Trabajo – Problema de las Jarras

Materia:

Inteligencia Artificial

Carrera:

Facultad de informática - UAS

#Lenguaje de Programación JAVA – Búsqueda en Anchura

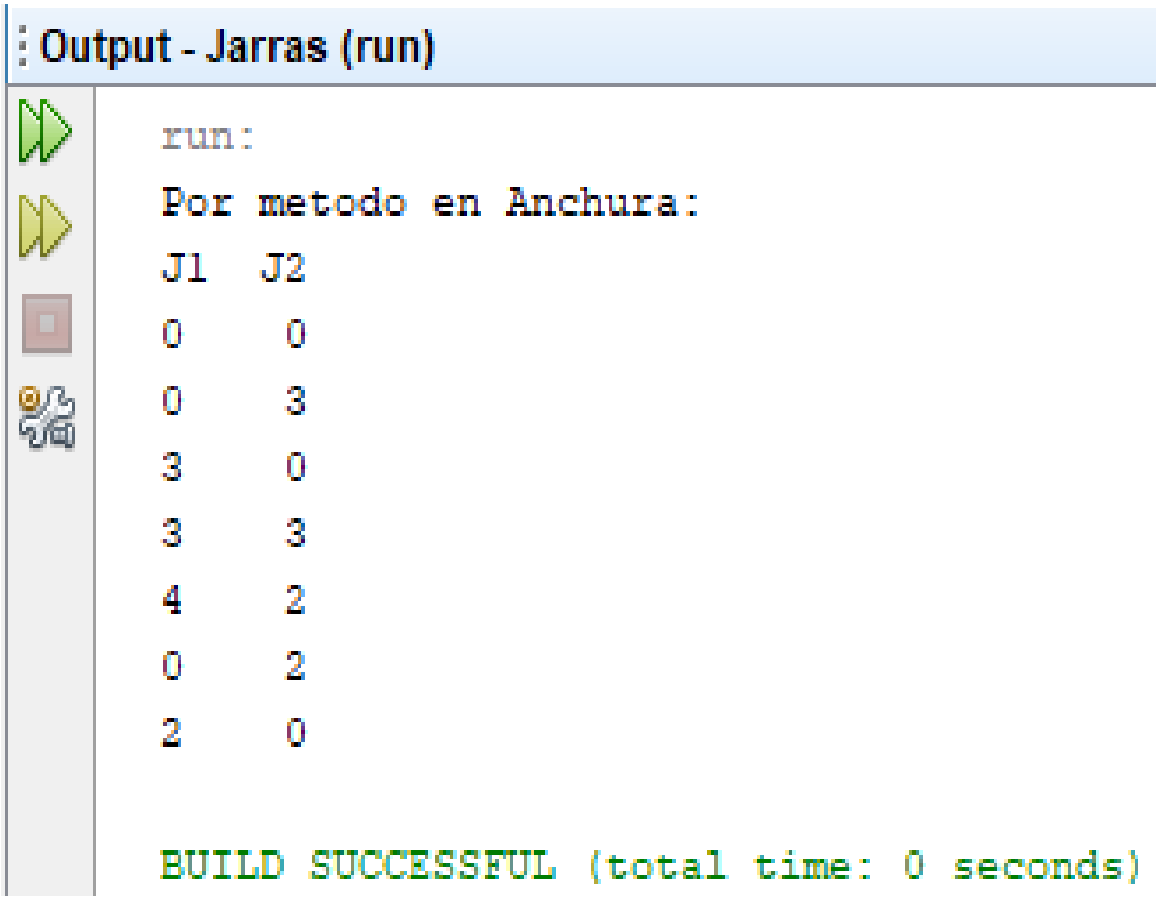
```java
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Stack;

class Evaluar {

    HashSet<Nodos> cerrados;

    void metodoEA() {
        // Cacidad de la Jarra 1
        int jr1 = 4;
        //Capacidad de la Jarra 2
        int jr2 = 3;
        // Valor Final deseado en la Jarra 1
        int valorFinal = 2;

        Nodos estadoInicial = new Nodos(0, 0);
        Nodos estadoFinal = new Nodos(valorFinal, 0);
        Nodos caminoFinal = null;

        cerrados = new HashSet<>();

        LinkedList<Nodos> cola = new LinkedList<>();
        cola.add(estadoInicial);

        while (!cola.isEmpty()) {
            Nodos estadoActual = cola.poll();
            if (estadoActual.equals(estadoFinal)) {
                caminoFinal = estadoActual;
                break;
            }

            // Si x es 0 llenarlo
            if (estadoActual.x == 0) {
                Nodos estadoSiguiente = new Nodos(jr1, estadoActual.y, estadoActual);
                verificarEstadosUnicos(cerrados, estadoSiguiente, cola);
            }

            // Si y es 0 llenarlo
            if (estadoActual.y == 0) {
                Nodos estadoSiguiente = new Nodos(estadoActual.x, jr2, estadoActual);
                verificarEstadosUnicos(cerrados, estadoSiguiente, cola);
            }
```

```
        // Si x no esta vacio, vaciarlo
        if (estadoActual.x > 0) {
            Nodos estadoSiguiente = new Nodos(0, estadoActual.y, estadoActual);
            verificarEstadosUnicos(cerrados, estadoSiguiente, cola);
        }

        // Si y no esta vacia, vaciarlo
        if (estadoActual.y > 0) {
            Nodos estadoSiguiente = new Nodos(estadoActual.x, 0, estadoActual);
            verificarEstadosUnicos(cerrados, estadoSiguiente, cola);
        }

        // tranferir x a y, cuando x no esta vacia y y no esta llena
        if (estadoActual.x > 0 && estadoActual.y < jr2) {
            int CantATransferir = Math.min(estadoActual.x, jr2 - estadoActual.y);
            Nodos estadoSiguiente = new Nodos(estadoActual.x - CantATransferir, estadoActual.y
                    + CantATransferir,
                    estadoActual);
            verificarEstadosUnicos(cerrados, estadoSiguiente, cola);
        }

        // tranferir de y a x, cuando y no esta vacia y x no esta llena
        if (estadoActual.y > 0 && estadoActual.x < jr1) {
            int CantATransferir = Math.min(estadoActual.y, jr1 - estadoActual.x);
            Nodos estadoSiguiente = new Nodos(estadoActual.x + CantATransferir, estadoActual.y
                    - CantATransferir,
                    estadoActual);
            verificarEstadosUnicos(cerrados, estadoSiguiente, cola);
        }
    }

    if (caminoFinal != null) {
        System.out.println("J1  J2");
        System.out.println(caminoFinal);
    } else {
        System.out.println("No es posible");
    }
}

void verificarEstadosUnicos(HashSet<Nodos> cerrados, Nodos estadoEvaluar, LinkedList<Nodos> cola)
{
    if (!cerrados.contains(estadoEvaluar))
    {
```

```
            cerrados.add(estadoEvaluar);
            cola.add(estadoEvaluar);
        }
    }
```

Salida:

```
public static void main(String[] args) {
        System.out.println("Por metodo en Anchura:");
        new Evaluar().metodoEA();
    }
```



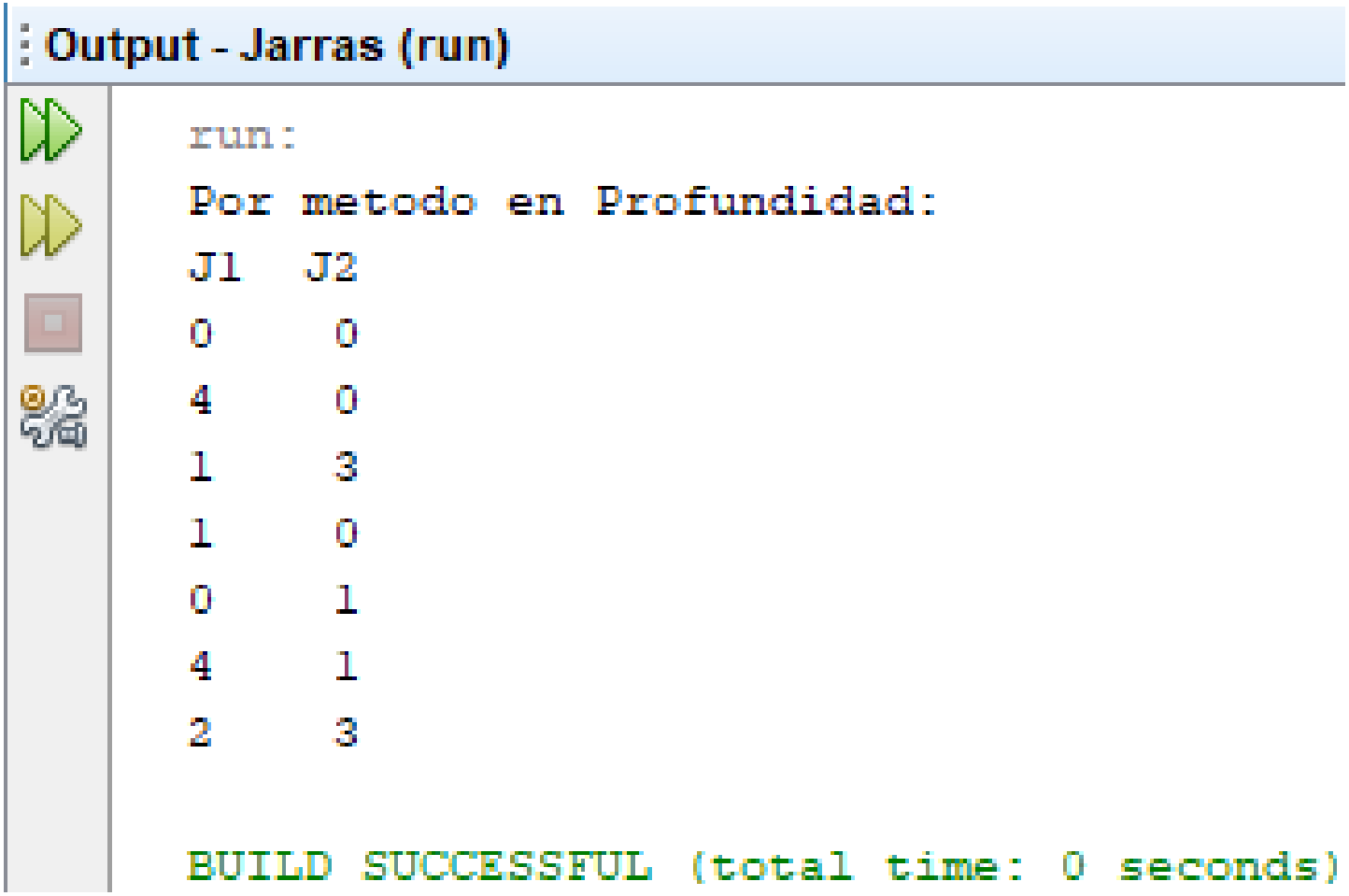
#Lenguaje de Programación JAVA – Búsqueda en Profundidad

```
void metodoPEP() {
        // Cacidad de la Jarra 1
        int jr1 = 4;
        //Capacidad de la Jarra 2
        int jr2 = 3;
        // Valor Final deseado en la Jarra 1
        int valorFinal = 2;

        Nodos estadoInicial = new Nodos(0, 0);
        Nodos estadoFinal = new Nodos(valorFinal, 3);
        Nodos caminoFinal = null;

        cerrados = new HashSet<>();

        Stack<Nodos> pila = new Stack<>();

        pila.push(estadoInicial);

        while (!pila.isEmpty()) {

            Nodos estadoActual = pila.pop();

            cerrados.add(estadoActual);

            if (estadoActual.equals(estadoFinal)) {
                caminoFinal = estadoActual;
                break;
            }

            if (estadoActual.y > 0 && estadoActual.x < jr1) {
                int CantATransferir = Math.min(estadoActual.y, jr1 - estadoActual.x);
                Nodos estadoSiguiente = new Nodos(estadoActual.x + CantATransferir, estadoActual.y
                        - CantATransferir,
                        estadoActual);
                verificarEstadosUnicosEP(cerrados, estadoSiguiente, pila);
            }

            if (estadoActual.y == 0) {
                Nodos estadoSiguiente = new Nodos(estadoActual.x, jr2, estadoActual);
                if (estadoSiguiente.equals(estadoFinal)) {
                    pila.push(estadoSiguiente);
```

```
                estadoActual = pila.pop();
                caminoFinal = estadoActual;
                break;
                }else{
            verificarEstadosUnicosEP(cerrados, estadoSiguiente, pila);}
        }

        if (estadoActual.x == 0) {
            Nodos estadoSiguiente = new Nodos(jr1, estadoActual.y, estadoActual);
            verificarEstadosUnicosEP(cerrados, estadoSiguiente, pila);
        }

        if (estadoActual.y > 0) {
            Nodos estadoSiguiente = new Nodos(estadoActual.x, 0, estadoActual);
            verificarEstadosUnicosEP(cerrados, estadoSiguiente, pila);
        }

        if (estadoActual.x < 0) {
            Nodos estadoSiguiente = new Nodos(0, estadoActual.y, estadoActual);
            verificarEstadosUnicosEP(cerrados, estadoSiguiente, pila);
        }

        if (estadoActual.x > 0 && estadoActual.y < jr2) {
            int CantATransferir = Math.min(estadoActual.x, jr2 - estadoActual.y);
            Nodos estadoSiguiente = new Nodos(estadoActual.x - CantATransferir, estadoActual.y
                    + CantATransferir,
                    estadoActual);
            verificarEstadosUnicosEP(cerrados, estadoSiguiente, pila);
        }
    }

    if (caminoFinal != null) {
        System.out.println("J1  J2");
        System.out.println(caminoFinal);
    } else {
        System.out.println("No es posible");
    }
}

void verificarEstadosUnicosEP(HashSet<Nodos> cerrados, Nodos estadoEvaluar, Stack<Nodos> pila) {
    if (!cerrados.contains(estadoEvaluar) && !pila.contains(estadoEvaluar)) {
        pila.push(estadoEvaluar);
    }
}
```

Salida:

```
public static void main(String[] args) {
        System.out.println("Por metodo en Profundidad:");
        new Evaluar().metodoPEP();
    }
```



Impresión de Padres (Nodos):

```
public String toString() {
        StringBuilder constructor = new StringBuilder();
        if (padre != null) {
            constructor.append(padre);
        }
        constructor.append(x);
        constructor.append("    ").append(y).append("\n");
        return constructor.toString();
    }
```

Clase Nodos:

```
class Nodos {
    //Cantidad en jarra 1 para el estado actual
    int x;
    //Cantidad en jarra 2 para el estado actual
    int y;
    //Padre del estado actual
    Nodos padre;

    public Nodos(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public Nodos(int x, int y, Nodos padre) {
        this.x = x;
        this.y = y;
        this.padre = padre;
    }

    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        Nodos other = (Nodos) obj;
        if (x != other.x) {
```

```
            return false;
        }
        if (y != other.y) {
            return false;
        }
        return true;
    }

    public String toString() {
        StringBuilder constructor = new StringBuilder();
        if (padre != null) {
            constructor.append(padre);
        }
        constructor.append(x);
        constructor.append("    ").append(y).append("\n");
        return constructor.toString();
    }
}
```