }

    public static void countDistinctInWindow(int[] array, int W) {

        // HashMap to store the frequency of elements in the current window

        HashMap<Integer, Integer> freqMap = new HashMap<>();

        // Initialize the frequency map for the first window

        for (int i = 0; i < W; i++) {

            freqMap.put(array[i], freqMap.getOrDefault(array[i], 0) + 1);

        }

        // Print the count of distinct elements for the first window

        System.out.println(freqMap.size());

        // Slide the window from start to end of the array

        for (int i = W; i < array.length; i++) {

            // Remove the element going out of the window

            int outElement = array[i - W];

            if (freqMap.get(outElement) == 1) {

                freqMap.remove(outElement);

            } else {

                freqMap.put(outElement, freqMap.get(outElement) - 1);

            }

            // Add the new element coming into the window

            int inElement = array[i];

            freqMap.put(inElement, freqMap.getOrDefault(inElement, 0) + 1);

            // Print the count of distinct elements for the current window

            System.out.println(freqMap.size());

        }

    }

}

**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 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.**

**Code constraints :**

In this scenario, the test cases fall under the following constraints:

1 ≤ n ≤ 10

The keys are case-sensitive.

3 ≤ Length of the input strings, k ≤ 100

**Sample test cases :**

**Input 1 :**

3

Alice 1234567890

Bob 9876543210

Charlie 4567890123

Bob

**Output 1 :**

The given key is removed!

Key: Alice; Value: 1234567890

Key: Charlie; Value: 4567890123

**Input 2 :**

2

Quinn 1112223333

Roger 2223334444

Sarah

**Output 2 :**

The given key is not found!

Key: Quinn; Value: 1112223333

Key: Roger; Value: 2223334444

// You are using Java

import java.util.LinkedHashMap;

import java.util.Map;

import java.util.Scanner;

class ContactManager {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        // Input number of contacts

        int n = sc.nextInt();

        sc.nextLine(); // Consume the newline character

        // Create a LinkedHashMap to maintain insertion order

        LinkedHashMap<String, String> contactList = new LinkedHashMap<>();

        // Input contacts

        for (int i = 0; i < n; i++) {

            String line = sc.nextLine();

            String[] parts = line.split(" ");

            String name = parts[0];

            String phoneNumber = parts[1];

            contactList.put(name, phoneNumber);

        }

        // Input the key to be checked or removed

        String keyToCheck = sc.nextLine();

        if (contactList.containsKey(keyToCheck)) {

            // Remove the contact

            contactList.remove(keyToCheck);

            System.out.println("The given key is removed!");

        } else {

            System.out.println("The given key is not found!");

        }

        // Print the contact list

        for (Map.Entry<String, String> entry : contactList.entrySet()) {

            System.out.println("Key: " + entry.getKey() + "; Value: " + entry.getValue());

        }

        sc.close();

    }

}

**Problem Statement**

You are given an array of integers and a target sum. Your task is to write a program that identifies and prints pairs of integers from the array that add up to the specified target sum.

To efficiently find pairs that sum up to the target sum, we will use a hash table to store the elements of the array. The **hash function** takes an integer key and a table size and returns the index in the hash table where the element should be stored.

**Note:** index = key % tableSize

**Example**

**Input:**

6

1 2 3 4 5 6

7

**Output:**

Pairs: 4 and 3

Pairs: 5 and 2

Pairs: 6 and 1

**Explanation:**

The pair (4, 3) is obtained by adding the elements 4 and 3 from the array, resulting in a sum of 7.

Similarly, the pairs (5, 2) and (6, 1) are formed using the same approach, resulting in sums of 7 as well.

**Input format :**

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

The second line of input consists of **n** space-separated integers, representing the elements of the array.

The third line of input consists of an integer **s**, representing the target sum.

**Output format :**

For each pair of integers that sum up to the given target sum, print the pairs in the format: "Pairs: A and B", where A and B are the two integers in the pair.

If no such pairs are found, print "No such pairs".

**Refer to the sample output for the exact format.**

**Code constraints :**

MAX\_SIZE is set to 100.

1 ≤ n ≤ 20

0 ≤ elements of the array ≤ 50

**Sample test cases :**

**Input 1 :**

6

1 2 3 4 5 6

7

**Output 1 :**

Pairs: 4 and 3

Pairs: 5 and 2

Pairs: 6 and 1

**Input 2 :**

6

11 12 13 14 11 6

7

**Output 2 :**

No such pairs

// You are using Java

import java.util.HashSet;

import java.util.Scanner;

import java.util.Set;

class PairSumFinder {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        // Input size of the array

        int n = sc.nextInt();

        int[] array = new int[n];

        // Input the elements of the array

        for (int i = 0; i < n; i++) {

            array[i] = sc.nextInt();

        }

        // Input the target sum

        int targetSum = sc.nextInt();

        sc.close();

        findPairsWithSum(array, targetSum);

    }

    public static void findPairsWithSum(int[] array, int targetSum) {

        // Use a HashSet to keep track of elements we have seen

        Set<Integer> seen = new HashSet<>();

        boolean foundPair = false;

        for (int num : array) {

            int complement = targetSum - num;

            if (seen.contains(complement)) {

                System.out.println("Pairs: " + num + " and " + complement);

                foundPair = true;

            }

            seen.add(num);

        }

        if (!foundPair) {

            System.out.println("No such pairs");

        }

    }

}

**Problem Statement**

You are assigned the task of developing a basic student database management system using a hash table with **linear probing** for collision resolution.

Your program should facilitate the insertion of student records and allow efficient retrieval of student names based on their unique IDs.

**Input format :**

The first line of input consists of an integer **N,** representing the number of student records to insert.

The following **N**lines consist of IDs and names of students, separated by space.

The last line consists of an integer, representing the ID of the student name to be retrieved.

**Output format :**

The output prints the student's name, corresponding to the provided ID.

If the student is not found, display "Student not found".

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

HASH\_SIZE = 100.

The maximum number of students in the database is 100.

Student names are strings of up to 50 characters.

**Sample test cases :**

**Input 1 :**

3

101 Alice

202 Bob

303 Charlie

101

**Output 1 :**

Student name: Alice

**Input 2 :**

2

123 John

456 Jane

789

**Output 2 :**

Student name: Student not found

import java.util.Scanner;

class StudentDatabase {

    // Define the size of the hash table

    private static final int HASH\_SIZE = 100;

    // Define the hash table to store student records

    private String[] names = new String[HASH\_SIZE];

    private int[] ids = new int[HASH\_SIZE];

    // Method to compute hash index based on the student ID

    private int hash(int id) {

        return id % HASH\_SIZE;

    }

    // Method to insert a student record into the hash table

    public void insert(int id, String name) {

        int index = hash(id);

        // Linear probing for collision resolution

        while (names[index] != null && ids[index] != id) {

            index = (index + 1) % HASH\_SIZE;

        }

        names[index] = name;

        ids[index] = id;

    }

    // Method to retrieve a student's name based on the ID

    public String getStudentName(int id) {

        int index = hash(id);

        // Linear probing to find the correct record

        while (names[index] != null) {

            if (ids[index] == id) {

                return names[index];

            }

            index = (index + 1) % HASH\_SIZE;

        }

        return null; // Student not found

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        StudentDatabase database = new StudentDatabase();

        // Read number of student records to insert

        int N = Integer.parseInt(scanner.nextLine());

        // Read student records

        for (int i = 0; i < N; i++) {

            String[] input = scanner.nextLine().split(" ");

            int id = Integer.parseInt(input[0]);

            String name = input[1];

            database.insert(id, name);

        }

        // Read the ID to be queried

        int queryId = Integer.parseInt(scanner.nextLine());

        // Retrieve and print the student's name for the given ID

        String studentName = database.getStudentName(queryId);

        if (studentName != null) {

            System.out.println("Student name: " + studentName);

        } else {

            System.out.println("Student name: Student not found");

        }

        scanner.close();

    }

}

**Problem Statement**

You are required to implement a contact management system using hashing techniques.

The system should allow users to store and retrieve contact information based on names. The implementation should include hash table collision handling using **chaining**.

Write a program that accepts contact information and retrieves the contact number based on the provided name.

**Input format :**

The first line of input consists of an integer **N**, representing the number of contacts to be added.

The following **N** lines consist of two space-separated strings each: the name (a non-empty string) and the phone number (a non-empty string), representing the name and phone number of a contact.

The last line consists of a string, representing the name for which the phone number needs to be retrieved.

**Output format :**

The output prints "Phone number for X: Y", where X is the name and Y is the phone number associated with the given name.

If the contact is not found, print "Phone number for X: Contact not found", where X is the name.

**Refer to the sample output for formatting specifications.**

**Code constraints :**

The hash table size is fixed to 100

1 ≤ N ≤ 10

1 ≤ phone number length ≤ 10

1 ≤ length of a contact ≤ 50

Names are strings consisting of alphanumeric characters, without spaces.

Phone numbers are strings of digits.

**Sample test cases :**

**Input 1 :**

3

Alice 9632587415

Bob 9876543210

Carol 7412589636

Alice

**Output 1 :**

Phone number for Alice: 9632587415

**Input 2 :**

2

John 8521479632

Mary 9876543212

Janice

**Output 2 :**

Phone number for Janice: Contact not found

// You are using Java

import java.util.LinkedList;

import java.util.Scanner;

class Node {

    String name;

    String phoneNumber;

    Node next;

    Node(String name, String phoneNumber) {

        this.name = name;

        this.phoneNumber = phoneNumber;

        this.next = null;

    }

}

class HashTable {

    private static final int SIZE = 100;

    private LinkedList<Node>[] table;

    @SuppressWarnings("unchecked")

    public HashTable() {

        table = new LinkedList[SIZE];

        for (int i = 0; i < SIZE; i++) {

            table[i] = new LinkedList<>();

        }

    }

    private int hashFunction(String name) {

        int hash = 0;

        for (char c : name.toCharArray()) {

            hash = (hash + c) % SIZE;

        }

        return hash;

    }

    public void insert(String name, String phoneNumber) {

        int index = hashFunction(name);

        LinkedList<Node> bucket = table[index];

        for (Node node : bucket) {

            if (node.name.equals(name)) {

                node.phoneNumber = phoneNumber;

                return;

            }

        }

        bucket.add(new Node(name, phoneNumber));

    }

    public String find(String name) {

        int index = hashFunction(name);

        LinkedList<Node> bucket = table[index];

        for (Node node : bucket) {

            if (node.name.equals(name)) {

                return node.phoneNumber;

            }

        }

        return null;

    }

}

class ContactManagementSystem {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int N = Integer.parseInt(scanner.nextLine());

        HashTable hashTable = new HashTable();

        for (int i = 0; i < N; i++) {

            String[] input = scanner.nextLine().split(" ");

            String name = input[0];

            String phoneNumber = input[1];

            hashTable.insert(name, phoneNumber);

        }

        String queryName = scanner.nextLine();

        String phoneNumber = hashTable.find(queryName);

        if (phoneNumber != null) {

            System.out.println("Phone number for " + queryName + ": " + phoneNumber);

        } else {

            System.out.println("Phone number for " + queryName + ": Contact not found");

        }

        scanner.close();

    }

}

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

class ContactManagementSystem {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        Map<String, String> contacts = new HashMap<>();

        int N = Integer.parseInt(scanner.nextLine());

        for (int i = 0; i < N; i++) {

            String[] input = scanner.nextLine().split(" ");

            String name = input[0];

            String phoneNumber = input[1];

            contacts.put(name, phoneNumber);

        }

        String queryName = scanner.nextLine();

        String phoneNumber = contacts.get(queryName);

        if (phoneNumber != null) {

            System.out.println("Phone number for " + queryName + ": " + phoneNumber);

        } else {

            System.out.println("Phone number for " + queryName + ": Contact not found");

        }

        scanner.close();

    }

}