



AYUDANTÍA 15:
Repaso Examen – Recursión y Archivos
IIC1102 – Introducción a la Programación – Sección 4

PROBLEMAS

1. Suma Recursiva

Escriba un método recursivo que entregue el resultado de la suma de 2 enteros.

2. Los Conejos de Fibonacci

Cierto matemático italiano de nombre *Leonardo de Pisa*, pero mejor conocido como *Fibonacci*, propuso el siguiente problema: Suponga que acabamos de comprar una pareja de conejos adultos. Al cabo de un mes, esa pareja tiene una pareja de conejitos (*un conejo y una coneja*). Un mes después, nuestra primera pareja tiene otra pareja de conejitos (nuevamente, *un conejo y una coneja*) y, al mismo tiempo, sus primeros hijos se han vuelto adultos. Así que cada mes que pasa, cada pareja de conejos adultos tiene una pareja de conejitos, y cada pareja de conejos nacida el mes anterior se vuelve adulta. La pregunta es, ¿cuántas parejas de conejos adultos habrá al cabo de n meses? Para resolver este problema, llamemos F_n al número de parejas adultas al cabo de n meses. No es difícil convencerse de que si n es al menos 2, entonces F_n es igual a $F_{n-1} + F_{n-2}$. Así que F_n queda en términos de F_{n-1} y F_{n-2} , que a su vez quedan en términos de F_{n-2} , F_{n-3} y F_{n-4} , y así sucesivamente. Ahora salimos del *ciclo* recordando que al principio había una pareja de conejos adultos, la misma que había al final del primer mes, así que $F_0 = F_1 = 1$.

Ejemplo:

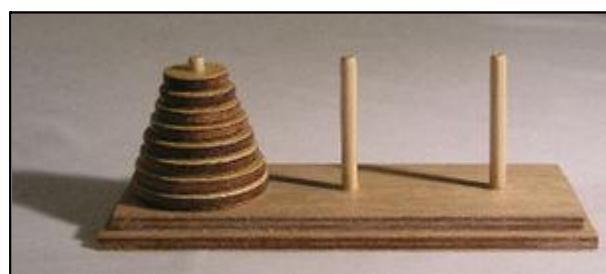
$$F_4 = F_3 + F_2 = (F_2 + F_1) + (F_1 + F_0) = ((F_1 + F_0) + 1) + (1 + 1) = ((1 + 1) + 1) + 2 = (2 + 1) + 2 = 3 + 2 = 5.$$

La sucesión de números $F_0 = 1, F_1 = 1, F_2 = 2, F_3 = 3, F_4 = 5, etc.$ recibe el nombre de **Sucesión de Fibonacci**.

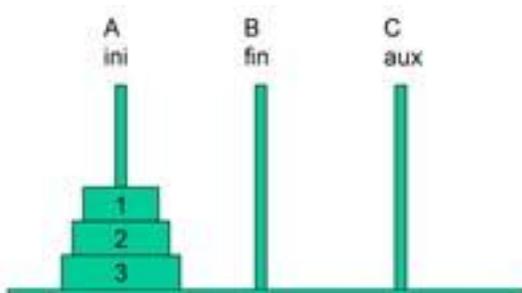
A continuación, escribe un método recursivo que resuelva el problema de Fibonacci.

3. Torres de Hanoi

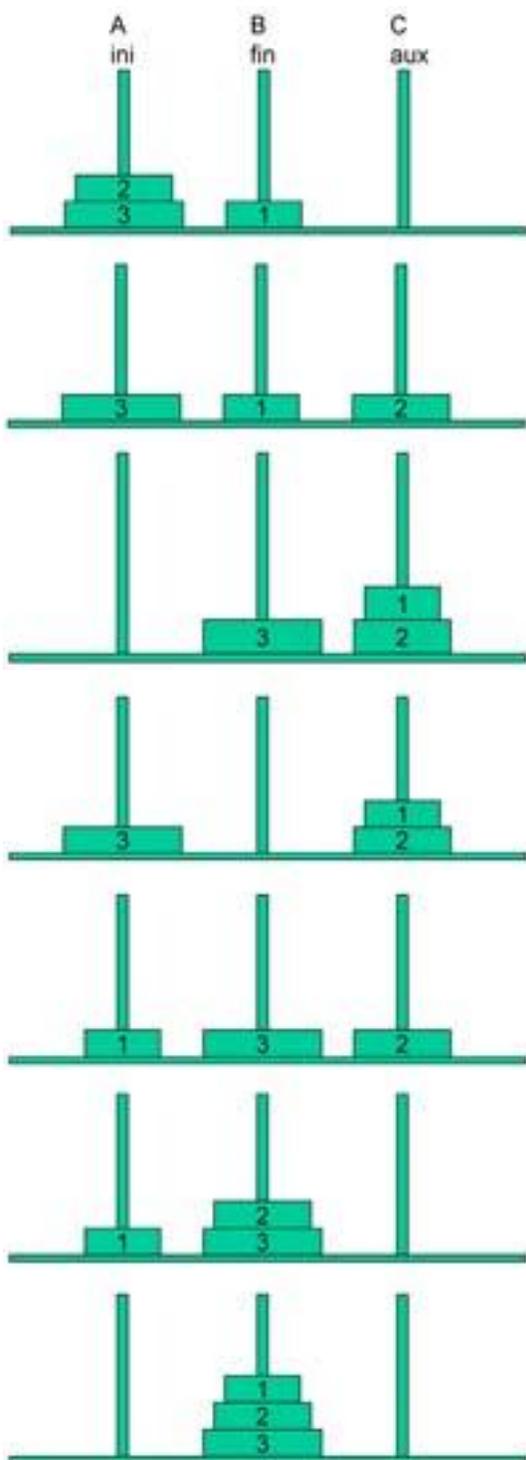
Las *Torres de Hanoi* es un juego matemático. Consiste en tres varillas verticales y un número indeterminado de discos que determinarán la complejidad de la solución. No hay dos discos iguales. Están colocados de mayor a menor en la primera varilla ascendentemente, y no se puede colocar ningún disco mayor sobre uno menor a él en ningún momento. El juego consiste en pasar todos los discos a la varilla final colocados de mayor a menor ascendentemente.



Ejemplo con tres discos:



Argumentos que usaremos:
 n, ini, fin, aux



Pasar los discos superiores de A a C:

ini – fin

ini – aux

fin – aux

Hasta ahora:
(n-1, ini, aux, fin)

Pasar el tercer disco de A a B.

ini - fin

Pasar los dos discos superiores de C a B.

aux – ini

aux – fin

ini – fin

Hasta ahora:
(n-1, aux, fin, ini)

Recursivamente tenemos:

1. Pasar $n-1$ discos de A a C.
2. Pasar 1 disco de A a B.
3. Pasar $n-1$ discos de C a B.

Siendo el caso base cuando $n = 0$.

A continuación, resuelva este problema usando *Java*. Su programa debe pedir al usuario un número n y luego desplegar en la consola todos los pasos a seguir para resolver las torres de Hanoi con ese número.

4. Lectura/Escritura de Archivos

14 – Primer Semestre 2005 – Pregunta 2

Por un error se almacenaron las notas de las interrogaciones I1 e I2 en dos archivos separados. Así, el archivo `IIC1102_JAL.txt` contiene las notas de ambas interrogaciones para los alumnos cuyos apellidos van de la **A** a la **L** y el archivo `IIC1102_JMZ.txt` contiene las notas de los alumnos cuyos apellidos van de la **M** a la **Z**. El número de alumnos es **a lo más 200**. Se desea crear un programa que una ambos archivos generando un tercer archivo (`I1I102_Promedio.txt`) que mantenga un formato similar, es decir: nombres de los alumnos, las notas de la I1 e I2 y los promedios respectivos. La estructura de los archivos se muestra a continuación:

```

IIC1102_IAL.txt
Apellidos      5.3    5.5
Apellidos      3.2    3.6
Apellidos      4.8    4.6
123456789012345  123    123

IIC1102_IMZ.txt
Apellidos      5.2    5.0
Apellidos      4.3    4.8
Apellidos      5.6    6.1
123456789012345  123    123

IIC101_Promedio.txt
Apellidos      5.3    5.5    5.4
Apellidos      3.2    3.6    3.4
Apellidos      4.8    4.6    4.7
Apellidos      5.2    5.0    5.1
Apellidos      4.3    4.8    4.55
Apellidos      5.6    6.1    5.85
123456789012345  123    123    1234

```

Las columnas son de tamaño fijo, y no tienen títulos o encabezados:
Columna 1: 15 caracteres; columna 2: 3 caracteres; columna 3: 3 caracteres.

Ud. deberá implementar un programa completo en Java, que lea los archivos correctamente, calcule los promedios y genere el archivo de salida.

Nota: En el archivo de salida, el promedio puede ocupar hasta un tamaño máximo de 4 caracteres (sólo se considerarán 2 decimales). En caso de que el promedio tenga más de 2 decimales, éstos se truncan, no se redondean. Ejemplo: 5.45687 se almacena como 5.45.

Puede utilizar el método `ValueOf` de la clase `String` el cual convierte a `String` un `int`.

```

Ejemplo:    double d = 5.45687;
            String s = String.valueOf(d);

```

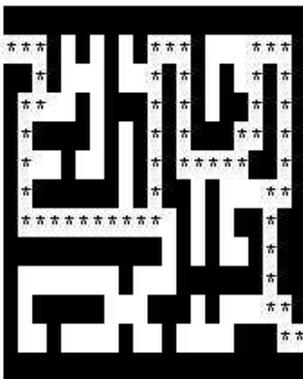
Luego de ejecutar estas instrucciones el `String s` contendrá el valor `"5.45687"`.

5. Resolución de un Laberinto

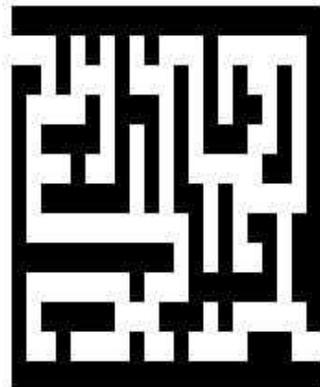
Gentileza Profesor Felipe Csaszar - <http://www.csaszar.org/>

El laberinto a resolver se representa mediante una matriz de números enteros.

Su configuración está dada por la inicialización que se haga a dicha matriz.



Conceptualización: El problema se ubica en un contexto en el que se debe buscar una solución a un laberinto con una entrada y una salida, y con movimientos válidos en cuatro direcciones (no diagonales). En el caso de este ejemplo, el contexto está limitado a una aplicación de computador, en la que el laberinto está representado mediante una matriz de números enteros.



Objetivo: Partiendo de la entrada del laberinto, señalar una ruta que nos guíe hasta la salida.

El caso base de la recursividad (*condición de término*) es que las coordenadas correspondan con la salida del laberinto.

En caso contrario, se procede a marcar la casilla y a intentar en las distintas direcciones (*izquierda, derecha, arriba y abajo*), asegurándose primero que en esa dirección haya una casilla válida (*no ladrillo, no fuera del laberinto y no visitada*).

En caso en que ninguna dirección nos lleve a la salida, se desmarca la casilla actual (pues se va a retroceder) y se devuelve falso como valor de retorno. Las marcas sirven también para señalar la ruta que conforma la solución, una vez alcanzada.

A continuación Ud. deberá completar el método `public static boolean recorre(int fil, int col)` del programa que se presenta en la siguiente página.

```

1  import iic1102Package.*;
2  public class Laberinto {
3
4      /* Constantes que definen la dimensión del laberinto */
5      private static int FILAS = 13;
6      private static int COLUMNAS = 21;
7
8      /* X se usa para las paredes */
9      /* ' ' es un espacio en blanco, para los caminos libres */
10     /* '*' es la marca que señala las posiciones visitadas (camino) */
11     private static char L = 'X';
12     private static char E = ' ';
13     private static char MARCA = '*';
14
15     private static char[][] lab =
16         { { L, L },
17           { E, E, E, L, E, L, E, L, E, L, E, E, E, L, E, E, E, E, E, L },
18           { L, L, E, L, E, E, E, L, E, E, E, L, E, L, E, L, E, E, L, E, L },
19           { L, E, E, E, E, L, E, L, L, L, E, L, E, L, E, L, L, E, L, E, L },
20           { L, E, L, L, L, L, E, L, E, L, E, L, E, L, L, L, E, E, L, E, L },
21           { L, E, E, E, L, E, E, L, E, L, E, L, E, E, E, E, E, L, L, E, L },
22           { L, E, L, L, L, L, L, L, E, L, E, L, L, E, L, E, E, E, E, E, L },
23           { L, E, L, E, L, E, L, L, E, L, L },
24           { L, E, L, E, L, E, E, L, E, L, L },
25           { L, E, E, E, E, E, E, E, L, E, E, E, L, L, L, L, L, L, E, L, L },
26           { L, E, L, L, L, L, L, E, E, E, L, L, L, E, L, E, E, E, E, E, L },
27           { L, E, E, L, E, E, E, E, L, E, E, L, E, E, E, E, L, L, L, E, E },
28           { L, L }
29     };
30
31     private static int x0 = 1;
32     private static int y0 = 0;
33     private static int xf = 11;
34     private static int yf = 20;
35
36     public static void main(String[] args) {
37         boolean ok; //para saber si se obtuvo solución
38         Interfaz.MostrarMensajeConsola("Laberinto:");
39         desplegarLaberinto(lab); //despliega el laberinto sin resolver
40         ok = recorre(x0, y0); /* Resuelve el laberinto desde la entrada: (x0,y0) */
41         if(!ok) //si no se pudo resolver se muestra un mensaje
42             Interfaz.MostrarMensajeConsola("\nLaberinto sin solución.");
43         else {
44             Interfaz.MostrarMensajeConsola("\nLaberinto con Solución:");
45             desplegarLaberinto(lab); //despliega el laberinto resuelto (con el camino)
46         }
47     }
48     public static boolean recorre(int fil, int col) {
49         ...
50     }
51     public static void desplegarLaberinto(char[][] lab) {
52         Interfaz.MostrarMensajeConsola("");
53         for(int i = 0; i < FILAS; ++i) {
54             String texto = "";
55             for(int j = 0; j < COLUMNAS; ++j) texto += lab[i][j] + " ";
56             Interfaz.MostrarMensajeConsola(texto);
57         }
58     }
59     public static boolean valida(int f, int c) {
60
61         //controla si la posición está fuera del laberinto
62         if (f < 0 || f == FILAS || c < 0 || c == COLUMNAS)
63             return false;
64         //controla si la posición ya fue visitada o es muro
65         if (lab[f][c] == MARCA || lab[f][c] == L)
66             return false;
67
68         return true;
69     }
70 }
71 }

```