

In this challenge, you need to determine if a card number is valid. A card number includes 16 digits:  
- The first 6 correspond to the issuing bank,  
- 9 digits are taken randomly,  
- the last digit is the Luhne key  
  
The Luhne key is based on the first 15 digits. One digit out of two is multiplied by 2 starting with the first one. If the result is greater than 9 then 9 is deduced from the result. This provides a new series of digits. The Luhne key is the value that needs to be added to this sum to get a number that can be divided by 10.  
Example: **5295 4648 5201 3672**

/\*\*\*\*\*\*\*

\* Solution by Yan Shpilsky

\*\*\*\*\*\*\*\*\*/

package com.isograd.exercise;

import java.util.\*;

public class IsoContest {

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

String line;

Scanner sc = new Scanner(System.in);

int numcards = 0;

int numvalidcards = 0;

int j = 0;

while(sc.hasNextLine()) {

line = sc.nextLine();

/\*Read the input here and perform the calculations\*/

if (j==0) numcards=Integer.parseInt(line);

else {

char[] charArray = line.toCharArray();

int[] finalArray = new int[16];

for (int i = 0; i < 15; i++){

String s = ""+charArray[i];

int number = Character.getNumericValue(s.charAt(0));

if (i % 2 == 0 || i == 0){

finalArray[i] = (number \* 2);

if (finalArray[i] > 9) finalArray[i] -= 9;

}

else finalArray[i] = number;

}

int sum = 0;

for (int i = 0; i < 16; i++){

sum += finalArray[i];

}

sum += Character.getNumericValue(charArray[15]);

if (sum % 10 == 0) {

numvalidcards++;

}

}

j++;

}

System.out.println(numvalidcards);

/\*You can also perform the calculations once you have read all the data.\*/

}

}

**2.Question :**

Let's start with an easy challenge... You just have to compute how much is left on an account after a series of transactions. You are provided with an opening balance, and then you get a series of transactions that either increase or decrease the balance. Transactions will be represented by amounts, negative when they reduce the balance and positive when they increase the balance.  
  
You can assume that the balance will never be negative.  
  
**Data**  
  
Input  
Row 1 : an integer between *50,000* and *100,000* representing the opening balance.  
Row 2 : an integer **N** between *1* and *100* representing the number of transactions.  
Row 3 to **N** + 2 : an integer between *-1000* and *1000*representing a transaction.  
  
Output  
An integer representing the closing balance.

/\*\*\*\*\*\*\*

\* Solution by John Bigelow

\*\*\*\*\*\*\*\*\*/

package com.isograd.exercise;

import java.util.\*;

public class IsoContest {

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

String line;

Scanner sc = new Scanner(System.in);

String startS=sc.nextLine();

int start=Integer.parseInt(startS);

int numLines = Integer.parseInt(sc.nextLine());

int trancount = 0;

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

/\*Read the input here and perform the calculations\*/

String val = sc.nextLine();

int debit = Integer.parseInt(val);

start = start + debit;

}

System.out.println(start);

}

}

**Question 3.**

In this challenge, you need to determine the time at which the most employees are working simultaneously in the building.  
  
**Data format**  
  
Input  
Row 1: an integer **N** representing the total number of people that enter the building over one day.  
Row 2 to **N** + 1: 2 integers between *0* and *24*, separated by a space, representing the hours of arrival and departure of every employee.  
  
To simplify the challenge, the first integer on each row will be strictly smaller than the second (e.g., no employee stays overnight, and every employee stays at least one hour). Please note that if a person is working from 8 o'clock to 16 o'clock, it means that he or she is **not** working at 16 o'clock.  
  
Output  
An integer between 0 and 24 representing the time at which most people are working inside the building. If there are multiple hours where the maximum is reached display all of them sorted ascendingly and separated by spaces.

/\*\*\*\*\*\*\*

\* Solution by Divya Arora

\*\*\*\*\*\*\*\*\*/

package com.isograd.exercise;

import java.util.\*;

public class IsoContest {

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

IsoContestBase.localEcho("Start");

String line;

int noOfPeople=0;

Map<Integer,Integer> hourMap = new HashMap<>();

Scanner sc = new Scanner(System.in);

line = sc.nextLine();

noOfPeople = Integer.parseInt(line);

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

if(sc.hasNextLine()) {

line = sc.nextLine();

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

int arrival = Integer.parseInt(times[0]);

int dep = Integer.parseInt(times[1]);

for(int x=arrival; x<dep; x++){

int count=1;

if(hourMap.containsKey(x))

count = hourMap.get(x) + 1;

hourMap.put(x, count);

}

}

}

ArrayList<Integer> maxHourList = new ArrayList<Integer>();

int max=-1;

for(Integer key : hourMap.keySet()){

if(hourMap.get(key) > max){

max = hourMap.get(key);

}

}

for(Integer key : hourMap.keySet()){

if(hourMap.get(key) == max){

maxHourList.add(key);

}

}

Collections.sort(maxHourList);

StringBuffer sb = new StringBuffer();

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

sb.append(maxHourList.get(i)+ " ");

}

System.out.println(sb.toString().trim());

//IsoContestBase.dump("Map", );

}

}

**Question 4 :**

When a cash dispenser handles a cash request, a classical method is to provide in priority the largest note then to provide smaller notes to cover the rest of the request and to continue like this until the request is filled.   
  
Example : if the request is $190 and the cash dispenser has $100, $50, $20, and $10 notes, the cash dispenser will provide a $100 note and a $50 note and 2 $20 notes.  
  
In this challenge, you will be provided with a cash request and the list of available notes and you will have to determine the notes that the dispenser provides according to the above algorithm. You can assume that the dispenser has a sufficient quantity of each notes to fill the withdrawal.  
  
The challenge will not use realistic data, notes can have value like *12311*and the withdrawal can be very high.  
  
**Data Format**  
  
Input  
Row 1 : an integer between *1* and *100,000* representing the amount that you have to deliver.  
Row 2 : an integer **N** representing the number of types of notes that the cash dispenser can use.  
Row 3 to **N**+2 : an integer between *1* and *50,000* representing the face values of the notes that are available. The notes are sorted ascendingly.  
  
You can assume that the dispenser will always hold the right notes to match the withdrawal.  
  
Output  
  
A series of pairs of integers **Xi** and **Yi** separated by a space. For each type of note used to fill the order, **Xi** represents the number of notes of value **Yi**that you want to deliver. The pairs have to be sorted in descending order of Yi. If we take the example above, the output would be :*1 100 1 50 2 20*

/\*\*\*\*\*\*\*

\* Solution by Naga Karavadi

\*\*\*\*\*\*\*\*\*/

package com.isograd.exercise;

import java.util.\*;

public class IsoContest {

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

String line;

Scanner sc = new Scanner(System.in);

while(sc.hasNextLine()) {

line = sc.nextLine();

/\*Read the input here and perform the calculations\*/

int amountToDeliv = Integer.parseInt(line);

line = sc.nextLine();

int numberOfTypes = Integer.parseInt(line);

TreeMap<Integer, Integer> typeAndNumberMap = new TreeMap<Integer, Integer>();

for (int i = 1; i <= numberOfTypes; i++) {

line = sc.nextLine();

typeAndNumberMap.put(Integer.parseInt(line), Integer.MAX\_VALUE);

}

TreeMap<Integer, Integer> dispenseCurCntMap = new TreeMap<Integer, Integer>();

while(amountToDeliv > 0) {

for(Integer face: typeAndNumberMap.descendingKeySet()) {

if (face < amountToDeliv) {

int count = amountToDeliv/face;

amountToDeliv -= face\*count;

int rem = typeAndNumberMap.get(face) -count;

if(dispenseCurCntMap.get(face) == null) {

dispenseCurCntMap.put(face, count);

}

else {

dispenseCurCntMap.put(face, dispenseCurCntMap.get(face) + count);

}

if (rem == 0) {

typeAndNumberMap.remove(face);

}

else {

typeAndNumberMap.put(face, rem);

}

}

}

}

print(dispenseCurCntMap);

return;

}

/\*You can also perform the calculations once you have read all the data.\*/

}

public static void print(TreeMap<Integer, Integer> dispenseCurCntMap) {

String out = "";

for(int face: dispenseCurCntMap.descendingKeySet()) {

out += dispenseCurCntMap.get(face) + " " + face + " ";

}

System.out.print(out.trim());

}

}

**Question 5**

The aim of this challenge is to determine which cash dispensers have delivered the largest amount of money. For this purpose you are provided with a log file that contains usage information.   
  
**Data Format**  
  
Input  
Row 1 : an integer **N** between *1* and *1000*indicating the number of entries in the log file.  
Row 2 to **N** + 1 : a string of *8* alphanumeric characters and an integer number between *10* and *1000*separated by a space. They represent the identifier of the cash dispenser and the amount withdrawn.  
  
Output  
The identifiers of the *10* dispensers that have distributed the most money ordered by descending amounts. Identifiers should be separated by a space. If there are less than *10* dispensers in the log file, display them all. If multiple dispensers have delivered the same amount of cash then order them by identifier in ascending order.

/\*\*\*\*\*\*\*

\* Solution by Maxim Deryuga

\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*

\* Read input from System.in

\* Use System.out.println to ouput your result.

\* Use:

\* IsoContestBase.localEcho( variable)

\* to display variable in a dedicated area.

\* \*\*\*/

package com.isograd.exercise;

import java.util.\*;

public class IsoContest {

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

String line;

int count = 0;

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

Scanner sc = new Scanner(System.in);

int i = 0;

while (sc.hasNextLine()) {

line = sc.nextLine();

/\* Read the input here and perform the calculations \*/

if (i == 0) {

count = Integer.parseInt(line);

} else {

String name = line.substring(0, 8);

int amount = Integer.parseInt(line.substring(9));

Integer val = map.get(name);

if (val != null) {

amount = amount + val.intValue();

}

map.put(name, Integer.valueOf(amount));

}

if (i == (count)) {

break;

}

i++;

}

ValueComparator bvc = new ValueComparator(map);

TreeMap<String, Integer> sorted\_map = new TreeMap<String, Integer>(bvc);

sorted\_map.putAll(map);

String result = "";

int ind = 0;

for (String ent : sorted\_map.keySet()) {

result += (result.length() > 0 ? " " : "") + ent;

ind++;

if (ind == 10) {

break;

}

}

System.out.print(result);

/\* You can also perform the calculations once you have read all the data. \*/

}

static class ValueComparator implements Comparator<String> {

Map<String, Integer> base;

public ValueComparator(Map<String, Integer> base) {

this.base = base;

}

public int compare(String a, String b) {

if (base.get(a) > base.get(b)) {

return -1;

} else if (base.get(a) < base.get(b)) {

return 1;

} else {

return a.compareTo(b);

}

}

}

}

**Question 6**

Following a flood, the bank needs to be evacuated. The building of modern design is rectangular and made of blocks. Each block has multiple levels that are linked by a stairway. Your office is located at the top left corner of the building and the exit is located at the bottom right corner. You can go through a block if and only if it is higher or equal that the water level. The water level is constant in the building. The goal of this challenge is to determine if you can exit the building.  
  
**Data format**  
  
Input  
Row 1 : 3 integers **W**,**L** and **H** space separated representing the width and length of the building, and the height that the water has reached. **W** and **L**are between *1* and *50*. **H** is between *1* and *100,000*.  
Row 2 to **W** + 1 : **L** integers between *1* and *100,000* separated by a space representing the height of the block for a given row. The first value of row 2 is consequently the top left corner of the building and the last value of row**W**+1 is the bottom right corner of the building.  
  
You can only move horizontally or vertically between two points of the map. Furthermore, top-left and bottom-right corners are at height 100,000, thus the entrance and exit are never underwater.  
  
Output  
The string *YES* if you can go from your office to the exit by only going through blocks that are higher or equal than the water level. Otherwise, the string *NO*.

/\*\*\*\*\*\*\*

\* Solution by Arnaud Lech

\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*

\* Read input from System.in

\* Use System.out.println to ouput your result.

\* Use:

\* IsoContestBase.localEcho( variable)

\* to display variable in a dedicated area.

\* \*\*\*/

package com.isograd.exercise;

import java.util.\*;

public class IsoContest {

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

String line;

Scanner sc = new Scanner(System.in);

int w = sc.nextInt();

int l = sc.nextInt();

int h = sc.nextInt();

IsoContestBase.localEcho("" + w + " " + l + " " + h);

int[][] building = new int[w][l];

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

for (int j = 0; j < l; j++) {

building[i][j] = sc.nextInt();

//IsoContestBase.localEcho("" + i + " " + j + " " + building[i][j]);

}

}

boolean result = move(building, w, l, h, 0, 0);

IsoContestBase.localEcho(result ? "YES" : "NO");

System.out.println(result ? "YES" : "NO");

}

private static boolean move(int[][] building, int w, int l, int h, int i, int j) {

if (i == w - 1 && j == l - 1) {

return true;

}

boolean result = false;

if (building[i][j] >= h) {

// good - continue

if (i + 1 < w) {

result = move(building, w, l, h, i + 1, j);

}

if (j + 1 < l) {

result = result || move(building, w, l, h, i, j + 1);

}

}

return result;

}

}

Question 7

Thanks to your amazing study about flooding, you just got promoted. Now, you need to take care of fires.

You're given the map of the city, including the location of every bank and fire station. The goal of this challenge is to find, upon every fire alarm in a bank, the distance to the closest fire station.

Data format

Input

Row 1: two space-separated integers W and L between 2 and 50 representing the width and length of the map.

Rows 2 to W + 1: L characters representing either:

- a bank 'B';

- a fire station 'F';

- an empty space '.'(dot).

To reach a bank from a fire station, there must be a path composed of empty spaces.

Row W + 2: an integer N between 1 and 20,000 representing the number of fire alarms in the city.

Rows W + 3 to W + 2 + N: two space-separated integers I and J representing the coordinates of a detected fire in a bank of the city.

Furthermore, there is at least one bank and one fire station. Several fire alarms can occur in the same bank. You can only either move horizontally or vertically between two points on the map.

Output

N space-separated integers representing for each fire alarm the distance from the bank on fire to the closest fire station if it exists, -1 otherwise.

/\*\*\*\*\*\*\*

\* Read input from System.in

\* Use System.out.println to ouput your result.

\* Use:

\* IsoContestBase.localEcho( variable)

\* to display variable in a dedicated area.

\* \*\*\*/

package com.isograd.exercise;

import java.util.\*;

public class IsoContest {

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

String line;

Scanner sc = new Scanner(System.in);

while(sc.hasNextLine()) {

line = sc.nextLine();

/\* Lisez les données et effectuez votre traitement \*/

}

/\* Vous pouvez aussi effectuer votre traitement une fois que vous avez lu toutes les données.\*/

}

}