**1. Java: Movie Library**

Question description

A movie library application is being developed. This app has methods to maintain movie listings, with

functionality to add movies, remove movies, and search for movies.

Create a new class called Film and implement the IFilm interface.

Inside the Film class, define the following properties:

title (String): the title of the film

director (String): the director of the film

year (int): the year the film was released

Create another class called FilmLibrary and implement the IFilmLibrary interface.

Inside the FilmLibrary class, declare a private field called films of type List<IFilm> to store the films.

Add the following methods to the FilmLibrary class:

addFilm(IFilm film): adds a film to the film library. It takes an IFilm object as a parameter and

adds it to the films list

removeFilm(String title): removes a film from the film library based on its title if it is in the films

getFilms(): returns a list of all films in the film library

searchFilms(String query): searches for films in the film library based on a query string. It returns

a list of films whose title or director contains the query

getTotalFilmCount(): returns the total number of films in the film library

Example

There are 2 Film objects, with Title, Director, and Year.

HarryPotter DavidYates 2007

TheLordOfTheRings PeterJackson 2001

Add them to the list and determine the word to search from the movie list.

DavidYates

Finally, select the movie to be removed from the movie list.

TheLordOfTheRings

Output:

Total Film Count: 2

Search Results for DavidYates:

HarryPotter (DavidYates, 2007)

Removed Film: TheLordOfTheRings (PeterJackson, 2001)

All Films:

HarryPotter (DavidYates, 2007)

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer n, the number of films.

Each of the next n lines contains the film information separated by space (Title Director Year.).

The next line contains a string, the Director to search.

The next line contains a string, the Title to delete.

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN Function

----- --------

4 number of films n = 4

Film-1 Director-2 2004 first film Title = 'Film-1', Director = 'Director-2', 'Year' = 2004

Film-2 Director-1 2018 second film...

Film-3 Director-1 2001

Film-4 Director-3 2017

Director-1 Title or Director to search for = 'Director-1'

Film-1 Title to delete = 'Film-1'

Sample Output

Total Film Count: 4

Search Results for Director-1:

Film-2 (Director-1, 2018)

Film-3 (Director-1, 2001)

Removed Film: Film-1 (Director-2, 2004)

All Films:

Film-2 (Director-1, 2018)

Film-3 (Director-1, 2001)

Film-4 (Director-3, 2017)

Explanation

There are 4 films to add. The code stub reads the data, makes the method calls, and generates results.

SAMPLE CASE 2

Sample Input For Custom Testing

4 Film-1

Director-4

2014

Film-2 Director-3 2016

Film-3 Director-4 2012

Film-4 Director-3 2003

Director-3

Film-4

Sample Output

Total Film Count: 4

Search Results for Director-3:

Film-2 (Director-3, 2016)

Film-4 (Director-3, 2003)

Removed Film: Film-4 (Director-3, 2003)

All Films:

Film-1 (Director-4, 2014)

Film-2 (Director-3, 2016)

Film-3 (Director-4, 2012)

Explanation

There are 4 films to add. The code stub reads the data, makes the method calls, and generates results.

: Java 17

import java.io.\*;

import java.util.\*;

interface IFilm {

void setTitle(String title);

String getTitle();

void setDirector(String director);

String getDirector();

void setYear(int year);

int getYear();

}

interface IFilmLibrary {

void addFilm(IFilm film);

void removeFilm(String title);

List<IFilm> getFilms();

List<IFilm> searchFilms(String query);

int getTotalFilmCount();

}

//Write The Code Here

class Film implements IFilm {

private String title;

private String director;

private int year;

@Override

public String toString(){

return title + "(" + director + "," + year + ")";

}

@Override

public void setTitle(String title){

this.title = title;

}

@Override

public void setDirector(String director) {

this.director = director;

}

@Override

public void setYear(int year) {

this.year = year;

}

@Override

public String getTitle() {

return title;

}

@Override

public String getDirector() {

return director;

}

@Override

public int getYear() {

return year;

}

}

class FilmLibrary implements IFilmLibrary {

private List<IFilm> films;

public FilmLibrary() {

this.films = new ArrayList<>();

}

@Override

public void addFilm(IFilm film) {

films.add(film);

}

@Override

public void removeFilm(String title) {

films.removeIf(film -> film.getTitle().equalsIgnoreCase(title));

}

@Override

public List<IFilm> searchFilms(String query) {

List<IFilm> result = new ArrayList<>();

for (IFilm film : films) {

if (film.getTitle().contains(query) || film.getDirector().contains(query)) {

result.add(film);

}

}

return result;

}

@Override

public List<IFilm> getFilms() {

return films;

}

@Override

public int getTotalFilmCount() {

return (int) films.stream().count();

}

}

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

PrintWriter out = new PrintWriter(System.out);

IFilmLibrary filmLibrary = new FilmLibrary();

List<IFilm> films = new ArrayList<>();

int fCount = Integer.parseInt(br.readLine().trim());

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

String[] a = br.readLine().trim().split(" ");

IFilm e = new Film();

e.setTitle(a[0]);

e.setDirector(a[1]);

e.setYear(Integer.parseInt(a[2]));

filmLibrary.addFilm(e);

films.add(e);

}

int totalFilmCount = filmLibrary.getTotalFilmCount();

out.println("Total Film Count: " + totalFilmCount);

String[] b = br.readLine().trim().split(" ");

String query = b[0];

List<IFilm> searchResults = filmLibrary.searchFilms(query);

out.println("Search Results for " + query + ":");

for (IFilm film : searchResults) {

out.println(film.getTitle() + " (" + film.getDirector() + ", " + film.getYear() + ")");

}

String[] c = br.readLine().trim().split(" ");

String title = c[0];

IFilm randomFilm = null;

for (IFilm film : films) {

if (film.getTitle().equals(title)) {

randomFilm = film;

break;

}

}

if (randomFilm != null) {

filmLibrary.removeFilm(randomFilm.getTitle());

out.println("Removed Film: " + randomFilm.getTitle() + " (" + randomFilm.getDirector() + ", " + randomFilm.getYear() + ")");

}

List<IFilm> allFilms = filmLibrary.getFilms();

out.println("All Films:");

for (IFilm film : allFilms) {

out.println(film.getTitle() + " (" + film.getDirector() + ", " + film.getYear() + ")");

}

out.flush();

out.close();

}

}

**Java: Calculator Classes**

Question description

Before computers were common, accountants used a mechanical adding machine for their calculations.

It did just that: it added signed numbers. If they wanted to multiply, they had to add that many times. For

example, to multiply 3 by 3, they pressed 3 <add> 3 <add> 3 <add> <total> and got 3 3 3 T9 on their

printout, resetting the sum to 0 at <total>. Luckily there was a subtotal so they could print out a value

and carry it forward.

Alex wants to build two calculators as described below:

The Adder calculator that returns the sum of two integers.

The Multiplier calculator that returns the product of two integers via addition.

In this challenge, help Alex build the calculators by writing the complete implementations of the

following two classes:

1. The Adder class should implement the method int add(int a, int b) to return the sum of two integers,

a and b. It should also print Adding integers: a b each time it is called.

2. The Multiplier class should implement the method int multiply(int a, int b, Calculator calculator) to

return the result of a x b by repeated addition using the Adderclass.

The locked stub code in the editor consists of the following:

An abstract class Calculator that contains an abstract method, int add(int a, int b).

A solution class that tests the implementation of the Adder and the Multiplier classes by

creating an object of the Adder class.

reading the inputs and passes them along with the Adder class object in the method int

multiply(int a, int b, Calculator calculator) of the Multiplier class.

Adds headers and footers

Testing Addition, Sum =

Testing Multiplication, Product =

Constraints

1 ≤ a, b ≤ 10

INPUT FORMAT FOR CUSTOM TESTING

The first line contains the first integer, a.

The next line contains the second integer, b.

SAMPLE CASE 0

Sample Input 0

3

74

Sample Output 0

Testing Addition

Adding integers: 7 4

Sum = 11

Testing Multiplication

Adding integers: 7 7

Adding integers: 14 7

Adding integers: 21 7

Product = 28

Explanation 0

Multiplying 7 by 4 is the same as adding 7 to itself 4 times.

7 + 7 = 14.

14 + 7 = 21.

21 + 7 = 28.

: Java 17

import java.util.Scanner;

**abstract class Calculator {**

**abstract int add(int a, int b);**

**}**

**/\***

**\* Write the implementations of Adder and Multiplier classes.**

**\*/**

**class Adder extends Calculator {**

**@Override**

**int add(int a, int b) {**

**System.out.println("Adding integers: " + a + " " + b);**

**return a + b;**

**}**

**}**

**class Multiplier {**

**int multiply(int a, int b, Calculator cal) {**

**int p = a;**

**for (int i = 0; i < b - 1; i++) {**

**p = cal.add(p, a);**

**}**

**return p;**

**}**

**}**

public class Solution {

private static final Scanner INPUT\_READER = new Scanner(System.in);

private static final Calculator CALCULATOR = new Adder();

private static void testAddition(int a, int b) {

System.out.println("Sum = " + CALCULATOR.add(a, b));

}

private static void testMultiplication(int a, int b) {

System.out.println("Product = " + new Multiplier().multiply(a, b, CALCULATOR));

}

public static void main(String[] args) {

int a = Integer.parseInt(INPUT\_READER.nextLine());

int b = Integer.parseInt(INPUT\_READER.nextLine());

System.out.println("Testing Addition");

testAddition(a, b);

System.out.println("\nTesting Multiplication");

testMultiplication(a, b);

}

}

1. **Preemptive Scheduling**

Coding Easy Real-World Stacks Interviewer Guidelines

Question description

When multiple tasks are executed on a single-threaded CPU, the tasks are scheduled based on the

principle of pre-emption. When a higher-priority task arrives in the execution queue, then the lowerpriority

task is pre-empted, i.e. its execution is paused until the higher-priority task is complete.

There are n functions to be executed on a single-threaded CPU, with each function having a unique ID

between 0 and n - 1. Given an integer n, representing the number of functions to be executed, and an

execution log as an array of strings, logs, of size m, determine the exclusive times of each of the

functions. Exclusive time is the sum of execution times for all calls to a function. Any string representing

an execution log is of the form {function\_id}:{"start"|"end"}: {timestamp}, indicating that the function

with ID = function\_id, either starts or ends at a time identified by the timestamp value.

Note: While calculating the execution time of a function call, both the starting and ending times of the

function call have to be included. The log of the form {function\_id}:{start}: {timestamp} means that the

running function is preempted at the beginning of timestamp second. The log of the form {function\_id}:

{end}: {timestamp} means that the function function\_id is preempted after completing its execution at

timestamp second i.e after timestamp second.

Example

Suppose n = 3, logs = ["0:start:0", "2:start:4", "2:end:5", "1:start:7", "1:end:10", "0:end:11"]

Timestamp Function Running Remarks

0 0 Function 0 starts

4 of 20

1 0

2 0

3 0

4 2 Function 2 starts and Function 0 is preempted

5 2 Function 2 ends

6 0 Function 0 resumes

7 1 Function 1 starts and Function 0 is preempted

8 1

9 1

10 1 Function 1 ends

11 0 Function 0 ends

Thus the total number of seconds allocated to functions 0, 1, and 2 are 6, 4, and 2 respectively. Hence

the answer is [6, 4, 2].

Function Description

Complete the function getTotalExecutionTime in the editor below.

getTotalExecutionTime has the following parameters:

int n: the number of functions to be executed

string logs[m]: the execution logs of the different calls to the functions

Returns

int[n]: the execution time of all functions with IDs from 0 to n - 1.

Constraints

1 ≤ n ≤ 100

1 ≤ m ≤ 500

0 ≤ function\_id < n

0 ≤ timestamp ≤ 3 \* 103

5 of 20

The timestamps are given in non-decreasing order.

No two starting timestamps and no two ending timestamps are equal

Every function "start" call has a corresponding function "end" call.

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, n.

The next line contains an integer, m, the size of logs[].

Each of the next m lines contains an integer, logs[i].

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

2 → n = 2

4 → logs[] size m = 4

0:start:0 → logs = ["0:start:0",

1:start:3 "1:start:3",

1:end:6 "1:end:6",

0:end:10 "0:end:10"]

Sample Output

74

Explanation

The execution of functions happens in the following order:

Time = [0, 3): Function ID = 0 executes

Time = [3, 6]: Function ID = 1 executes

Time = [6, 10]: Function ID = 0 executes

Thus the exclusive times for different functions are as follows:

Function (ID = 0) = 3 + 4 = 7

Function (ID = 1) = 4

SAMPLE CASE 1

Sample Input For Custom Testing

6 of 20

STDIN FUNCTION

----- --------

3 → n = 3

6 → logs[] size m = 6

0:start:0 → logs = ["0:start:0",

1:start:3 "1:start:3",

1:end:6 "1:end:6",

2:start:8 "2:start:8",

2:end:10 "2:end:10",

0:end:12 "0:end:12"]

Sample Output

643

Explanation

The execution of functions happens in the following order:

Time = [0, 3): Function ID = 0 executes

Time = [3, 6]: Function ID = 1 executes

Time = [7, 8): Function ID = 0 executes

Time = [8, 10]: Function ID = 2 executes

Time = [11, 12]: Function ID = 0 executes

Thus the exclusive times for different functions are as follows:

Function (ID = 0) = 3 + 1 + 2 = 6

Function (ID = 1) = 4

Function (ID = 2) = 3

So the answer to be returned is [6, 4, 3].

Interviewer guidelines

SOLUTION

Skills: Stacks

We maintain a stack of the functions currently preempted or running. Iterate through the logs,

whenever we encounter the start event of some function, we push it to the stack. If some function is

already present on the stack, it will be preempted, hence add its total execution time so far. Whenever

a function has ended pop it from the stack and add its total execution time.

COMPLEXITY ANALYSIS

Time Complexity - O(m)

We iterate through the logs taking a total of O(m) amount of time.

Space Complexity - O(n)

The answer array and stack take O(n) extra space in the worst case.

: Java 17

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

public static List<Integer> getTotalExecutionTime(int n, List<String> logs) {

int[] executionTime = new int[n];

Stack<int[]> stack = new Stack<>();

for (String log : logs) {

String[] parts = log.split(":");

int functionId = Integer.parseInt(parts[0]);

String type = parts[1];

int timestamp = Integer.parseInt(parts[2]);

if (type.equals("start")) {

stack.push(new int[]{functionId, timestamp});

} else {

int[] startLog = stack.pop();

int startTime = startLog[1];

int execTime = timestamp - startTime + 1;

executionTime[functionId] += execTime;

if (!stack.isEmpty()) {

executionTime[stack.peek()[0]] -= execTime;

}

}

}

s

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

for (int time : executionTime) {

result.add(time);

}s

return result;

}

}

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int n = Integer.parseInt(bufferedReader.readLine().trim());

int logsCount = Integer.parseInt(bufferedReader.readLine().trim());

List<String> logs = IntStream.range(0, logsCount).mapToObj(i -> {

try {

return bufferedReader.readLine();

} catch (IOException ex) {

throw new RuntimeException(ex);

}

}).collect(toList());

List<Integer> result = Result.getTotalExecutionTime(n, logs);

bufferedWriter.write(

result.stream()

.map(Object::toString)

.collect(joining("\n"))

+ "\n"

);

bufferedReader.close();

bufferedWriter.close();

}

}

**1. Faulty Server Detection**

Coding Easy Hashing Real-World

Question description

Implement a prototype service to automate the detection and replacement of faulty servers to improve

the availability of an application.

There are n servers with ids s1, s2, ..., sn, and an array of strings, logs, of size m. Log format is "

<server\_id> <success/error>", the id of the server, and the status of the processed request. If a particular

server id logs an error for three consecutive requests, it is considered faulty and is replaced with a new

server with the same id.

Given n and the array logs, find the number of times a faulty server was replaced.

Example

Suppose n = 2 and logs = ["s1 error", "s1 error", "s2 error", "s1 error", "s1 error", "s2 success"].

Log Server Id Status Last 3 status at server id Fault Detection

s1 error s1 error [error]

s1 error s1 error [error, error]

s2 error s2 error [error]

s1 error s1 error [error, error, error] Detected, s1 is replaced

s1 error s1 error [error]

4 of 14

s2 success s2 success []

s1 was replaced one time. So, output should be 1.

Function Description

Complete the function countFaults in the editor below.

countFaults has the following parameters:

int n: the number of servers

string logs[m]: the application logs

Returns

int: the number of times servers were replaced

Constraints

1 ≤ n ≤ 200

1 ≤ m ≤ 2 x 10

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, n.

The next line contains an integer, m.

Each of the next m lines contains a string, logs[i].

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

5 → n = 5

7 → m = 7

s1 error → logs = ["s1 error", "s2 error", "s1 error", "s4 success", "s5 error", "s3 success", "s1 erro

r"]

s2 error

s1 error

s4 success

s5 error

4

5 of 14

s3 success

s1 error

Sample Output

1

Explanation

Only s1 is replaced. It has three errors without any successes at the first, third, and last lines of the log.

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

3 → n = 3

6 → m = 6

s2 error → logs = ["s2 error", "s3 error", "s2 error", "s2 error", "s3 error", "s3 error"]

s3 error

s2 error

s2 error

s3 error

s3 error

Sample Output

2

Explanation

s2 and s3 are replaced at the fourth and last logs respectively.

Skills: Map, Strings

Iterate through the logs and maintain a map of the number of errors for each server id. Reset the

count of errors for a server whenever it shows success. Increment the answer and reset the count

when the number of consecutive errors reaches three for a server.

COMPLEXITY ANALYSIS

Time Complexity - O(m)

Assuming the length of the strings is small, we iterate over the logs once amounting to a total time

complexity of O(m).

Space Complexity - O(n)

The errors count map takes O(n) extra space in the worst case.

: Java 15

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'countFaults' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts following parameters:

\* 1. INTEGER n

\* 2. STRING\_ARRAY logs

\*/

public static int countFaults(int n, List<String> logs) {

**Map<String, Integer> errorCount = new HashMap<>();**

**int replacements = 0;**

**for (String log : logs) {**

**String[] parts = log.split(" ");**

**String server\_id = parts[0];**

**String status = parts[1];**

**if (status.equals("error")) {**

**errorCount.put(server\_id, errorCount.getOrDefault(server\_id, 0) + 1);**

**if (errorCount.get(server\_id) == 3) {**

**replacements++;**

**errorCount.put(server\_id, 0);**

**}**

**} else {**

**errorCount.put(server\_id, 0);**

**}**

**}**

**return replacements;**

**}**

}

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int n = Integer.parseInt(bufferedReader.readLine().trim());

int logsCount = Integer.parseInt(bufferedReader.readLine().trim());

List<String> logs = IntStream.range(0, logsCount).mapToObj(i -> {

try {

return bufferedReader.readLine();

} catch (IOException ex) {

throw new RuntimeException(ex);

}

}).collect(toList());

int result = Result.countFaults(n, logs);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

**Server Traffic Monitor**

Coding Easy Sorting Real-World Interviewer Guidelines

Question description

There is client-server architecture with n clients and one server. Each client starts its interaction with the

server at the second start[i] and stops at the second end[i]. The maximum traffic is defined as the

maximum number of concurrent interactions with the server.

Find the earliest time at which the maximum number of clients are interacting with the server.

Note: The endpoint is also included in the interaction.

Example

Suppose start = [1, 6, 2 ,9] and end = [8, 7, 6, 10]

Time Connection Active Clients

1 Client 1 Joined 1

2 Client 3 Joined 1, 3

3 1, 3

4 of 14

4 1, 3

5 1, 3

6 Client 2 Joined 1, 2, 3

7 Client 3 Left 1, 2

8 Client 2 Left 1

9 Client 1 Left and Client 4 Joined 4

10 4

11 Client 4 Left

The maximum number of concurrent interactions is 3 which happens first at the 6 second. Return 6.

Function Description

Complete the function getMaxTrafficTime in the editor below.

getMaxTrafficTime has the following parameters:

int start[n]: interaction start times

int end[n]: interaction end times

Returns

int: the earliest time of maximum concurrent interactions

Constraints

1 ≤ n ≤ 10

1 ≤ start[i] ≤ end[i] ≤ 10

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer n, the number of clients.

Each of the next n lines contains an integer start[i].

The next line contains the same integer n, the number of clients.

th

5

9

5 of 14

Each of the next n lines contains an integer end[i].

SAMPLE CASE 0

Sample Input 0

STDIN FUNCTION

----- --------

5 → start[] size n = 5

2 → start[] = [2, 3, 7, 4, 7]

37475

→ end[] size n = 5

4 → end[] = [4, 5, 8, 7, 10]

587 10

Sample Output 0

4

Explanation

At time t = 4, the first, second, and fourth clients are connected to the server. At time t = 7, the last

three clients are connected.

The earliest time when 3 are connected is t = 4.

SAMPLE CASE 1

Sample Input 1

STDIN FUNCTION

----- --------

3 → start[] size n = 3

1 → start[] = [1, 1, 1]

113

→ end[] size n = 3

5 → end[] = [5, 5, 5]

55

6 of 14

Sample Output 1

1

Explanation

All three clients are connected to the server at t = 1.

Interviewer guidelines

SOLUTION

Skills: Tuple, Sorting, Constructive Algorithms

The first thing to notice is that the maximum number of clients at the first point in time will always be

at a time when some client starts.

We can maintain an array of tuples of (start[i], 1) and (end[i], -1) in an array. Sorting this makes sure

that any client x that starts after some client y will appear after y in the array with +1 and similarly any

client x that ends after client y will appear after y with -1. We can iterate through the array and add

and subtract the second parameter. The maximum value of count can be maintained with the

corresponding time the client joined which should give us the answer.

7 of 14

COMPLEXITY ANALYSIS

Time Complexity - O(n logn)

We sort the array of tuples in O(n logn) time.

Space Complexity - O(n)

The array to store the tuples takes O(n) extra space in the worst case.

: Java 15

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'getMaxTrafficTime' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts following parameters:

\* 1. INTEGER\_ARRAY start

\* 2. INTEGER\_ARRAY end

\*/

public static int getMaxTrafficTime(List<Integer> start, List<Integer> end) {

**// Write your code here**

**TreeMap<Integer, Integer> t = new TreeMap<>();**

**for (int s : start) {**

**t.put(s, t.getOrDefault(s, 0) + 1);**

**}**

**for (int e : end) {**

**t.put(e + 1, t.getOrDefault(e + 1, 0) - 1);**

**}**

**int mc = 0;**

**int res = 0;**

**int curr = 0;**

**for (Map.Entry<Integer, Integer> e : t.entrySet()) {**

**curr += e.getValue();**

**if (curr > mc) {**

**mc = curr;**

**res = e.getKey();**

**}**

**}**

**return res;**

**}**

**}**

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int startCount = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> start = IntStream.range(0, startCount).mapToObj(i -> {

try {

return bufferedReader.readLine().replaceAll("\\s+$", "");

} catch (IOException ex) {

throw new RuntimeException(ex);

}

})

.map(String::trim)

.map(Integer::parseInt)

.collect(toList());

int endCount = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> end = IntStream.range(0, endCount).mapToObj(i -> {

try {

return bufferedReader.readLine().replaceAll("\\s+$", "");

} catch (IOException ex) {

throw new RuntimeException(ex);

}

})

.map(String::trim)

.map(Integer::parseInt)

.collect(toList());

int result = Result.getMaxTrafficTime(start, end);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

1**. Cache Queries**

Question description

Implement a prototype of a simple cache query handler.

There are n data entries stored in the cache. Each entry is of the form {timestamp, key, value}, where

timestamp represents the time at which the entry was stored in the cache, key represents the ID

assigned to the cache entry, and value represents the data value of the entry, an integer represented as

a string. The keys assigned to the cache entries may not be unique. The cache query handler receives q

query requests, where each query is of the form {key, timestamp}, where key represents the ID of the

cache entry to find, and timestamp represents the time the entry was added.

Given two 2D arrays of strings, cache\_entries, and queries, of sizes n x 3 and q x 2 respectively, return an

array of size q with the data values for each query.

Example

4 of 16

cache\_entries = [["12:30:22", "a2er5i80", "125"], ["09:07:47", "io09ju56", "341"], ["01:23:09", "a2er5i80",

"764"]]

queries = [["a2er5i80", "01:23:09"], ["io09ju56", "09:07:47"]]

queries[0] corresponds to the data entry at index 2, with value = "764"

queries[1] corresponds to the data entry at index 1, with value = "341"

Event Type Data Cache Query Result

Update cache ["09:07:47", "io09ju56", "341"] io09ju56 = 341 -

Query ["io09ju56", "09:07:47"] io09ju56 = 341 341

Update cache ["12:30:22", "a2er5i80", "125"] io09ju56 = 341, a2er5i80 = 125 -

Update Cache ["01:23:09", "a2er5i80", "764"] io09ju56 = 341, a2er5i80 = 764 -

Query ["a2er5i80", "01:23:09"] io09ju56 = 341, a2er5i80 = 764 764

Return [764, 341].

**Function Description**

Complete the function getQueryAnswers in the editor below.

getQueryAnswers has the following parameters:

string cache\_entries[n][3]: the cache data entries

string queries[q][2]: the queries

Returns

int [q]: the answers to the queries

Constraints

1 ≤ n ≤ 10

1 ≤ q ≤ 10

1 ≤ int(cache\_entries[i][2]) ≤ 10

cache\_entries[i][0] represents a valid timestamp in the format hh:mm:ss

size(cache\_entries[i][0]) = 8

5

5

8

5 of 16

cache\_entries[i][1], is an alphanumeric value, consisting of only lowercase English letters (a-z), and

digits (0-9)

It is guaranteed that the queried {key, timestamp} pair is present in the cache.

At a particular timestamp, there can be no duplicate keys.

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, n, the number of elements in cache\_entries.

The next line contains a constant integer, 3.

Each of the next n lines contains three space-separated strings, cache\_entries[i][0], cache\_entries[i][1],

cache\_entries[i][2].

The next line contains a constant integer, q, the number of elements in queries.

The next line contains a constant integer, 2.

Each of the next q lines contains two space-separated strings, query[i][0] and query[i][1].

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

3 → cache\_entries[] size n = 3

3 → constant integer = 3 (elements per cache\_entries[i])

03:00:23 1e3fg08w 101 → cache\_entries = [["03:00:23", "1e3fg08w", "101"],

12:45:36 ef41t56l 243 ["12:45:36", "ef41t56l", "243"],

21:09:31 ko0e51j8 809 ["21:09:31", "ko0e51j8", "809"]]

1 → queries[] size q = 1

2 → constant integer = 2 (elements per queries[i])

ef41t56l 12:45:36 → queries = [["ef41t56l", "12:45:36"]]

Sample Output

243

Explanation

The only query corresponds to the data entry at index 1, with value = 243.

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

6 of 16

4 → cache\_entries[] size n = 4

3 → constant integer = 3 (elements per cache\_entries[i])

01:34:05 166k1kph 352 → cache\_entries = [["01:34:05", "166k1kph", "352"],

56:38:37 8pvj20oo 107 ["56:38:37", "8pvj20oo", "107"],

36:17:33 r0v06eec 180 ["36:17:33", "r0v06eec", "180"],

20:34:20 e15y6dv4 490 ["20:34:20", "e15y6dv4", "490"]]

2 → queries[] size q = 2

2 → constant integer = 2 (elements per queries[i])

e15y6dv4 20:34:20 → queries = [["e15y6dv4", "20:34:20"],

8pvj20oo 56:38:37 ["8pvj20oo", "56:38:37"]]

Sample Output

490

107

Explanation

The queries correspond to data entries at indices 3 and 1 respectively, with values 490 and 107.

Interviewer guidelines

SOLUTION

Skills: Map

Concatenate the timestamp and key to store in a map and use it to answer the queries from the map.

sCOMPLEXITY ANALYSIS

Time Complexity - O(n + q)

7 of 16

Iterating through the data entries and queries takes O(n + q) time.

Space Complexity - O(n)

The map takes O(n) extra space.

: Java 8

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'getQueryAnswers' function below.

\*

\* The function is expected to return an INTEGER\_ARRAY.

\* The function accepts following parameters:

\* 1. 2D\_STRING\_ARRAY cache\_entries

\* 2. 2D\_STRING\_ARRAY queries

\*/

public static List<Integer> getQueryAnswers(List<List<String>> cache\_entries, **List<List<String>> queries) {**

**// Write your code here**

**List<Integer> ans = new ArrayList<>();**

**Map<String, Integer> map = new HashMap<>();**

**for (List<String> list : cache\_entries) {**

**String key = list.get(0) + "\_" + list.get(1);**

**map.put(key, Integer.valueOf(list.get(2)));**

**}**

**for (List<String> query : queries) {**

**String key = query.get(1) + "\_" + query.get(0);**

**ans.add(map.get(key));**

**}**

**return ans;**

**}**

}

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int cache\_entriesRows = Integer.parseInt(bufferedReader.readLine().trim());

int cache\_entriesColumns = Integer.parseInt(bufferedReader.readLine().trim());

List<List<String>> cache\_entries = new ArrayList<>();

IntStream.range(0, cache\_entriesRows).forEach(i -> {

try {

cache\_entries.add(

Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.collect(toList())

);

} catch (IOException ex) {

throw new RuntimeException(ex);

}

});

int queriesRows = Integer.parseInt(bufferedReader.readLine().trim());

int queriesColumns = Integer.parseInt(bufferedReader.readLine().trim());

List<List<String>> queries = new ArrayList<>();

IntStream.range(0, queriesRows).forEach(i -> {

try {

queries.add(

Stream.of(bufferedReader.readLine().replaceAll("\\s+$", "").split(" "))

.collect(toList())

);

} catch (IOException ex) {

throw new RuntimeException(ex);

}

});

List<Integer> result = Result.getQueryAnswers(cache\_entries, queries);

bufferedWriter.write(

result.stream()

.map(Object::toString)

.collect(joining("\n")) + "\n"

);

bufferedReader.close();

bufferedWriter.close();

}

}

**Car Inheritance**

Question description

Build on an abstract class and initialize an instance of each class with a variable. The program will then

test the implementation by retrieving the stored data.

11 of 16

The locked code in the editor does the following:

1. Declares an abstract class named Car with the implementations for getIsSedan() and getSeats()

methods and an abstract method named getMileage() .

2. Creates WagonR, HondaCity, or InnovaCrysta object based on input (0 for WagonR, 1 for HondaCity

and 2 for InnovaCrysta).

3. Calls the getIsSedan() , getSeats() , and getMileage() methods on the object.



The details for each car are provided below:

1. WagonR is not a sedan and has 4 seats.

2. HondaCity is a sedan and has 4 seats.

3. InnovaCrysta is not a sedan and has 6 seats.

12 of 16

Function Description

Complete the code in the editor below to implement the following:

1. Create classes named WagonR , HondaCity , and InnovaCrysta that all inherit from the Car class.

2. Each class must have a constructor that receives one integer argument representing the mileage of

the car.

3. Each class must implement a getMileage() method which returns a string in the form of ' <mileage>

kmpl' where <mileage> is the value provided to the constructor.

Constraints

0 ≤ type of car ≤ 2

5 ≤ mileage ≤ 30

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer that describes the type of car to instantiate.

The second line contains an integer, the mileage of the car.

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN Function

----- --------

0 → type of car to instantiate = 0 (WagonR)

22 → mileage = 22

Sample Output

A WagonR is not Sedan, is 4-seater, and has a mileage of around 22 kmpl.

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN Function

----- --------

1 → type of car to instantiate = 1 (HondaCity)

12 → mileage = 12

13 of 16

Sample Output

A HondaCity is Sedan, is 4-seater, and has a mileage of around 12 kmpl.

import java.io.\*;

abstract class Car {

protected boolean isSedan;

protected String seats;

public Car(boolean isSedan, String seats) {

this.isSedan = isSedan;

this.seats = seats;

}

public boolean getIsSedan() {

return this.isSedan;

}

public String getSeats() {

return this.seats;

}

abstract public String getMileage();

public void printCar(String name) {

System.out.println(

"A " + name + " is " + (this.getIsSedan() ? "" : "not ")

+ "Sedan, is " + this.getSeats() + "-seater, and has a mileage of around "

+ this.getMileage() + ".");

}

}

**// Write your code here.**

**/\*\***

**\* WagonR class**

**\*\*/**

**class WagonR extends Car {**

**int mileage;**

**WagonR(int mileage) {**

**super(false, "4");**

**this.mileage = mileage;**

**}**

**@Override**

**public String getMileage() {**

**return this.mileage + " kmpl";**

**}**

**}**

**/\*\***

**\* HondaCity class**

**\*\*/**

**class HondaCity extends Car {**

**int mileage;**

**HondaCity(int mileage) {**

**super(true, "4");**

**this.mileage = mileage;**

**}**

**@Override**

**public String getMileage() {**

**return mileage + " kmpl";**

**}**

**}**

**/\*\***

**\* InnovaCrysta class**

**\*\*/**

**class InnovaCrysta extends Car {**

**int mileage;**

**InnovaCrysta(int mileage) {**

**super(false, "6");**

**this.mileage = mileage;**

**}**

**@Override**

**public String getMileage() {**

**return mileage + " kmpl";**

**}**

**}**

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

int carType = Integer.parseInt(bufferedReader.readLine().trim());

int carMileage = Integer.parseInt(bufferedReader.readLine().trim());

if (carType == 0) {

Car wagonR = new WagonR(carMileage);

wagonR.printCar("WagonR");

}

if (carType == 1) {

Car hondaCity = new HondaCity(carMileage);

hondaCity.printCar("HondaCity");

}

if (carType == 2) {

Car innovaCrysta = new InnovaCrysta(carMileage);

innovaCrysta.printCar("InnovaCrysta");

}

}

}

**1. Similar Password**

Coding Easy Strings Greedy Algorithms Sorting

Question description

A password detection system for HackerRank accounts detects a password as similar if the number of

vowels is equal to the number of consonants in the password.

Passwords consist of lowercase English characters only, and vowels are ('a', 'e', 'i', 'o', 'u').

To check the strength of a password and how easily it can be hacked, some manipulations can be made to

the password. In one operation, any character of the string can either be incremented or decremented.

For example, 'f' can be incremented to 'g', or decremented to 'e'. Note that character 'a' cannot be

decremented and 'z' cannot be incremented.

Find the minimum number of operations in which the password can be made similar.

Example

Consider password = "hack". The 'h' can be changed to 'i' in one operation. The resultant string is "iack"

which has 2 vowels ('i', 'a') and 2 consonants ('c', 'k') and hence the string is similar. Return 1, the

minimum number of operations required.

Function Description

Complete the function countMinimumOperations in the editor below.

countMinimumOperations has the following parameter:

string password: the password

Rishi Khuntia

4 of 15

Returns

int: the minimum number of operations required to make the password similar

Constraints

2 ≤ |password| ≤ 3\*10

It is guaranteed that the length of the password is even.

The given string consists of lowercase Latin characters only.

INPUT FORMAT FOR CUSTOM TESTING

The first and only line contains the given string password.

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

abcd → password = "abcd"

Sample Output

1

Explanation

In one operation, 'd' can be changed to 'e'. The resultant string is "abce" which has an equal number of

vowels and consonants.

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

bigbangt → the given string, password = "bigbangt"

Sample Output

2

Explanation

5

In the first operation, the first occurrence of 'b' can be changed to 'a'. In the next operation, another

occurrence of 'b' can be changed to 'a'. The resultant string is "aigaangt" which has an equal number

of vowels and consonants.

Interviewer guidelines

SOLUTION

Skills: Sorting, Strings, Greedy Algorithms

Optimal Solution:

Consider the number of vowels to be v and the number of consonants to be c. Firstly, each vowel can

be changed to a consonant in a single operation. So, if v is greater than c, then the extra (v - c) / 2

vowels can be changed in one operation each. Hence, the answer in this case is (v - c) / 2.

Otherwise, for each consonant, find the minimum number of operations needed to convert it to a

vowel, and then choose the minimum (c - v) / 2 consonants, and sum their operations to obtain the

answer.

COMPLEXITY ANALYSIS

Time Complexity - O(n log n)

Traversing each character and finding the minimum operations for each character takes O(n) time.

There can be up to n consonants, so sorting the operations array takes O(n log n) time. Hence, time

complexity is O(n log n).

Space Complexity - O(n)

The operations array can have up to n entries, so space complexity is O(n).

: Java 8

import java.util.ArrayList;

import java.util.PriorityQueue;

class Result {

/\*

\* Complete the 'countMinimumOperations' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts STRING password as parameter.

\*/

public static int countMinimumOperations(String p) {

**int n = p.length();**

**int half = n / 2, v = 0, c = 0;**

**ArrayList<Character> arr = new ArrayList<Character>();**

**arr.add('a');**

**arr.add('e');**

**arr.add('i');**

**arr.add('o');**

**arr.add('u');**

**for (int i = 0; i < n; i++) {**

**char ch = p.charAt(i);**

**if (arr.contains(ch)) {**

**v++;**

**} else {**

**c++;**

**}**

**}**

**if (v > c) {**

**return v - half;**

**} else if (c > v) {**

**int diff = c - half;**

**PriorityQueue<Integer> pq = new PriorityQueue<Integer>();**

**for (int i = 0; i < n; i++) {**

**char ch = p.charAt(i);**

**int mini = Integer.MAX\_VALUE;**

**if (!arr.contains(ch)) {**

**mini = Math.min(mini, Math.abs(ch - 'a'));**

**mini = Math.min(mini, Math.abs(ch - 'e'));**

**mini = Math.min(mini, Math.abs(ch - 'i'));**

**mini = Math.min(mini, Math.abs(ch - 'o'));**

**mini = Math.min(mini, Math.abs(ch - 'u'));**

**}**

**pq.add(mini);**

**}**

**int ans = 0;**

**while (diff > 0) {**

**int top = pq.remove();**

**ans += top;**

**diff--;**

**}**

**return ans;**

**}**

**return 0;**

**}**

}

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

String password = bufferedReader.readLine();

int result = Result.countMinimumOperations(password);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

**Java: Grocery Receipt**

Coding Abstract Class Easy OOPS Java

Question description

A software development team for an e-commerce company is working on billing software. They are

given some product prices, followed by discounts and the number of items purchased by a user.

Given a list of unique items with their prices, a list of unique items with their discounts, and a list of

items purchased by the user, generate the final invoice. The format for each item is item, price,

total\_price (total\_price is calculated as sum\_of\_quantity \* price\_of\_item, and if a discount exists, it's

subtracted from it). There might be duplicates in the list of purchases.

This invoice list should be sorted in ascending order of product name.

GroceryReceipt class:

Create the 'GroceryReceipt' class that extends GroceryReceiptBase

These properties will be passed.

Prices - the list of items and their prices

Discounts - the list of discounts for each item

Example

It is given that Bananas, Apples, and Oranges are priced at 10, 20, and 5, respectively. The discount on an

Orange is 10%. The customer shopping list is as follows.

item quantity

Banana 5

Orange 2

Orange 1

The final invoice is:

item price total\_price

Banana 10 50

Orange 5 13.5

Normally, an Orange costs 5 per unit, but there is a 10% discount. For 3 units of Orange, the gross price

is 15, less 10% is 13.5.

Function Description

Create GroceryReceipt class by extending the GroceryReceiptBase abstract class. Implement a Calculate

function to create invoices. The function should return the item, quantity, and the total price after the

discount for every item in the grocery receipt.

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer n, the number of fruits.

Each of the next n lines contains two space-separated values, (fruit (String), price per unit (Double)).

The next line contains an integer m, the number of discount items.

Each of the next m lines contains two space-separated values, (fruit (String), percentage discount

(Integer)).

The next line contains an integer k, the number of items purchased.

Each of the next k lines contains two space-separated values, (fruit (String), units purchased (Integer)).

SAMPLE CASE 0

Sample Input

STDIN FUNCTION

----- --------

3 → n = 3

Apple 34 → [fruit,price per unit] = [["Apple", 34],

Banana 14 ["Banana", 14],

Orange 4 ["Orange", 4]]

1 → m = 1

Orange 10 → [fruit,percentage discount] = [["Orange", 10]]

2 → k = 2

Apple 2 → [fruit, units] = [["Apple", 2], ["Apple", 5]]

Apple 5

Sample Output

Apple 34.0 238.0

Explanation

The shopping list is 2 units Apple and 5 units Apple. 7 units of Apple cost 7 \* 34 = 238 and there is no

discount.

SAMPLE CASE 1

Sample Input

STDIN FUNCTION

----- --------

3 → n = 3

Apple 31 → [fruit,price per unit] = [["Apple", 31],

Banana 39 ["Banana", 39],

Orange 47 ["Orange", 47]]

3 → m = 3

Apple 40 → [fruit,percentage discount] = [["Apple", 40],

Banana 40 ["Banana", 40],

Orange 50 ["Orange", 50]]

2 → k = 2

Banana 4 → [fruit, units] = [["Banana", 4], ["Apple", 3]]

Apple 3

Sample Output

Apple 31.0 55.8

Banana 39.0 93.6

Explanation

3 units of Apples and 4 units of Bananas are purchased. There is a 40% discount on both items.

Apple: 3 \* 31 = 93, less 40% is 55.8

Banana: 4 \* 39 = 156, less 40% is 93.6

class Grocery {

String fruit;

double price, total;

Grocery(String fruit, double price, double total) {

this.fruit = fruit;

this.price = price;

this.total = total;

}

}

class Node {

String fruit;

int count;

Node(String fruit, int count) {

this.fruit = fruit;

this.count = count;

}

}

abstract class GroceryReceiptBase {

private Map<String, Double> prices;

private Map<String, Integer> discounts;

public GroceryReceiptBase(Map<String, Double> prices, Map<String, Integer> discounts) {

this.prices = prices;

this.discounts = discounts;

}

public abstract List<Grocery> Calculate(List<Node> shoppingList);

public Map<String, Double> getPrices() {

return prices;

}

public Map<String, Integer> getDiscounts() {

return discounts;

}

}

// Create the 'GroceryReceipt' class that extends GroceryReceiptBase above.

**class GroceryReceipt extends GroceryReceiptBase {**

**public GroceryReceipt(Map<String, Double> prices, Map<String, Integer> discounts) {**

**super(prices, discounts);**

**}**

**@Override**

**public List<Grocery> Calculate(List<Node> shoppingList) {**

**Map<String, Integer> list = new TreeMap<>();**

**for (Node s : shoppingList) {**

**String fruit = s.fruit;**

**int sum = s.count;**

**if (!list.containsKey(fruit)) {**

**list.put(fruit, sum);**

**} else {**

**list.put(fruit, list.get(fruit) + sum);**

**}**

**}**

**List<Grocery> res = new ArrayList<>();**

**for (Map.Entry<String, Integer> entry : list.entrySet()) {**

**String fruit = entry.getKey();**

**int sum = entry.getValue();**

**double price = getPrices().get(fruit);**

**double total = 0;**

**if (getDiscounts().containsKey(fruit)) {**

**total = sum \* price \* (1 - (double) getDiscounts().get(fruit) / 100);**

**} else {**

**total = sum \* price;**

**}**

**res.add(new Grocery(fruit, price, total));**

**}**

**return res;**

**}**

**}**

class Solution {

public static void main(String[] args) throws IOException {

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

PrintWriter writer = new PrintWriter(System.getenv("OUTPUT\_PATH"), "UTF-8");

List<Node> boughtItems = new ArrayList<>();

Map<String, Double> prices = new HashMap<>();

Map<String, Integer> discounts = new HashMap<>();

int n = Integer.parseInt(reader.readLine().trim());

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

String[] a = reader.readLine().trim().split(" ");

prices.put(a[0], Double.parseDouble(a[1]));

}

int m = Integer.parseInt(reader.readLine().trim());

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

String[] a = reader.readLine().trim().split(" ");

discounts.put(a[0], Integer.parseInt(a[1]));

}

int b = Integer.parseInt(reader.readLine().trim());

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

String[] a = reader.readLine().trim().split(" ");

boughtItems.add(new Node(a[0], Integer.parseInt(a[1])));

}

GroceryReceipt g = new GroceryReceipt(prices, discounts);

List<Grocery> result = g.Calculate(boughtItems);

for (Grocery x : result) {

writer.printf("%s %.1f %.1f\n", x.fruit, x.price, x.total);

}

writer.flush();

writer.close();

}

1. **Location Detection**

Coding Easy Strings Real-World

Question description

Implement a prototype service to detect a user's location based on their IP addresses.

The IP addresses belonging to the IPv4 space are conventionally represented by four octets, "a.b.c.d" -

such as 127.10.20.30. To keep it simple, these IP addresses are classified into 5 different regions indexed

from 1 to 5 on the basis of the order of the bits in the first octet.

Broadly, the IP Addresses are categorized as follows:

1. 0.0.0.0 - 127.255.255.255

2. 128.0.0.0 - 191.255.255.255

3. 192.0.0.0 - 223.255.255.255

4. 224.0.0.0 - 239.255.255.255

5. 240.0.0.0 - 255.255.255.255

Given a list of strings, ip\_addresses, of size n, representing possible IPv4 addresses, for each address,

determine if it is a valid IP or not, and classify it into one of the 5 classes. Return an array of n integers

that represent the index of the regions for the corresponding IP addresses. Represent an invalid IP as -1.

Example

Suppose ip\_addresses = ["128.12.34.0", "31.258.90.11"]

IP Region Remarks

128.12.34.0 2 Belongs to [128.0.0.0, 191.255.255.255]

31.258.90.11 -1 The second octet, 258, is invalid.

4 of 14

Hence the answer is [2, -1].

Function Description

Complete the function getRegions in the editor below.

getRegions has the following parameter:

string ip\_addresses[n]: the list of possible IPv4 addresses

Returns

int [n]: the regions of the IP addresses or -1 if the IP is invalid

Constraints

1 ≤ n ≤ 10

7 ≤ size(ip\_addresses) ≤ 15

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, n, the number of elements in ip\_addresses.

Each of the next n lines contains a string, ip\_addresses[i].

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

3 → ip\_addresses[] size n = 3

123.211.102.13 → ip\_addresses = ["123.211.102.13",

271.142.67.142 "271.142.67.142",

225.217.132.58 "225.217.132.58"]

Sample Output

1 -

1

4

Explanation

IP Region Remark

3

5 of 14

123.211.102.13 1 Since 0 < 123 < 127

271.142.67.142 -1 271 is invalid

225.217.132.58 4 224 < 225 < 240

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

4 → ip\_addresses[] size n = 4

15.231.268.11 → ip\_addresses = ["15.231.268.11",

234.11.89.45 "234.11.89.45",

249.255.12.91 "249.255.12.91",

178.102.163.34 "178.102.163.34"]

Sample Output

-1

452

Explanation

IP Region Remark

15.231.268.11 -1 268 is invalid

234.11.89.45 4 224 < 234 < 239

249.255.12.91 5 240 < 249 < 255

178.102.163.34 2 128 < 178 < 191

Interviewer guidelines

SOLUTION

Skills: Strings, Loops

6 of 14

Parse each IP string into four integer octets. To ensure that IP is valid, check if ther are 4 octets

between 0 and 255. Finally, assign the region to each octet based on their first octet.

COMPLEXITY ANALYSIS

Time Complexity - O(n)

The iteration on the IP addresses takes O(n) time.

Space Complexity - O(n)

The answer array takes O(n) extra space.

: Java 17

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'getRegions' function below.

\*

\* The function is expected to return an INTEGER\_ARRAY.

\* The function accepts STRING\_ARRAY ip\_addresses as parameter.

\*/

public static List<Integer> getRegions(List<String> ip\_addresses) {

// Write your code here

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

for (int i = 0; i < ip\_addresses.size(); i++) {

String[] arr = ip\_addresses.get(i).split("[.]+");

if (Integer.parseInt(arr[1]) <= 255 && Integer.parseInt(arr[2]) <= 255

&& Integer.parseInt(arr[3]) <= 255) {

if (Integer.parseInt(arr[0]) >= 0 && Integer.parseInt(arr[0]) < 128)

check.add(1);

else if (Integer.parseInt(arr[0]) >= 128 && Integer.parseInt(arr[0]) < 192)

check.add(2);

else if (Integer.parseInt(arr[0]) >= 192 && Integer.parseInt(arr[0]) < 224)

check.add(3);

else if (Integer.parseInt(arr[0]) >= 224 && Integer.parseInt(arr[0]) < 240)

check.add(4);

else if (Integer.parseInt(arr[0]) >= 240 && Integer.parseInt(arr[0]) < 256)

check.add(5);

else

check.add(-1);

} else

check.add(-1);

}

return check;

}

}

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int ip\_addressesCount = Integer.parseInt(bufferedReader.readLine().trim());

List<String> ip\_addresses = IntStream.range(0, ip\_addressesCount).mapToObj(i -> {

try {

return bufferedReader.readLine();

} catch (IOException ex) {

throw new RuntimeException(ex);

}

}).collect(toList());

List<Integer> result = Result.getRegions(ip\_addresses);

bufferedWriter.write(

result.stream()

.map(Object::toString)

.collect(joining("\n"))

+ "\n"

);

bufferedReader.close();

bufferedWriter.close();

}

}

**String Comparison**

Coding Strings Easy Implementation

Question description

Given a pair of strings. The goal is to make them equal by doing one of the following operations at most

once:

Select two characters, char1 and char2, and replace all occurrences of char1 in the first string with

char2.

Select two characters, char1 and char2, and replace all occurrences of char1 in the second string with

char2.

For example, the string "xxyz" can be converted to any of the following strings by performing the

operation as mentioned above: "yyyz", "vvyz", "xxzz", "xxwz", etc. However, it cannot be converted to

the string "yzyz" (Since all the occurrences of a character can be replaced by a single new character).

Note that doing both operations simultaneously is not allowed. For example, given two strings "xz" and

"zx", we can not convert both the x's in the respective strings to 'z' to make them equal.

Given two arrays of n strings each, compare the strings present at the same indices in both the arrays

and find if they can be made equal or not. Return an array of n strings, where the i element would be

YES if the strings at index i can be equalized; otherwise, the answer is NO.

Example

Consider the array of strings to be:

firstStrings = ["aaa", "abbc", "zyz"]

secondStrings = ["bbb", "cccc", "zyx"]

th

4 of 21

Let us compare the strings at each index:

i = 0: firstStrings[0] = "aaa", secondStrings[0] = "bbb"

We can change all the occurrences of 'a' to 'b' in the first string in this case. Therefore, the answer is

"YES".

i = 1: firstStrings[1] = "abbc", secondStrings[1] = "cccc"

There is no way in which we can make these strings identical. Therefore, the answer is "NO".

i = 2: firstStrings[2] = "zyz", secondStrings[2] = "zyx"

We can replace the only occurrence of 'x' in the second string with 'z' to make them identical.

Therefore, the answer is "YES".

Hence, the answer corresponding to each pair of strings is ["YES," "NO," "YES"].

Function Description

Complete the function canBeEqualized in the editor below.

canBeEqualized has the following parameters:

firstStrings[firstStrings[0],...firstStrings[n-1]]: an array of strings representing the first string of each

pair

secondStrings[secondStrings[0],...secondStrings[n-1]]: an array of strings representing the second

string of each pair

Returns

string[]: an array of strings representing the answer to each pair of strings

Constraints

1 ≤ n ≤ 10

1 ≤ |firstStrings[i]|, |secondStrings[i]| ≤ 10

All the strings consist of lowercase English characters only.

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, n, denoting the number of elements in firstStrings.

Each line i of the n subsequent lines (where 0 ≤ i < n) contains a string describing firstStrings[i].

The next line contains an integer, n, denoting the number of elements in secondStrings.

Each line i of the n subsequent lines (where 0 ≤ i < n) contains a string describing secondStrings[i].

SECOND STRINGSSAMPLE CASE 0

3

3

5 of 21

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

2 → firstStrings[] size n = 2

ac → firstStrings = ["ac", "pqqrr"]

pqqrr

2 → secondStrings[] size n = 2

ca → secondStrings = ["ca", "prrrr"]

prrrr

Sample Output

NO

YES

Explanation

We can check that there is no way to make them equal for the first pair of strings. Thus the answer

is "NO".

For the second pair of strings, we can convert all the occurrences of 'q' in the first string to 'r', thus

the answer is "YES".

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

2 → firstStrings[] size n = 2

a → firstStrings = ["a", "aaabbb"]

aaabbb

2 → secondStrings[] size n = 2

b → secondStrings = ["b", "abbbbb"]

abbbbb

Sample Output

YES

NO

Explanation

For the first pair of strings, we can convert the only occurrence of 'a' to 'b', thus the answer is "YES".

For the second pair of strings, it can be seen that it is impossible to make the two strings equal,

thus the answer is "NO".

6 of 21

Interviewer guidelines

SOLUTION

Topic Covered: loops and counters, arrays, strings

Optimal Solution: This problem can be solved by iterating over all n pairs of strings, and finding the

answers one by one. To verify whether two strings are similar, we need to make sure the following:

The length of both of these strings is the same.

There is at most one pair of characters that mismatch in the corresponding indices.

For this pair of mismatch, we should ensure that at least one of the strings don't have this character to

be paired up with the same character.

COMPLEXITY ANALYSIS

Time Complexity - O(n\*len) where len represents the maximum length of firstString overall given

strings.

Space Complexity - O(1) as we're using constant space.

: Java 17

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'canBeEqualized' function below.

\*

\* The function is expected to return a STRING\_ARRAY.

\* The function accepts following parameters:

\* 1. STRING\_ARRAY firstStrings

\* 2. STRING\_ARRAY secondStrings

\*/

public static List<String> canBeEqualized(List<String> firstStrings,

List<String> secondStrings) {

**// Write your code here**

**List<String> list = new ArrayList<>();**

**int flag = 0;**

**for (int i = 0; i < firstStrings.size(); i++) {**

**if (firstStrings.get(i).length() != secondStrings.get(i).length()) {**

**list.add("NO");**

**continue;**

**}**

**for (int j = 0; j < firstStrings.get(i).length(); j++) {**

**String temp = secondStrings.get(i).replaceAll(secondStrings.get(i).charAt(j) + "", firstStrings.get(i).charAt(j) + "");**

**if (temp.equals(firstStrings.get(i))) {**

**list.add("YES");**

**flag = 1;**

**break;**

**}**

**}**

**if (flag == 0) {**

**for (int j = 0; j < firstStrings.get(i).length(); j++) {**

**String temp = firstStrings.get(i).replaceAll(firstStrings.get(i).charAt(j) + "", secondStrings.get(i).charAt(j) + "");**

**if (temp.equals(secondStrings.get(i))) {**

**list.add("YES");**

**flag = 1;**

**break;**

**}**

**}**

**}**

**if (flag == 0)**

**list.add("NO");**

**flag = 0;**

**}**

**return list;**

**}**

}

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int firstStringsCount = Integer.parseInt(bufferedReader.readLine().trim());

List<String> firstStrings = IntStream.range(0, firstStringsCount).mapToObj(i -> {

try {

return bufferedReader.readLine();

} catch (IOException ex) {

throw new RuntimeException(ex);

}

}).collect(toList());

int secondStringsCount = Integer.parseInt(bufferedReader.readLine().trim());

List<String> secondStrings = IntStream.range(0, secondStringsCount).mapToObj(i -> {

try {

return bufferedReader.readLine();

} catch (IOException ex) {

throw new RuntimeException(ex);

}

}).collect(toList());

List<String> result = Result.canBeEqualized(firstStrings, secondStrings);

bufferedWriter.write(

result.stream()

.collect(joining("\n"))

+ "\n"

);

bufferedReader.close();

bufferedWriter.close();

}

}

**Java: Data Encryption**

Question description

You are required to customize a class named DataEncryption, having the following attributes:

encryptionRatio, baseEncryptionValue, and a method titled encryptValue().

The default encryptValue() function works by calculating the encrypted value as follows:

encryptedValue = baseEncryptionValue + encryptionRatio

Add more functionality to the existing method encryptValue() so that it accepts a variable named

encryptionAdjustment of integer data type. The function should accordingly adjust the calculation of the

encrypted value as:

encryptedValue = (baseEncryptionValue + encryptionRatio) - encryptionAdjustment

Additionally, you are asked to overload this method so that it accepts encryptionAdjustment as a string

type, converts it to an integer, and further uses it in the calculation of the encrypted value.

encryptedValue = (baseEncryptionValue + encryptionRatio) - encryptionAdjustment

There are three overloaded versions of encryptValue() in the DataEncryption class:

1. Default Version: Takes no parameters and calculates the encrypted value as baseEncryptionValue +

encryptionRatio .

2. First Overloaded Version: Accepts an integer encryptionAdjustment and calculates the encrypted

value as (baseEncryptionValue + encryptionRatio) - encryptionAdjustment .

3. Second Overloaded Version: Accepts a string encryptionAdjustment , converts it to an integer, and

calculates the encrypted value as (baseEncryptionValue + encryptionRatio) - encryptionAdjustment .

Example

encryptionRatio = 50

baseEncryptionValue = 200

encryptionAdjustment = 14

11 of 15

Default, encryptedValue = 250

With the first method modification, encryptedValue = 236

With the second method modification, encryptedValue = 236

INPUT FORMAT FOR CUSTOM TESTING

A single line of input consists of space-separated integers: encryptionRatio, baseEncryptionValue,

and encryptionAdjustment.

SAMPLE CASE 0

Sample Input For Custom Testing

0 0 0

Sample Output

0 0 0

SAMPLE CASE 1

Sample Input For Custom Testing

10 12 4

Sample Output

22 18 18

Interviewer guidelines

: Java 8

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

class DataEncryption {

private int encryptionRatio;

private int baseEncryptionValue;

public DataEncryption(int encryptionRatio, int baseEncryptionValue) {

this.encryptionRatio = encryptionRatio;

this.baseEncryptionValue = baseEncryptionValue;

}

public int encryptValue() {

int encryptedValue = baseEncryptionValue + encryptionRatio;

return encryptedValue;

}

public int encryptValue(int encryptionAdjustment) {

int encryptedValue = (baseEncryptionValue + encryptionRatio) - encryptionAdjustment;

return encryptedValue;

}

public int encryptValue(String encryptionAdjustment) {

int a = Integer.parseInt(encryptionAdjustment);

int encryptedValue = (baseEncryptionValue + encryptionRatio) - a;

return encryptedValue;

}

}

public class Solution {

public static void main(String args[]) throws Exception {

// reader and writer

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

String[] values = bufferedReader.readLine().split(" ");

// check if int array

try {

Integer.parseInt(values[0]);

// System.out.println("Integer array");

Integer[] ia = new Integer[values.length];

for (int i = 0; i < values.length; i++) {

ia[i] = new Integer(values[i]);

}

DataEncryption dataEncryption = new DataEncryption(ia[0], ia[1]);

bufferedWriter.write(

dataEncryption.encryptValue() + " "

+ dataEncryption.encryptValue(ia[2]) + " "

+ dataEncryption.encryptValue(String.valueOf(ia[2]))

);

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

} catch (NumberFormatException nfe) {

// then string array

// System.out.println("String array");

System.out.println();

}

}

}

**Chess Tournament**

Coding Easy Queue

Question description

The city of Hackerland organized a chess tournament for its citizens.

There are n participants numbered 1 to n where the i participant has potential denoted by

potential[i]. The potential of each player is distinct. Initially, all players stand in a queue in order from

the 1 to the n player In each game, the first 2 participants of the queue compete and the participant

with a higher potential wins the game. After each game, the winner remains at the beginning of the

queue and plays with the next person from the queue and the losing player goes to the end of the

queue. The game continues until a player wins k consecutive games.

Given the potential of the participants and the deciding factor k, find the potential of the winning player.

Example

Consider n = 4 participants have potential = [3, 2, 1, 4], and k = 2.

Initial position of participants: [1, 2, 3, 4].

Participants 1 and 2 compete. Their potentials are 3 and 2. Player 1 wins due to the higher potential.

Player 1 stays at the front of the queue and player 2 moves to the back. Now their positions are [1, 3,

4, 2].

th

st th

Participants 1 and 3 compete. Their potentials are 3 and 1. 1 wins a second consecutive game. Since k

= 2, player 1 has won enough consecutive games.

Return player 1's potential, 3.

Function Description

Complete the function getPotentialOfWinner in the editor below.

getPotentialOfWinner has the following parameters:

int potential[n]: the potentials of participants

long\_int k: the number of consecutive matches the winning participant must win

Returns

int: the potential of the winning player

Constraints

2 ≤ n ≤ 10

1 ≤ potential[i] ≤ n

2 ≤ k ≤ 10

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, n, the number of elements in potential. Each line i of the n

subsequent lines (where 0 ≤ i < n) contains an integer, potential[i].

The next line contains an integer, k.

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

5 → potential[] size n = 5

1 → potential = [1, 3, 2, 4, 5]

3

2

4

5

2 → k = 2

5

14

Sample Output

3

Explanation

.

Initial position of participants: [1, 2, 3, 4, 5].

potential[1] = 1, potential[2] = 3, player 2 wins. The positions of participants after match 1: [2, 3, 4,

5, 1].

potential[2] = 3, potential[3] = 1, player 2 wins. Since k = 2, player 2 is the winner,

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

4 → potential[] size n = 4

3 → potential = [3, 2, 1, 4]

2143

→ k = 3

Sample Output

4

Explanation

Potentials Consecutive

Positions 1st in line 2nd in line Winner wins

------------ ----------- ----------- ------ -----------

[1, 2, 3, 4] 3 2 1 1

[1, 3, 4, 2] 3 1 1 2

[1, 4, 2, 3] 3 4 4 1

[4, 2, 3, 1] 4 2 4 2

[4, 3, 1, 2] 4 1 4 3

Interviewer guidelines

SOLUTION

Skills: Arrays, Dequeues

Optimal Solution:

The maximum number of games needed to decide the winner can never exceed (n + k). Moreover, if k

≥ n - 1, it can be seen that the answer is n. This is because in the first pass, there will be n - 1 games

scheduled, and the player with maximum potential = n, will be at the front of the queue. Since this

player's potential is maximum, they will win all further games no matter the count, hence the potential

of the winner will be n.

For cases where k < n - 1, a double-ended queue can be used to simulate the games and determine

the potential of the winner.

Time Complexity: O(n)

Space Complexity: O(n)

COMPLEXITY ANALYSIS

Time Complexity - O(n)

For cases where k ≥ n - 1, the answer is returned in O(1) time. For cases where k < n - 1, as already

stated, the maximum number of games can be n + k, and since k < n - 1, this is of the order O(n).

Hence, time complexity is O(n).

Space Complexity - O(n)

The deque stores n entries, hence space complexity is O(n).

: Java 8

import java.io.\*;

import java.math.\*;

import java.security.\*;

import java.text.\*;

import java.util.\*;

import java.util.concurrent.\*;

import java.util.function.\*;

import java.util.regex.\*;

import java.util.stream.\*;

import static java.util.stream.Collectors.joining;

import static java.util.stream.Collectors.toList;

class Result {

/\*

\* Complete the 'getPotentialOfWinner' function below.

\*

\* The function is expected to return an INTEGER.

\* The function accepts following parameters:

\* 1. INTEGER\_ARRAY potential

\* 2. LONG\_INTEGER k

\*/

public static int getPotentialOfWinner(List<Integer> potential, long k) {

**// Write your code here**

**int n = potential.size();**

**int x = potential.get(0);**

**ArrayList<Integer> queue = new ArrayList<>(potential.subList(1, n));**

**int c = 0;**

**while (true) {**

**int y = queue.get(0);**

**queue.remove(0);**

**if (x > y) {**

**queue.add(y);**

**c++;**

**} else {**

**queue.add(x);**

**x = y;**

**c = 1;**

**}**

**if (c == k || c == n - 1) {**

**return x;**

**}**

**}**

**}**

}

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int potentialCount = Integer.parseInt(bufferedReader.readLine().trim());

List<Integer> potential = IntStream.range(0, potentialCount).mapToObj(i -> {

try {

return bufferedReader.readLine().replaceAll("\\s+$", "");

} catch (IOException ex) {

throw new RuntimeException(ex);

}

})

.map(String::trim)

.map(Integer::parseInt)

.collect(toList());

long k = Long.parseLong(bufferedReader.readLine().trim());

int result = Result.getPotentialOfWinner(potential, k);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}

**Java: Employee Profile**

Question description

Implement the following classes:

1. abstract class Employee with the following methods:

\* abstract void setSalary(int salary) method

\* abstract int getSalary() method

\* abstract void setGrade(String grade) method (grade of the employee in the organization)

\* abstract String getGrade() method

\* void label() method which prints "Employee's data:\n" (Concrete method, implementation is hidden

from end user)

2. class Engineer extending class Employee:

\* private attribute int salary

\* private attribute String grade

\* implement the setter and getter methods from the parent class to set and get attributes (salary and

grade)

3. class Manager extending class Employee:

\* private attribute int salary

\* private attribute String grade

\* implement the setter and getter methods from the parent class to set and get attributes (salary and

grade)

Note: The code stub handles input and calls the methods.

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, n, that denotes the number of employees to be instantiated.

Each line i of the n subsequent lines (where 0 ≤ i < n) contains 3 variables.

TYPE\_OF\_EMPLOYEE GRADE SALARY

SAMPLE CASE 0

Sample Input For Custom Testing

2 ENGINEER B

50000

MANAGER A 70000

Sample Output

Employee's data:

GRADE : B

SALARY : 50000

Employee's data:

GRADE : A

SALARY : 70000

SAMPLE CASE 1

Sample Input For Custom Testing

3 ENGINEER B

50000

MANAGER A 70000

MANAGER A 90000

Sample Output

Employee's data:

GRADE : B

SALARY : 50000

Employee's data:

GRADE : A

SALARY : 70000

Employee's data:

GRADE : A

SALARY : 90000

Interviewer guidelines

: Java 8

import java.io.\*;

import java.util.\*;

import java.text.\*;

import java.math.\*;

import java.util.regex.\*;

//Write The Code Here

**abstract class Employee {**

**private int salary;**

**private String grade;**

**public abstract void setSalary(int salary);**

**public abstract int getSalary();**

**public abstract void setGrade(String grade);**

**public abstract String getGrade();**

**public void label() {**

**System.out.println("Employee's data:");**

**}**

**}**

**class Engineer extends Employee {**

**private int salary;**

**private String grade;**

**public void setSalary(int salary) {**

**this.salary = salary;**

**}**

**public int getSalary() {**

**return salary;**

**}**

**public void setGrade(String grade) {**

**this.grade = grade;**

**}**

**public String getGrade() {**

**return grade;**

**}**

**}**

**class Manager extends Employee {**

**private int salary;**

**private String grade;**

**public void setSalary(int salary) {**

**this.salary = salary;**

**}**

**public int getSalary() {**

**return salary;**

**}**

**public void setGrade(String grade) {**

**this.grade = grade;**

**}**

**public String getGrade() {**

**return grade;**

**}**

**}**

public class Solution {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String sub = sc.nextLine();

int n = Integer.parseInt(sub);

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

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

if (input[0].equals("ENGINEER")) {

Engineer e = new Engineer();

e.setSalary(Integer.parseInt(input[2]));

e.setGrade(input[1]);

e.label();

System.out.println("GRADE : " + e.getGrade());

System.out.println("SALARY : " + e.getSalary());

}

if (input[0].equals("MANAGER")) {

Manager e = new Manager();

e.setSalary(Integer.parseInt(input[2]));

e.setGrade(input[1]);

e.label();

System.out.println("GRADE : " + e.getGrade());

System.out.println("SALARY : " + e.getSalary());

}

}

}

}

Similar Strings

Coding Strings Easy Sorting Hash Map

Question description

Two strings are said to be similar if they are composed of the same characters. For example "abaca" and

"cba" are similar since both of them are composed of characters 'a', 'b' and 'c'. However "abaca" and

"bcd" are not similar since they do not share all of the same letters.

Given an array of strings words of length n, find the number of pairs of strings that are similar.

Note:

Each string is composed of lowercase English characters only.

Pairs are considered index-wise, i.e., two equal strings at different indices are counted as separated

pairs.

A pair at indices (i, j) is the same as the pair at (j, i).

Example

Consider n = 3, words = ["xyz", "foo", "of"].

Here, the strings "foo" and "of" are similar because they are composed of the same characters ['o', 'f'].

There are no other similar pairs so the answer is 1.

Function Description

Complete the function countSimilarPairs in the editor below.

countSimilarPairs has the following parameter:

string words[n]: an array of n strings

Returns

long\_int : the number of similar pairs

Constraints

1 ≤ n ≤ 10

The Sum of the lengths of all strings does not exceed 10 .

All strings consist of lowercase English characters only.

INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, n, the number of strings in words. Each line i of the n subsequent

lines (where 0 ≤ i < n) contains a string, words[i].

SAMPLE CASE 0

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

4 → words[] size n = 4

aba → words = ["aba", "abaca", "baab", "cba"]

abaca

baab

cba

Sample Output

2

Explanation

Strings "aba" and "baab" are similar because they are made up of the same characters 'a' and 'b'.

Strings "abaca" and "cba" are similar because they are made up of the same characters 'a', 'b' and

'c'.

SAMPLE CASE 1

Sample Input For Custom Testing

STDIN FUNCTION

----- --------

3 → words[] size n = 3

abab → words = ["abab", "aaaaab", "bbbba"]

aaaaab

bbbba

5

6

Sample Output

3

Explanation

These 3 pairs are made up of 'a' and 'b':

"abab" and "aaaaab"

"abab" and "bbbba"

"aaaaab" and "bbbba"

Interviewer guidelines

SOLUTION

Skills: Strings, Hashing, Sorting

Optimal Solution:

We iterate over all strings, and for each string we store the characters present in a string using a set.

Two strings are similar if their respective sets contain the same elements. So, we sort the sets, and

maintain a map of the corresponding sorted set. Now, if there are x occurrences of a sorted set, the

number of similar pairs corresponding to this set is (x \* (x - 1)) / 2. So, we obtain the count for each

sorted pair and sum it up to get the answer.

Time Complexity: O(n + sum(x log(x )), where x is the number of distinct characters in the i word

Space Complexity: O(n)

COMPLEXITY ANALYSIS

Time Complexity - O(n + sum(x log(x )), where x is the number of distinct characters in the i word

For each word, we store all distinct characters in a set, and then store them in sorted order. Sorting a

string of length k takes O(k logk) time. Thus, sorting all strings takes O(sum(x log(x )) time.

Furthermore, we iterate over all map entries to compute the answer, and the map can have at most n

entries. Thus, the time complexity is O(n + sum(x log(x )).

Space Complexity - O(n)

The map can have at most n entries, thus space complexity is O(n).

: Java 17

class Result {

/\*

\* Complete the 'countSimilarPairs' function below.

\*

\* The function is expected to return a LONG\_INTEGER.

\* The function accepts STRING\_ARRAY words as parameter.

\*/

public static long countSimilarPairs(List<String> words) {

// Write your code here

HashMap<String, Integer> ans = new HashMap<>();

for (String i : words) {

TreeSet<Character> chec = new TreeSet<>();

for (int j = 0; j < i.length(); j++) {

chec.add(i.charAt(j));

}

String str = "";

for (char j : chec) {

str += j;

}

ans.put(str, ans.getOrDefault(str, 0) + 1);

}

long count = 0L;

for (int i : ans.values()) {

long a = (long) i;

if (i > 1) {

count += (a \* (a - 1)) / 2L;

}

}

return count;

}

}

public class Solution {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT\_PATH")));

int wordsCount = Integer.parseInt(bufferedReader.readLine().trim());

List<String> words = IntStream.range(0, wordsCount)

.mapToObj(i -> {

try {

return bufferedReader.readLine();

} catch (IOException ex) {

throw new RuntimeException(ex);

}

})

.collect(Collectors.toList());

long result = Result.countSimilarPairs(words);

bufferedWriter.write(String.valueOf(result));

bufferedWriter.newLine();

bufferedReader.close();

bufferedWriter.close();

}

}