# novática

**Revista de la Asociación de Técnicos de Informática**

***Novática***, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática), organización que edita también la revista **REICIS** (Revista Española de Innovación, Calidad e Ingeniería del Software).

**<http://www.ati.es/novatica/>**
**<http://www.ati.es/reicis/>**

**ATI** es miembro fundador de **CEPIS** (*Council of European Professional Informatics Societies*), representa a España en **IFIP** (*International Federation for Information Processing*) y es miembro de **CLEI** (*Centro Latinoamericano de Estudios de Informática*) y de **CECUA** (*Confederation of European Computer User Associations*). Asimismo tiene un acuerdo de colaboración con **ACM** (*Association for Computing Machinery*) y colabora con diversas asociaciones informáticas españolas.





**Coordinación Editorial, Redacción Central y Redacción ATI Madrid**
Gutierre de Cetina 24, 28017 Madrid • Tlfn.914029391 <novatica@ati.es>
**Administración y Redacción ATI Cataluña**
Calle Avila 50, 3a planta, local 9, 08005 Barcelona
Tlfn.934125235 <secregen@ati.es>
**Redacción ATI Andalucía** <secreand@ati.es>
**Redacción ATI Galicia** <secregal@ati.es>
**Suscripción y Ventas** <novatica.subscripciones@atinet.es>
**Publicidad** Gutierre de Cetina 24, 28017 Madrid
Tlnf.914029391 <novatica@ati.es>
**Imprenta:** Impresión Offset Derra S.L., Llull 41, 08005 Barcelona.
**Depósito legal:** B 15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC
**Portada:** "La decisión" - Concha Arias Pérez / © ATI
**Diseño:** Fernando Agresta / © ATI 2003


# sumario


**Nº 237, julio-octubre 2016, año XLII**




**Monografía del próximo número: "Seguridad digital"**

# El problema del robot de exploración de Marte

Julio Javier Castillo, Diego Javier Serrano, Marina Elizabeth Cárdenas
*Laboratorio de Investigación de Software MsLabs, Dpto. Ing. en Sistemas de Información, Facultad Regional Córdoba - Universidad Tecnológica Nacional (Argentina)*

**<jotacastillo@gmail.com>,**
**<diegojserrano@gmail.com>,**
**<ing.marinacardenas@gmail.com>**

Este es el enunciado del problema 5 que fue planteado en la Sexta Competencia de Programación de la Facultad Regional de Córdoba (Universidad Tecnológica Nacional, Argentina) UTN-FRC celebrada en octubre de 2014.

**Nivel del problema:** Medio

La Agencia Espacial Europea (ESA) es una organización internacional dedicada a la exploración espacial y está desarrollando un proyecto para colocar un robot de exploración sobre la superficie de Marte. El robot debe ser capaz de realizar exploraciones de forma remota controlado por un operador desde la Tierra ayudado por un programa que le permita realizar movimientos válidos. Dado que un mal movimiento del robot puede suponer el fracaso de la misión, el módulo de IA debe comportarse de forma conservadora y notificar permanentemente al robot acerca de posiciones peligrosas.

El robot parte siempre desde una posición base donde aterrizó la nave de transporte, que considera totalmente segura. A partir de allí, realiza movimientos de exploración siguiendo las órdenes del operador remoto. Cuando el módulo de IA determina que la posición actual es peligrosa, el robot realiza movimientos de regreso hacia la posición anterior para continuar desde ella su exploración por otros caminos.

El robot será capaz de realizar cuatro tipos de movimientos básicos: desplazamientos hacia adelante, atrás, derecha e izquierda. Para realizar un desplazamiento es necesario especificar la distancia (número entero que se representa en centímetros).

Como parte del equipo de ingenieros de la ESA, se le ha encargado realizar un programa que permita registrar los movimientos del robot de tal forma que éste sepa volver en todo momento a la posición base deshaciendo movimientos básicos hechos previamente. De esta manera, el operador del robot podrá ser capaz de conocer la ruta segura que debe realizar el robot para volver a la nave y la distancia total recorrida por el mismo.

Para poder identificar los obstáculos, la zona de navegación del robot está representada con una matriz dada, donde cada celda representa 1 centímetro cuadrado. El robot parte de la posición determinada en el punto medio de la matriz donde se encuentra la nave (por ejemplo para una matriz de 10 x 10 se considera el punto medio en la celda 4 x 4, para una matriz de 15 x 15 se considera el punto medio la celda 7 x 7) y procede con la navegación por la grilla según las indicaciones del operador. Previamente se ha realizado un reconocimiento del terreno por medio de imágenes satelitales y se han cargado los datos de los obstáculos en una matriz. Si en el desplazamiento del robot, se detecta algún obstáculo, el mismo retrocede a la posición anterior segura conocida, es decir a la posición anterior a la realización del desplazamiento actual. Cuando el operador termina la navegación el sistema debe mostrar el camino que el robot debe realizar hasta la nave y la distancia total recorrida.

**Entrada**

Se indicará con un número las dimensiones del área de exploración definida por una matriz cuadrada binaria. Las siguientes líneas indicarán cómo está completa el área de exploración, donde los obstáculos estarán indicados por el número "1" y los espacios sin obstáculos estarán indicados por el número "0".

En la siguiente línea se indicará la cantidad de desplazamientos que debe realizar el robot desde la posición inicial, identificando cada uno de los movimientos con un número (0: Arriba, 1: Abajo, 2: Derecha, 3: Izquierda) y, separada por un espacio, la distancia recorrida. Los movimientos que forman parte de un desplazamiento estarán separados entre sí por una coma. Por ejemplo: 0 10, 2 15, 1 25, 3 10 (suponiendo una matriz cuadrada de, al menos, 30 x 30) lo que se traduce a 10cm a hacia arriba, 15cm a la derecha, 25cm hacia abajo y 10cm a la izquierda.

**Salida**

Por cada caso de prueba (cada desplazamiento), se deberá imprimir la distancia total recorrida si los movimientos desde el origen hacia el destino están libres de obstáculos, y "NO SEGURO" en el caso que se detecte un obstáculo en alguna parte del desplazamiento. En caso que la distancia recorrida por un determinado movimiento supere las dimensiones de la matriz, se considerará la presencia de un obstáculo.

**Ejemplo de entrada**

```
10
1 1 1 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 0
0 0 1 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0 0 1
1 0 1 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 1
0 0 0 0 1 1 0 0 0 1
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0

5
0 10, 2 15, 1 25, 3 10
3 1, 1 5, 2 5, 0 7
3 1, 1 5, 2 5, 0 7, 3 1
1 5, 3 4, 2 5, 0 5
1 2, 2 4, 0 5, 3 5, 1 8
```

**Ejemplo de salida**

```
NO SEGURO
18
NO SEGURO
NO SEGURO
24
```

# Discos duros

Julio Javier Castillo, Diego Javier Serrano, Marina Elizabeth Cárdenas
*Laboratorio de Investigación de Software MsLabs, Dpto. Ing. en Sistemas de Información, Facultad Regional Córdoba - Universidad Tecnológica Nacional (Argentina)*

**<jotacastillo@gmail.com>, <diegojserrano@gmail.com>, <ing.marinacardenas@gmail.com>**

El enunciado de este problema apareció en el número 236 de ***Novática*** (abril-junio 2016, p. 79).

En este problema se nos solicita programar un algoritmo que permita calcular el espacio de almacenamiento que ocupan los directorios de determinados discos duros, e informar la cantidad de kilobytes que se ahorrarían si se eliminan las copias de archivos repetidos.

Para resolver este problema se propone el uso de tablas de dispersión (*hash*) que nos permitan almacenar (sin repeticiones) los nombres de los directorios de un disco duro, y la cantidad de bytes que ocupan todos los archivos de ese directorio. Asimismo, se propone el uso de otra tabla de dispersión cuyo valor de clave esté representado por el nombre del archivo y su tamaño, y el contenido con un valor igual al tamaño del archivo.

De esta manera, el problema puede resolverse realizando una tabla *hash* de acumulación, la cual se encuentra representada por un HashMap de "directorios" en la solución propuesta. Para ello se analizarán cada uno de los archivos del disco duro y se acumulará el tamaño de los mismos.

Evidentemente, este problema puede ser resuelto mediante el empleo de otras estructuras de datos como vectores o listas, pero se ha optado por el uso de tablas de dispersión dado que presentan un costo computacional menor respecto de las otras estructuras mencionadas. El costo de acceso y de acumulación en el interior de la tabla es constante.

La solución propuesta se codifica en el lenguaje de programación Java y utiliza las clases HashMap que son las estructuras de tablas de dispersión con las que cuenta Java en su paquete `java.util`.

La primera parte del programa lee los casos de prueba que en el problema representan diferentes discos duros. Seguidamente, se leen los archivos junto con los tamaños que los mismos ocupan. Se emplea el método `lastIndexOf()` de la clase `String` para encontrar la última posición de las "\\", los cuales sirven para delimitar el nombre del archivo y su ruta. Esta información se utiliza para poblar las tablas de dispersión.

Debemos notar que la tabla denominada "archivos" utiliza como clave al valor "nombre_archivo+tamaño" ya que es una manera de identificar las repeticiones, dado que en el planteo del problema solo se consideran archivos repetidos a aquellos que tengan el mismo nombre y tamaño.

La visualización de los resultados en consola se lleva a cabo realizando una iteración (mediante la estructura repetitiva denominada `for-each` de Java) sobre la tabla de dispersión. Nótese que los valores de bytes son divididos por 1.024 para obtener el correspondiente valor en kilobytes.

Finalmente, se provee como salida el valor del ahorro al "borrar archivos innecesarios" indicado en cantidad de kilobytes.

A continuación se presenta la solución del problema en el lenguaje de programación Java:

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class DiscosDuros {

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    String linea = sc.nextLine();
    int C = Integer.parseInt(linea);

    for (int caso = 1; caso <= C; caso++) {
    linea = sc.nextLine();
    int N = Integer.parseInt(linea);

    HashMap<String,Long> archivos = new HashMap<>();
    HashMap<String,Long> directorios = new HashMap<>();
    long ahorro = 0;

    for (int i = 0; i < N; i++) {
    linea = sc.nextLine();
    int posEsp = linea.lastIndexOf(" ");

    long tamaño = Long.parseLong(linea.substring(posEsp+1));
    int posUBI = linea.lastIndexOf("\\");
    int posPBI = linea.indexOf("\\",2);

    String nombreDir = linea.substring(1,posPBI);
    String nombreArchivo = linea.substring(posUBI+1,posEsp-1);
    String nombreCompleto = nombreArchivo + " " + tamaño;

    if (!archivos.containsKey(nombreCompleto))
    archivos.put(nombreCompleto, tamaño);
    else
    ahorro += tamaño;

    if (!directorios.containsKey(nombreDir))
    directorios.put(nombreDir, tamaño);
    else
    {
    Long tamActual = directorios.get(nombreDir);
    directorios.put(nombreDir, tamActual + tamaño);
    }
    }

    System.out.println("Disco duro " + caso + ":");

    for (Map.Entry<String, Long> dir: directorios.entrySet()) {
    Long tamañoKB = (long)Math.ceil(dir.getValue() / 1024f);
    System.out.println(dir.getKey() + " " + tamañoKB);
    }
    long ahorroKB = (long)Math.ceil(ahorro / 1024f);
    System.out.println(ahorroKB);
    }
  }
}
```