

Divide y Vencerás

Nombre: Jorge

Apellidos: Martinez Pazos

DNI: 39510046W

Correo: jmartinez5741@alumno.uned.es

Centro Asociado: Vigo (Pontevedra)

Preguntas Teóricas

1) Indica y razona sobre el coste temporal y espacial del algoritmo desarrollado.

COSTE TEMPORAL:

Divide y Vencerás sigue una estructura recursiva en la que cada llamada se divide en dos subproblemas más pequeños de tamaño $n - 1$. Lo que genera una relación de recurrencia:

$$T(n) \rightarrow 2T(n-1) + c$$

Donde $c = 1$ representa el coste constante de mover un disco. Resolviendo esto nos quedamos con:

$$T(n) = (2^n - 1)$$

Por lo que el coste temporal del algoritmo viene siendo $O(2^n)$, es decir, el número de movimientos crece exponencialmente con el número de discos:

Con $n = 3$ se requieren $2^3 - 1 = 7$ movimientos.

Con $n = 4$ se requieren $2^4 - 1 = 15$ movimientos.

COSTE ESPACIAL:

El coste espacial está dominado por la profundidad de la pila de recursión. En cada llamada recursiva, el programa guarda el estado actual hasta completar las llamadas subsiguientes. La profundidad máxima de la pila será igual al número de discos n , ya que cada llamada recursiva procesa un disco menos ($n - 1$) hasta llegar al caso base ($n = 1$).

Por lo que el coste espacial del algoritmo viene siendo $O(n)$, es decir, lineal y significativamente más eficiente que el coste temporal en términos de crecimiento.

2) Explica razonadamente las diferencias existentes frente a una implementación utilizando Vuelta Atrás, en caso de que sea posible. Comenta las diferencias de coste y el tamaño del espacio de búsqueda.

En la implementación con Divide y Vencerás, el problema se va dividiendo en subproblemas más pequeños, resolviendo cada uno de manera independiente antes de combinar los resultados. Y el tamaño de espacio de búsqueda se caracteriza porque no se exploran múltiples posibilidades, ya que la solución es fija y no hay que tomar decisiones.

En caso de una implementación con Vuelta Atrás, se explorarían todas las posibles configuraciones para llegar a la solución, pues al final viene siendo una búsqueda en profundidad con poda.

Debido a esto, el tamaño del espacio de búsqueda es exponencial, 3^n . Siendo 3 el número de postes donde se pueden colocar las piezas. Y el coste temporal y espacial serían mayores que $O(2^n)$ y $O(n)$ respectivamente, en el coste temporal porque se evalúan configuraciones incorrectas antes de encontrar la solución, y en el coste espacial porque se requiere mantener un registro de los estados intermedios para retroceder.

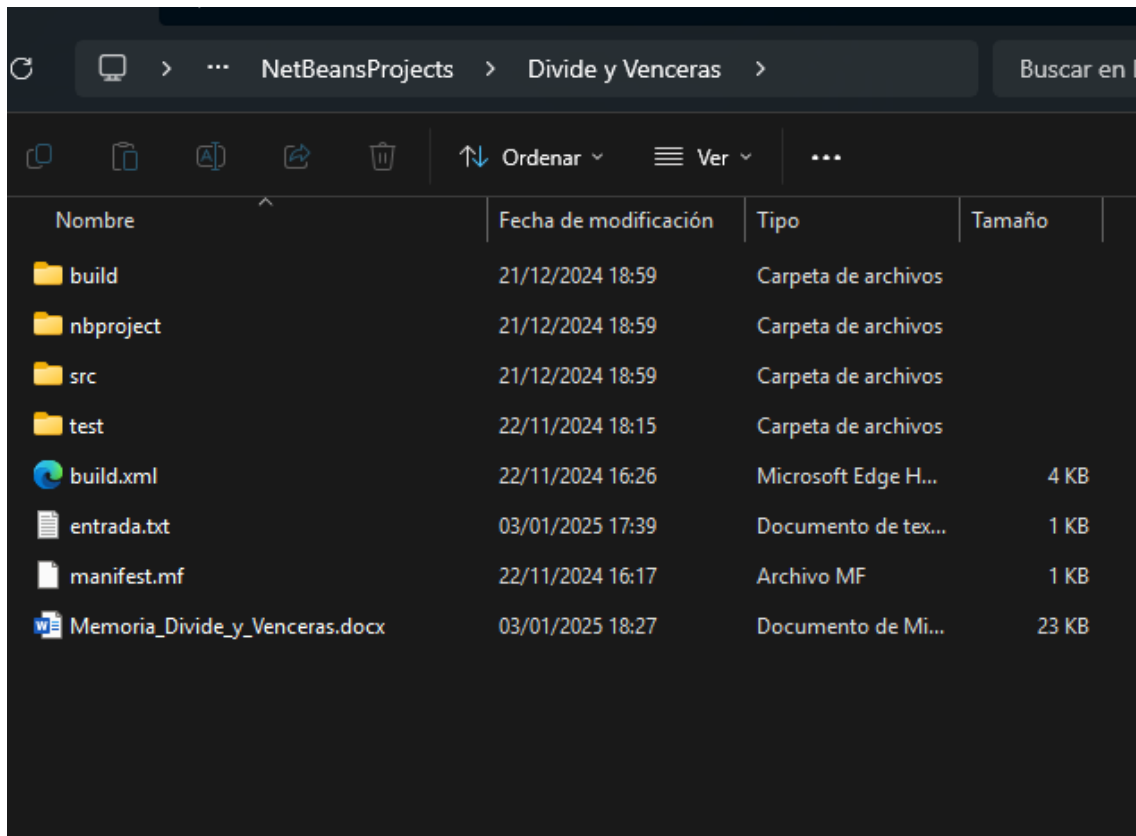
Es decir, las diferencias principales serían que el algoritmo Divide y Vencerás es determinista y eficiente en tiempo y espacio (para este problema), mientras que no es un buen problema para implementar Vuelta Atrás, ya que no hay múltiples caminos viables que explorar, sería ineficiente y redundante.

Ejemplos de Ejecución

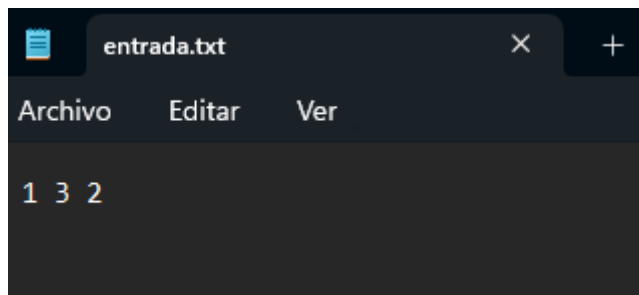
Posibles casos:

- 1) -t entrada.txt salida.txt
- 2) entrada1.txt (inexistente) salida1.txt
- 3) -h entrada1.txt salida1.txt (dan igual la entrada y salida)
- 4) -t
- 5) -x (argumento no valido)
- 6) (vacío, según el pdf debería funcionar el programa)
- 7) -t entrada2.txt

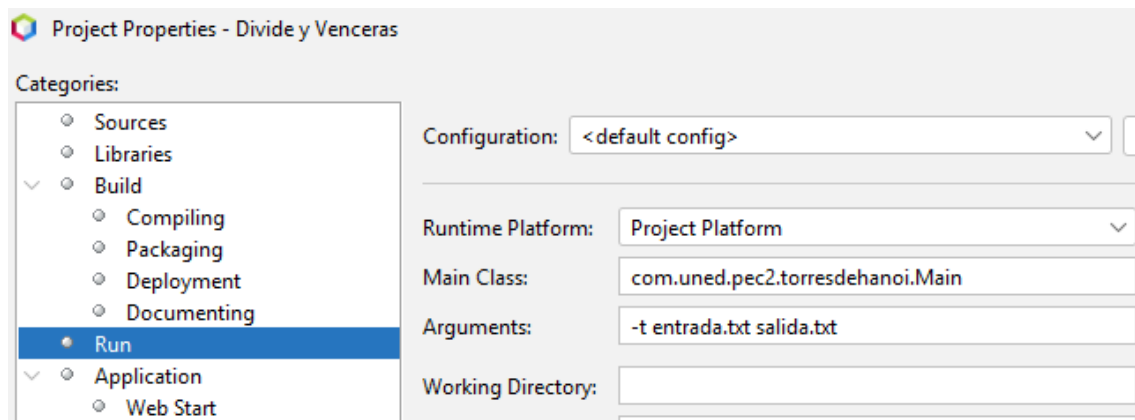
Para empezar, estos son los archivos presentes en mi carpeta del proyecto:



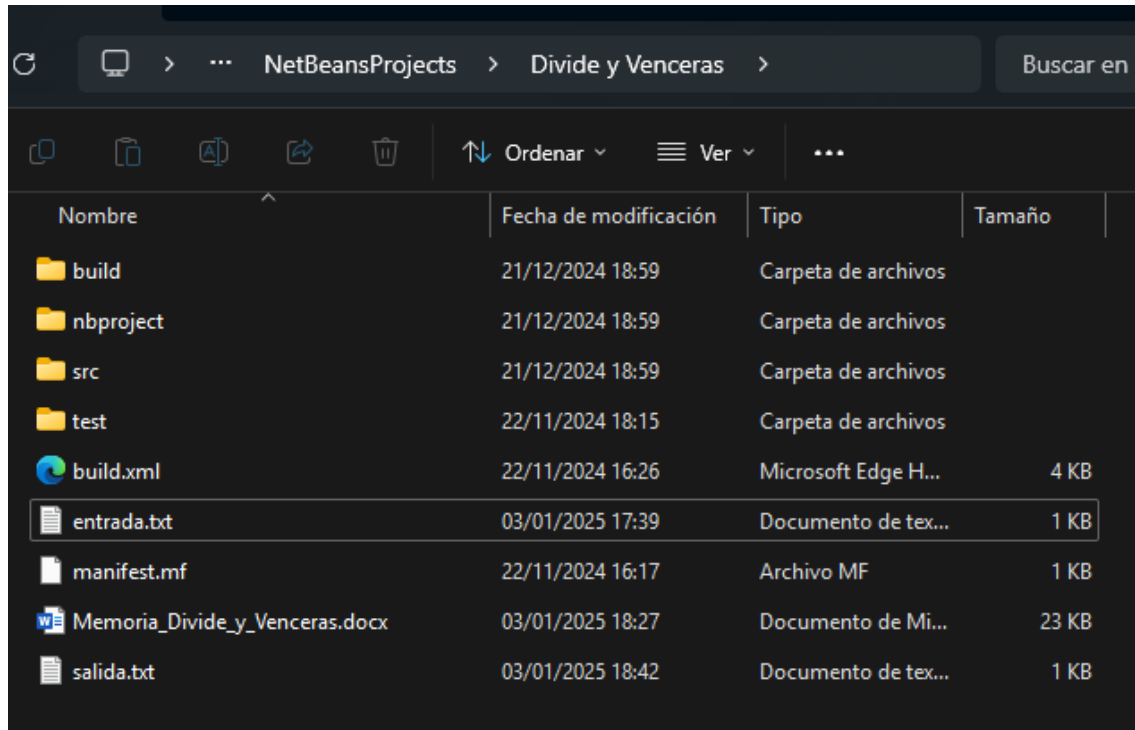
Se puede apreciar el archivo de entrada que usaré para alguna de las pruebas, y cuyo contenido es el siguiente:



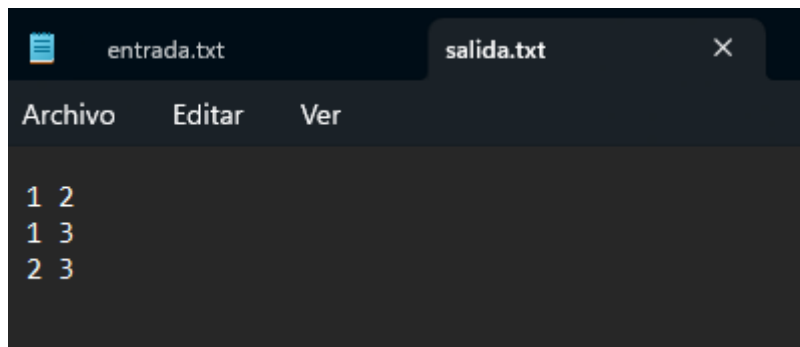
Bien, comenzamos con una prueba básica, aportando como argumentos al ejecutar “entrada.txt salida.txt”, que nos facilita NetBeans:



Por un lado, obtenemos el archivo salida.txt:



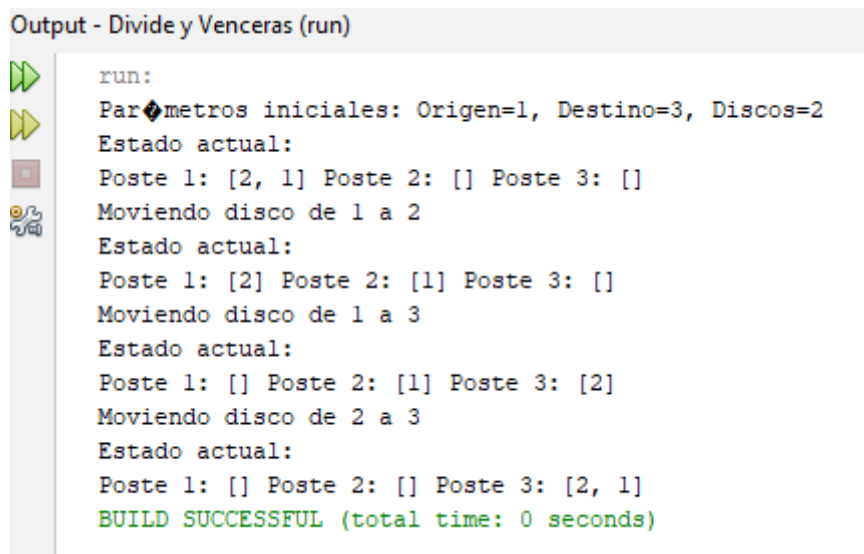
Con el siguiente contenido:



A screenshot of a text editor window with two tabs: 'entrada.txt' and 'salida.txt'. The 'salida.txt' tab is active and shows the following content:

```
1 2
1 3
2 3
```

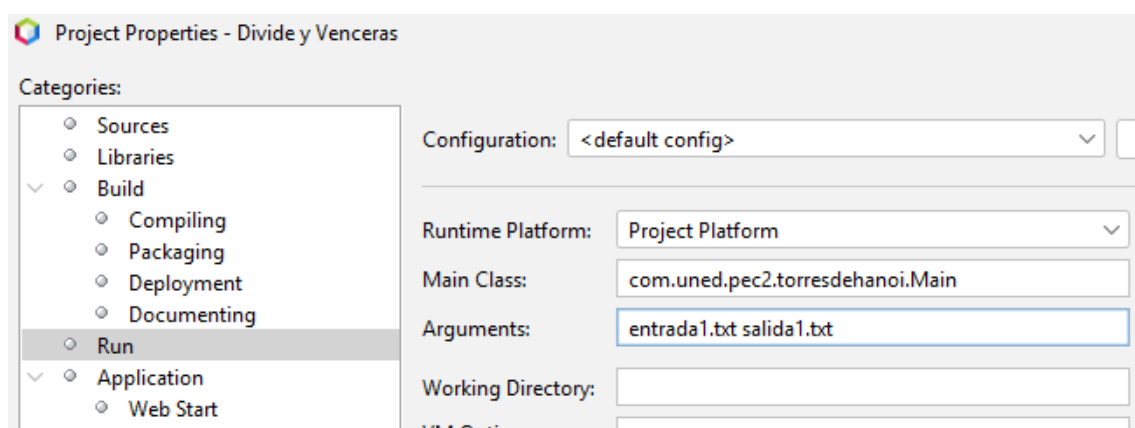
Y por otro lado obtenemos la traza en la consola:



A screenshot of the IDE's output console titled 'Output - Divide y Venceras (run)'. It shows the following execution trace:

```
run:
Parámetros iniciales: Origen=1, Destino=3, Discos=2
Estado actual:
Poste 1: [2, 1] Poste 2: [] Poste 3: []
Moviendo disco de 1 a 2
Estado actual:
Poste 1: [2] Poste 2: [1] Poste 3: []
Moviendo disco de 1 a 3
Estado actual:
Poste 1: [] Poste 2: [1] Poste 3: [2]
Moviendo disco de 2 a 3
Estado actual:
Poste 1: [] Poste 2: [] Poste 3: [2, 1]
BUILD SUCCESSFUL (total time: 0 seconds)
```

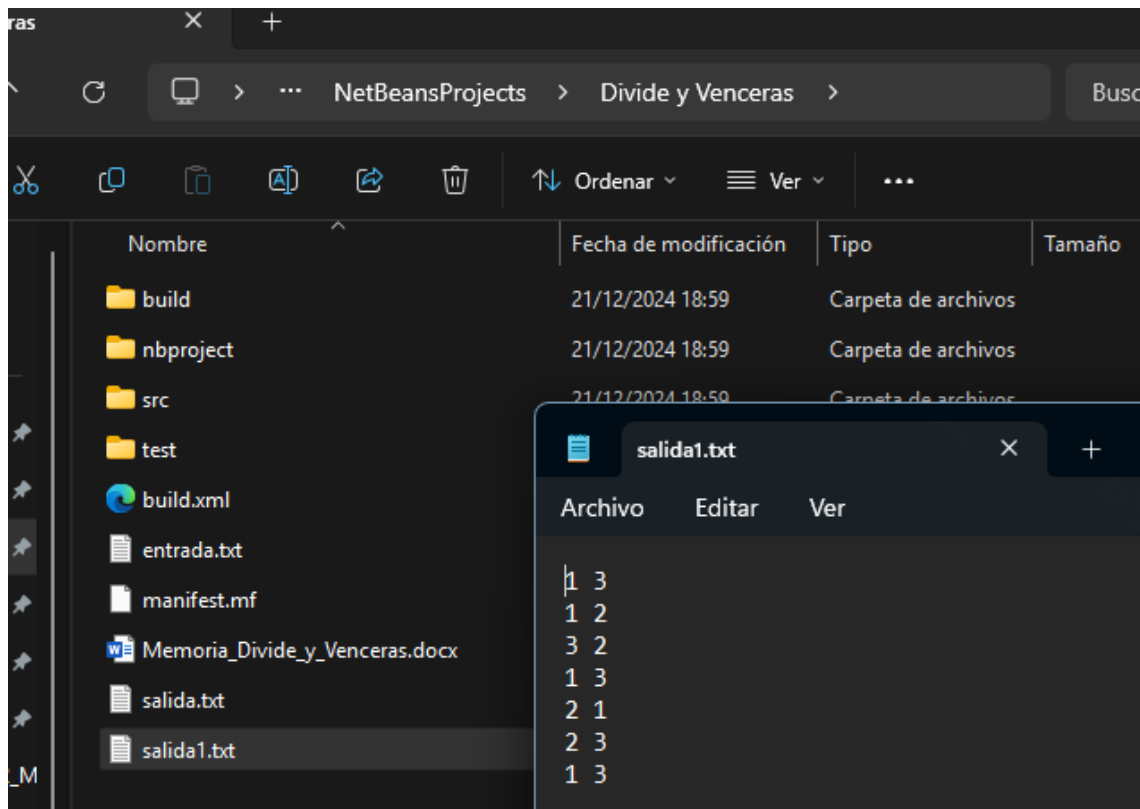
Bien, ahora procedemos a eliminar `-t` de los argumentos y cambiamos los archivos de entrada y salida para realizar otra prueba:



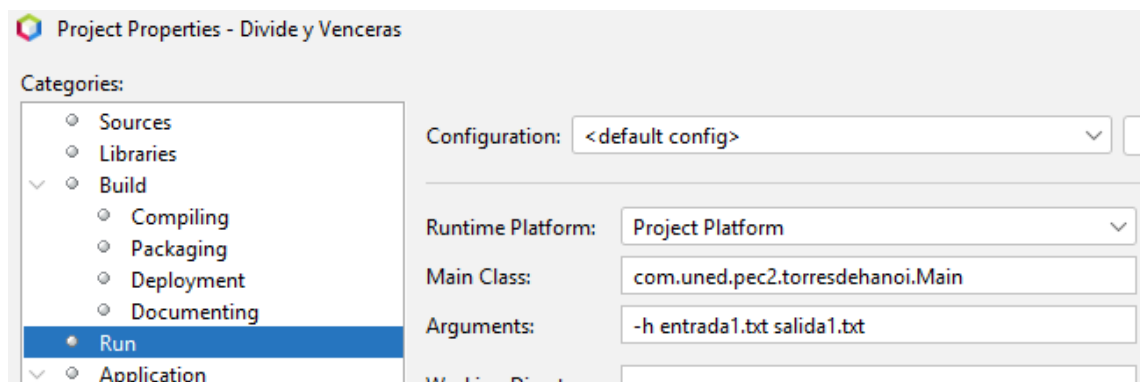
Esta vez ni usaremos la traza ni crearemos el archivo entrada1, para que así nos pida los argumentos por consola:

```
Output - Divide y Venceras (run)

run:
Introduce el poste origen:
1
Introduce el poste destino:
3
Introduce el número de discos:
3
BUILD SUCCESSFUL (total time: 2 seconds)
```



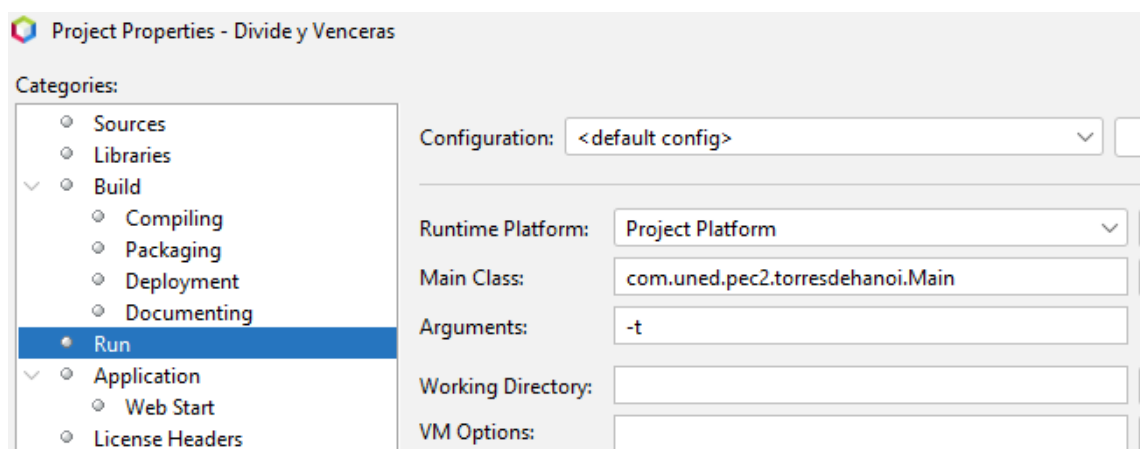
Ahora probaremos el comando de ayuda, y no es necesario cambiar los archivos de entrada o salida porque no se llegará a ejecutar:



```
Output - Divide y Venceras (run)

run:
SINTAXIS: java -jar hanoi.jar [-t] [-h] [fichero entrada] [fichero salida]
-t          Traza el algoritmo
-h          Muestra esta ayuda
[fichero entrada]  Nombre del fichero de entrada
[fichero salida]   Nombre del fichero de salida
BUILD SUCCESSFUL (total time: 0 seconds)
|
```

En caso de no aportar ni un fichero de entrada ni un fichero de salida, se pedirán los argumentos y se mostrará el resultado mediante la consola:

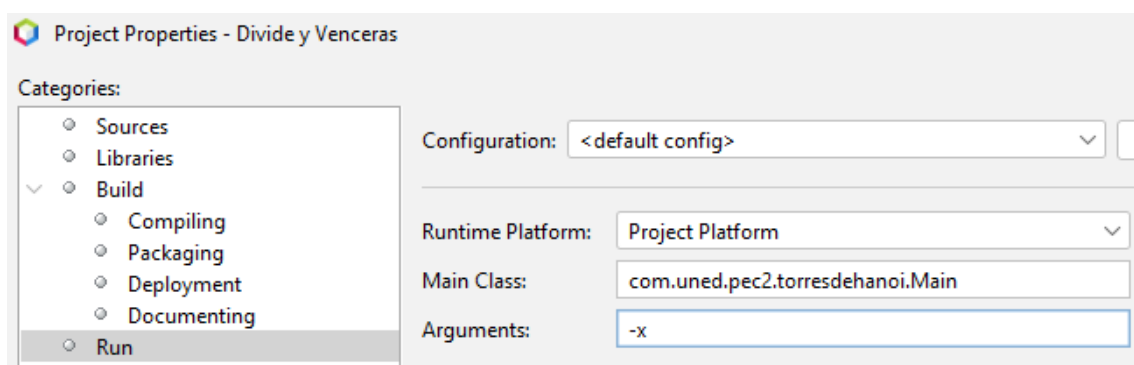



```
Output - Divide y Venceras (run)

run:
Introduce el poste origen:
1
Introduce el poste destino:
3
Introduce el número de discos:
3
Parámetros iniciales: Origen=1, Destino=3, Discos=3
Estado actual:
Poste 1: [3, 2, 1] Poste 2: [] Poste 3: []
Moviendo disco de 1 a 3
Estado actual:
Poste 1: [3, 2] Poste 2: [] Poste 3: [1]
Moviendo disco de 1 a 2
Estado actual:
Poste 1: [3] Poste 2: [2] Poste 3: [1]
Moviendo disco de 3 a 2
Estado actual:
Poste 1: [3] Poste 2: [2, 1] Poste 3: []
Moviendo disco de 1 a 3
Estado actual:
Poste 1: [] Poste 2: [2, 1] Poste 3: [3]
Moviendo disco de 2 a 1
Estado actual:
Poste 1: [1] Poste 2: [2] Poste 3: [3]
Moviendo disco de 2 a 3
Estado actual:
Poste 1: [1] Poste 2: [] Poste 3: [3, 2]
Moviendo disco de 1 a 3
Estado actual:
Poste 1: [] Poste 2: [] Poste 3: [3, 2, 1]
1 3
1 2
3 2
1 3
2 1
2 3
1 3

BUILD SUCCESSFUL (total time: 3 seconds)
```

En el caso de emplear un argumento incorrecto, devolverá un error:



```
Output - Divide y Venceras (run)

run:
Exception in thread "main" java.lang.IllegalArgumentException: Argumento no válido: -x
    at com.uned.pec2.torresdehanoi.Main.main(Main.java:26)
C:\Users\jorge\AppData\Local\NetBeans\Cache\24\executor-snippets\run.xml:111: The following error
C:\Users\jorge\AppData\Local\NetBeans\Cache\24\executor-snippets\run.xml:68: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

¿Y si no aplicamos argumentos? Técnicamente no debería saltar error, porque no hay ningún argumento que sea obligatorio de acuerdo con el enunciado de la práctica:

```
Output - Divide y Venceras (run)

run:
Introduce el poste origen:
1
Introduce el poste destino:
3
Introduce el número de discos:
3
1 3
1 2
3 2
1 3
2 1
2 3
1 3

BUILD SUCCESSFUL (total time: 8 seconds)
```

¿Y si introducimos como argumentos únicamente -t entrada.txt, sin una salida?

Project Properties - Divide y Venceras

Categories:

- Sources
- Libraries
- ▼ ○ Build
 - Compiling
 - Packaging
 - Deployment
 - Documenting
 - Run
- ▼ ○ Application
 - Web Start
 - License Headers

Configuration: <default config>

Runtime Platform: Project Platform

Main Class: com.uned.pec2.torresdehanoi.Main

Arguments: -t entrada2.txt

Working Directory:

VM Options:

entrada2.txt

Archivo Editar Ver

1 3 2

Output - Divide y Venceras (run)

```
run:
Parámetros iniciales: Origen=1, Destino=3, Discos=2
Estado actual:
Poste 1: [2, 1] Poste 2: [] Poste 3: []
Moviendo disco de 1 a 2
Estado actual:
Poste 1: [2] Poste 2: [1] Poste 3: []
Moviendo disco de 1 a 3
Estado actual:
Poste 1: [] Poste 2: [1] Poste 3: [2]
Moviendo disco de 2 a 3
Estado actual:
Poste 1: [] Poste 2: [] Poste 3: [2, 1]
1 2
1 3
2 3

BUILD SUCCESSFUL (total time: 0 seconds)
|
```

Código Fuente

Clase MAIN:

```
/*  
  
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt  
to change this license  
  
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit  
this template  
  
*/  
  
package com.uned.pec2.torresdehanoi;  
  
  
import java.io.*;  
import java.util.Scanner;  
  
  
/**  
  
 * @author jorge  
  
 */  
  
public class Main {  
  
    public static void main(String[] args) {  
  
        boolean traza = false;  
  
        String ficheroEntrada = null;  
  
        String ficheroSalida = null;  
  
  
        try {  
  
            // 1. Validación de los argumentos y asignación de traza  
  
            for (int i = 0; i < args.length; i++) {  
  
                String arg = args[i];  
  
                if (arg.equals("-t")) {
```

```

        traza = true; // Activar traza
    } else if (arg.equals("-h")) {
        mostrarAyuda();
        return;
    } else if (arg.endsWith(".txt")) {
        if (ficheroEntrada == null) {
            ficheroEntrada = arg; // Primer archivo que se encuentra es de entrada
        } else {
            ficheroSalida = arg; // Segundo archivo es de salida (si existe)
        }
    } else {
        throw new IllegalArgumentException("Argumento no válido: " + arg);
    }
}

```

// 2. Leer los parámetros del problema

```

int[] parametros;

if (ficheroEntrada != null && new File(ficheroEntrada).exists()) {
    parametros = leerParametros(ficheroEntrada);
} else if (ficheroEntrada == null) { // Si no hay fichero de entrada, pedir por
consola
    parametros = leerParametrosDesdeConsola();
} else {
    parametros = leerParametrosDesdeConsola(); // Si no existe el archivo,
pedir por consola
}

```

// 3. Resolver el problema

```

        Solucionador solucionador = new Solucionador(parametros[0],
parametros[1], parametros[2], traza);

        String resultado = solucionador.resolver();

        // 4. Escribir resultado en archivo o por consola
        if (ficheroSalida != null) {
            escribirResultado(ficheroSalida, resultado);
        } else {
            System.out.println(resultado); // Si no hay fichero de salida, se imprime por
consola
        }

    } catch (IllegalArgumentException e) {
        System.out.println("Error: " + e.getMessage());
        mostrarAyuda();
    } catch (Exception e) {
        System.out.println("Error inesperado: " + e.getMessage());
    }
}

private static int[] leerParametros(String ficheroEntrada) throws IOException {
    BufferedReader br = new BufferedReader(new FileReader(ficheroEntrada));
    String[] valores = br.readLine().split(" ");
    int origen = Integer.parseInt(valores[0]);
    int destino = Integer.parseInt(valores[1]);
    int numDiscos = Integer.parseInt(valores[2]);
    br.close();
    return new int[]{origen, destino, numDiscos};
}

```

```

private static int[] leerParametrosDesdeConsola() {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Introduce el poste origen:");
    int origen = scanner.nextInt();
    System.out.println("Introduce el poste destino:");
    int destino = scanner.nextInt();
    System.out.println("Introduce el número de discos:");
    int numDiscos = scanner.nextInt();
    return new int[]{origen, destino, numDiscos};
}

```

```

private static void escribirResultado(String ficheroSalida, String resultado)
throws IOException {
    BufferedWriter bw = new BufferedWriter(new FileWriter(ficheroSalida));
    bw.write(resultado);
    bw.close();
}

```

```

private static void mostrarAyuda() {
    System.out.println("SINTAXIS: java -jar hanoi.jar [-t] [-h] [fichero entrada]
[fichero salida]");
    System.out.println("-t          Traza el algoritmo");
    System.out.println("-h          Muestra esta ayuda");
    System.out.println("[fichero entrada]  Nombre del fichero de entrada");
    System.out.println("[fichero salida]  Nombre del fichero de salida");
}
}

```

Clase Solucionador:

```
/*  
  
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt  
to change this license  
  
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit  
this template  
  
*/  
  
package com.uned.pec2.torresdehanoi;  
  
  
import java.util.Stack;  
  
  
/**  
  
 * @author jorge  
  
 */  
  
public class Solucionador {  
  
    private int origen;  
  
    private int destino;  
  
    private int numDiscos;  
  
    private boolean traza;  
  
    private StringBuilder pasos;  
  
    private Stack<Integer>[] postes;  
  
  
    public Solucionador(int origen, int destino, int numDiscos, boolean traza) {  
  
        this.origen = origen;  
  
        this.destino = destino;  
  
        this.numDiscos = numDiscos;  
  
        this.traza = traza;  
  
        this.pasos = new StringBuilder();  
  
        this.postes = new Stack[3]; // Tres postes
```



```

// Inicializamos los postes con discos
for (int i = 0; i < 3; i++) {
    postes[i] = new Stack<>();
}

// Colocamos todos los discos en el poste de origen
for (int i = numDiscos; i > 0; i--) {
    postes[origen - 1].push(i);
}
}

public String resolver() {
    int auxiliar = 6 - origen - destino; // Los postes son 1, 2, 3. Suma 6 = 1+2+3.
    if (traza) {
        System.out.println("Parámetros iniciales: Origen=" + origen + ", Destino=" +
destino + ", Discos=" + numDiscos);
        mostrarEstadoPostes();
    }
    resolverRecursoivo(numDiscos, origen, destino, auxiliar);
    return pasos.toString();
}

private void resolverRecursoivo(int discos, int origen, int destino, int auxiliar) {
    if (discos == 1) {
        registrarPaso(origen, destino);
    } else {
        resolverRecursoivo(discos - 1, origen, auxiliar, destino);
    }
}

```

```
        registrarPaso(origen, destino);  
        resolverRecursivo(discos - 1, auxiliar, destino, origen);  
    }  
}
```

```
private void registrarPaso(int origen, int destino) {  
    // Realizamos el movimiento de un disco  
    int disco = postes[origen - 1].pop();  
    postes[destino - 1].push(disco);  
  
    pasos.append(origen).append(" ").append(destino).append("\n");  
    if (traza) {  
        System.out.println("Moviendo disco de " + origen + " a " + destino);  
        mostrarEstadoPostes(); // Mostrar el estado después de cada movimiento  
    }  
}
```

```
private void mostrarEstadoPostes() {  
    System.out.println("Estado actual:");  
    for (int i = 0; i < 3; i++) {  
        System.out.print("Poste " + (i + 1) + ": " + postes[i] + " ");  
    }  
    System.out.println();  
}  
}
```