**Coding Test**

**Duration: 30 mins**

Name: **Sudhist Kumar**

Position: **Backend Engineer**

Result: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
This test is designed to test your coding fluency and problem-solving skills. Solutions should focus on solving the given problem, producing clean, working code, and producing solutions which optimize time and space complexity. You can use any programming language and you will not be penalized for minor syntax errors.  
  
**Problem 1**: Given a string, write a function which returns a boolean value indicating if it is palindrome or not. A string is said to be a palindrome if the reverse of the string is the same as the string. For example, **malayalam** is a palindrome, **geek** is not.  
  
Language you are using: **Java**  
Runtime Complexity of your solution: **O(n)**  
Space Complexity of your solution: **O(1)**  
**Write your code below (continue on back if needed):**

import java.util.\*;

class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter a string: ");

String str = sc.nextLine();

if (isPalindrome(str)) {

System.out.println(str + " is a palindrome.");

} else {

System.out.println(str + " is not a palindrome.");

}

}

public static boolean isPalindrome(String str) {

int left = 0;

int right = str.length() - 1;

while (left < right) {

if (str.charAt(left) != str.charAt(right)) {

return false;

}

left++;

right--;

}

return true;

}

}

**Problem 2**: Write a function which takes in a 2D List/Array of transactions and returns a list of transaction IDs which are fraudulent. Any transaction greater than or equal to 10000 is considered fraudulent. Any transaction from the same credit card in a different city within 30 minutes is considered fraudulent.  
  
Input: A 2D List/Array of transactions with each transaction record having a transaction ID (integer), credit card ID (integer, transaction amount (double), city (string), and time in minutes (integer). You can assume all transactions happen on the same day.  
  
Example:  
Input: [ [1, 1000, 500.00, “Vadodara”, 0], [2, 1000, 500.00, “Mumbai”, 5], [3, 1001, 500.00, “Mumbai”, 10], [4, 1001, 10000.00, “Mumbai”, 10]]  
Output: [2, 4]  
  
Transactions 2 and 4 should be considered fraudulent. Transaction 2 occurred within 30 min of transaction 1 with the same credit card ID (1000) and in a different city. Transaction 4 has an amount is greater than or equal to 10000.  
  
Language you are using: **Java**

Runtime Complexity of your solution: **O(n)**

Space Complexity of your solution: **O(n)**  
**Write your code below (continue on back if needed):**

import java.util.\*;

class Transaction {

int id;

int cardId;

double amount;

String city;

int time;

public Transaction (int id, int cardId, double amount, String city, int time) {

this.id = id;

this.cardId = cardId;

this.amount = amount;

this.city = city;

this.time = time;

}

}

public class FraudulentTransactions {

public static List<Integer> findFraudulentTransactions(List<Transaction> transactions) {

List<Integer> fraudulentIds = new ArrayList<>();

Map<Integer, List<Transaction>> cardTransactions = new HashMap<>();

for (Transaction transaction : transactions) {

if (transaction.amount >= 10000) {

fraudulentIds.add(transaction.id);

}

if (!cardTransactions.containsKey(transaction.cardId)) {

cardTransactions.put(transaction.cardId, new ArrayList<>());

}

List<Transaction> previousTransactions = cardTransactions.get(transaction.cardId);

for (Transaction prevTransaction : previousTransactions) {

if (!prevTransaction.city.equals(transaction.city) && Math.abs(prevTransaction.time - transaction.time) <= 30) {

fraudulentIds.add(transaction.id);

break;

}

}

previousTransactions.add(transaction);

}

return fraudulentIds;

}

public static void main(String[] args) {

List<Transaction> transactions = new ArrayList<>();

transactions.add(new Transaction(1, 1000, 500.00, "Vadodara", 0));

transactions.add(new Transaction(2, 1000, 500.00, "Mumbai", 5));

transactions.add(new Transaction(3, 1001, 500.00, "Mumbai", 10));

transactions.add(new Transaction(4, 1001, 10000.00, "Mumbai", 10));

List<Integer> result = findFraudulentTransactions(transactions);

System.out.println(result);

}

}

**Problem 3**: A railway system is keeping track of customer travel times between different stations. They are using this data to calculate the average time it takes to travel from one station to another.

Implement a class called**UndergroundSystem**with the following functions:

**void checkIn(int id, string stationName, int t)**A customer with a card ID equal to id, checks in at the station stationName at time t. A customer can only be checked into one place at a time.

**void checkOut(int id, string stationName, int t)**A customer with a card ID equal to id, checks out from the station stationName at time t.

**double getAverageTime(string startStation, string endStation)**Returns the average time it takes to travel from startStation to endStation.

The average time is computed from all the previous traveling times from startStation to endStation that happened directly, meaning a check in at startStation followed by a check out from endStation. The time it takes to travel from startStation to endStation may be different from the time it takes to travel from endStation to startStation. There will be at least one customer that has traveled from startStation to endStation before getAverageTime is called. You may assume all calls to the checkIn and checkOut methods are consistent. If a customer checks in at time t1 then checks out at time t2, then t1 < t2. All events happen in chronological order.  
  
Language you are using: **Java**  
Runtime Complexity of your Solution: **O(1)**   
Space Complexity of your Solution: **O(n)**  
**Write your code below (continue on back if needed):**

import java.util.\*;

class UndergroundSystem {

// A map to store check-in data of each customer

private Map<Integer, CheckInData> checkIns;

// A map to store total travel time and trip counts between station pairs

private Map<String, TravelData> travelTimes;

public UndergroundSystem() {

checkIns = new HashMap<>();

travelTimes = new HashMap<>();

}

public void checkIn(int id, String stationName, int t) {

checkIns.put(id, new CheckInData(stationName, t));

}

public void checkOut(int id, String stationName, int t) {

CheckInData checkInData = checkIns.get(id);

if (checkInData != null) {

String routeKey = checkInData.stationName + "->" + stationName;

int travelTime = t - checkInData.time;

TravelData travelData = travelTimes.getOrDefault(routeKey, new TravelData());

travelData.totalTime += travelTime;

travelData.tripCount++;

travelTimes.put(routeKey, travelData);

// Remove the check-in data as the trip is completed

checkIns.remove(id);

}

}

public double getAverageTime(String startStation, String endStation) {

String routeKey = startStation + "->" + endStation;

TravelData travelData = travelTimes.get(routeKey);

if (travelData != null) {

return (double) travelData.totalTime / travelData.tripCount;

}

return 0.0;

}

// Helper class to store check-in data

private static class CheckInData {

String stationName;

int time;

CheckInData(String stationName, int time) {

this.stationName = stationName;

this.time = time;

}

}

// Helper class to store travel data between stations

private static class TravelData {

int totalTime;

int tripCount;

TravelData() {

this.totalTime = 0;

this.tripCount = 0;

}

}

public static void main(String[] args) {

UndergroundSystem undergroundSystem = new UndergroundSystem();

undergroundSystem.checkIn(45, "Mumbai", 3);

undergroundSystem.checkIn(32, "Bengaluru", 8);

undergroundSystem.checkIn(27, "Mumbai", 10);

undergroundSystem.checkOut(45, "Delhi", 15);

undergroundSystem.checkOut(27, "Delhi", 20);

System.out.println(undergroundSystem.getAverageTime("Mumbai", "Delhi"));

undergroundSystem.checkIn(10, "Mumbai", 24);

System.out.println(undergroundSystem.getAverageTime("Mumbai", "Delhi"));

undergroundSystem.checkOut(10, "Delhi", 38);

System.out.println(undergroundSystem.getAverageTime("Mumbai", "Delhi"));

}

}