Práctica Individual 4

Problema Número 2

Análisis y Diseño de Datos y Algoritmos / Estructura de Datos y Algoritmos

Grado de Software

Curso 2º

Luis Candelario Luna (luisitiluna@hotmail.com)

Tutor: María Luisa Parody Núñez

Índice

1 Enunciado y Cuestiones a resolver

Se dispone de dos jarras inicialmente vacías, Jarra1 y Jarra2, de capacidades en litros concretas. Se desea que las jarras contengan cada una cierta cantidad de litros de agua: **CantidadFinalEnJarra1** y **CantidadFinalEnJarra2**.

Como las jarras no poseen marcas de medida, la única manera de conseguirlo es haciendo trasvases de agua entre las mismas. Las operaciones posibles son:

Operación	Descripción	Código
vaciar J1	Vacía completamente el contenido de J1 al suelo.	0
volcar J1 en J2	Vuelca el contenido de la jarra J1 en la J2. Si ésta se llena, el agua restante de la J1 cae al suelo y desaparece. J1 siempre queda vacía.	1
echar J1 en J2	Echa el contenido de la jarra J1 en la J2. Si ésta se llena, el agua restante de la J1 se conserva en dicha jarra (no cae al suelo).	2
llenar J1	Llena completamente la jarra J1 desde el grifo.	3
llenar J2	Llena completamente la jarra J2 desde el grifo.	4
vaciar J2	Vacía completamente el contenido de J2 al suelo.	5
volcar J2 en J1	Vuelca el contenido de la jarra J2 en la J1. Si ésta se llena, el agua restante de la J2 cae al suelo y desaparece. J2 siempre queda vacía.	6
echar J2 en J1	Echa el contenido de la jarra J2 en la J1. Si ésta se llena, el agua restante de la J2 se conserva en dicha jarra (no cae al suelo).	7

El objetivo es conseguir las cantidades deseadas de agua en cada jarra realizando el **menor número de operaciones posibles**. Tendremos disponible una propiedad que nos indicará el número máximo de operaciones en total que se permiten, **numMaxOp**.

En la resolución del problema se controlará la cantidad de agua que hay en cada jarra en ese momento concreto (**cantidadActualEnJarra1** y **cantidadActualEnJarra2)**. Considere que las jarras están inicialmente vacías.

Tenga en cuenta que, según la cantidad de agua que tiene cada jarra en un momento determinado, habrá disponible diferentes posibilidades, por

ejemplo, si la jarra **J1** está vacía, podremos desde llenar la jarra **J1** hasta echar el contenido de la jarra **J2** en **J1**, pero no tiene sentido vaciar la jarra **J1** porque ya está vacía, o incluso volcar **J1** en **J2**. El número de operaciones realizadas nunca podrá superar el valor **numMaxOp**.

Un posible escenario sería el siguiente: supongamos que las capacidades de **Jarra1** y **Jarra2** son 4 y 3 litros, respectivamente. Deseamos obtener las cantidades de 2 litros de agua en **Jarra1** (es decir, **CantidadFinalEnJarra1** sea 2) y 0 litros en **Jarra2** (es decir, **CantidadFinalEnJarra2** sea 0). La secuencia menor de operaciones necesarias es: (4) llenar J2, (6) volcar J2 en J1, (4) llenar J2, (7) echar J2 en J1, (0) vaciar J1, (6) volcar J2 en J1.

SE PIDE (*) (**):

(1) Resolver el problema por PD, para ello:

a. Complete la ficha por la técnica PD.
b. Complete el proyecto que se le entrega para resolver adecuadamente el problema indicado por PD. Adicionalmente, puede añadir tantas clases, métodos y/o atributos como considere necesarios
c. Complete el test de prueba e indique qué solución obtiene para el problema del escenario indicado previamente en el enunciado (se facilitan los datos en un fichero).
d. El test de prueba debe generar un archivo con extensión ".gv" en el que se almacena el grafo and/or relacionado con la búsqueda llevada a cabo. Conviértalo a un archivo .png, e inclúyalo en la memoria a entregar.

(2) Resolver el problema mediante BT, para ello:

a. Complete la ficha por la técnica BT.
b. Complete el proyecto que se le entrega para resolver adecuadamente el problema indicado por BT sin hacer en principio uso de una **función de cota**. Puede añadir tantas clases, métodos y/o atributos como considere necesarios. Tenga en cuenta que al ser un problema de minimización si no se desea hacer uso de una función de cota, la función relacionada debe devolver MIN_VALUE.
c. Ejecute el test de prueba e indique qué solución obtiene para el problema del escenario indicado previamente en el enunciado (se facilitan los datos en un fichero).
d. Modifique la solución anterior para realizar una **función de cota** adecuada al problema a resolver.

2 Seguimiento

2.1 Control de seguimiento y tutorías

Fecha	Profesor	Detalles y dudas resueltas
16/05/2017	Luisa Parody	- Seguimiento del proyecto y resolución de algunas cuestiones que me impedían obtener el resultado correcto.

2.2 Portafolio

Fecha	Implementación
02/05/2017	- Comprensión del enunciado
04/05/2017	-Realización de los primeros métodos de la clase ProblemaJarraPD.
11/05/2017	-Realización de los métodos restantes de la clase ProblemaJarraPD.
16/05/2017	-Seguimiento del proyecto
19/05/2017	-Realización de todos los métodos de la clase EstadoJarra
24/05/2017	-Preparación del envío y realización memoria

3 Implementación

Resolución del problema mediante PD:

Problema de las Jarras	
Técnica: PD	
Tamaño: numMaxOperaciones-operacionesRealizadas	
Prop. Compartidas:	numMaxOperaciones(Integer) jarra1(Jarra) jarra2(Jarra) operaciones(List<Operación>) cantidadFinalEnJarra1(Integer) cantidadFinalEnJarra2(Integer)
Prop. Individuales:	operacionesRealizadas(Integer) cantidadAcumuladaJarra1(Integer) cantidadAcumuladaJarra2(Integer)
Solución:	Solucion → s(List<Operación>)
Objetivo:	Minimizar el valor de operacionesRealizadas cantidadAcumuladaJarra1=cantidadFinalEnJarra1 && cantidadAcumuladaJarra2=cantidadFinalEnJarra2
Alternativas:	Todas las operaciones posibles .Inicialmente tenemos un problema y dependiendo de alternativa que elijamos el problema se convertirá en otro problema cuyas propiedades distaran de las del problema inicial y pasaran a tener un valor distinto dependiendo de la alternativa que elijamos. En el método implementado se ha utilizado un filtro para descartar acciones que no tenga sentido realizar con el fin de agilizar todo el proceso y llegar a la solución de forma más rápida

Instanciación:	ProblemaJarra(0,0,0)
Problema generalizado:	$pJ(cA1,cA2,oR)$ { Sp(NOP,0) si $cA1==cFEJ1 \&\& cA2==cFEJ2$; $\perp$ si $oR > nMaxOp$; $pJ(cA1+a.incJ1,cA2+a.incJ2,oR+1)$ en otro }

cA1 = cantidadAcumuladaJarra1
cA2 = cantidadAcumuladaJarra2
cFEJ1 = cantidadFinalEnJarra1
cFEJ2 = cantidadFinalEnJarra2
oR = operacionesRealizadas
nMaxOp = numeroMaximoOperaciones
pJ =ProblemaJarra

Clase ProblemaJarraPD:

```
public class ProblemaJarraPD implements ProblemaPD<SolucionJarra,
Accion> {

//Propiedades Individuales

	private Integer cantidadAcumuladaJarra1;
	private Integer cantidadAcumuladaJarra2;
	private int operacionesRealizadas;

//Propiedades Compartidas

	public static Jarra jarra1;
	public static Jarra jarra2;
	public static List<Operacion> operaciones;
	public static Integer numMaxOperaciones;
	private static Integer cantidadFinalEnJarra1;
	private static Integer cantidadFinalEnJarra2;
```

```
	public static ProblemaJarraPD create() {
		return new ProblemaJarraPD(0, 0, 0);
	}

public ProblemaJarraPD(int cantidadAcumuladaJarra1, int
cantidadAcumuladaJarra2, int operacionesRealizadas) {

	operaciones = ProblemaJarra.operaciones;
	jarra1 = ProblemaJarra.jarra1;
	jarra2 = ProblemaJarra.jarra2;
	cantidadFinalEnJarra1 = ProblemaJarra.cantidadFinalEnJarra1;
	cantidadFinalEnJarra2 = ProblemaJarra.cantidadFinalEnJarra2;
	numMaxOperaciones = ProblemaJarra.numMaxOperaciones;
	this.cantidadAcumuladaJarra1 = cantidadAcumuladaJarra1;
	this.cantidadAcumuladaJarra2 = cantidadAcumuladaJarra2;
	this.operacionesRealizadas = operacionesRealizadas;
	}

	public us.lsi.pd.ProblemaPD.Tipo getTipo() {
		return Tipo.Min;
	}

	public int size() {
		return numMaxOperaciones - this.operacionesRealizadas;
	}

	public boolean esCasoBase() {
		Boolean casoBase1 = this.cantidadAcumuladaJarra1 ==
cantidadFinalEnJarra1 && this.cantidadAcumuladaJarra2 ==
cantidadFinalEnJarra2;
		Boolean casoBase2 = numMaxOperaciones <
this.operacionesRealizadas;

		return casoBase1 || casoBase2;
	}

	public Sp<Accion> getSolucionCasoBase() {

		if(this.cantidadAcumuladaJarra1 == cantidadFinalEnJarra1
				&& this.cantidadAcumuladaJarra2 ==
cantidadFinalEnJarra2) {

			return Sp.create(null, 0.);
		}
		return null;
	}

	public Sp<Accion> seleccionaAlternativa(List<Sp<Accion>> ls) {

	Sp<Accion> r = ls.stream().filter(x -> x.propiedad !=
null).min(Comparator.naturalOrder()).orElse(null);

		return r;
```

```
	}

	public ProblemaPD<SolucionJarra, Accion> getSubProblema(Accion
a, int np) {
		Preconditions.checkArgument(np == 0);

	Integer cantidadActualizadaJarra1 = this.cantidadAcumuladaJarra1
+ a.getIncJarra1();
	Integer cantidadActualizadaJarra2 = this.cantidadAcumuladaJarra2
+ a.getIncJarra2();
	Integer numOperaciones = this.operacionesRealizadas + 1;

	return new ProblemaJarraPD(cantidadActualizadaJarra1,
cantidadActualizadaJarra2, numOperaciones);
	}

	public Sp<Accion> combinaSolucionesParciales(Accion a,
List<Sp<Accion>> ls) {
		Double p = ls.get(0).propiedad + 1.;
		return Sp.create(a, p);

	}

	public List<Accion> getAlternativas() {
		List<Accion> alternativas = new LinkedList<Accion>();

		for (Operacion o : operaciones) {

			Integer codigo = o.getCodigo();

			if (codigo == 0 && cantidadAcumuladaJarra1 != 0) {
				Integer incrementoJarra1 = -
this.cantidadAcumuladaJarra1;
				Integer incrementoJarra2 = 0;
				Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
				alternativas.add(a);

			} else if (codigo == 1 &&
this.cantidadAcumuladaJarra1 != 0) {

				Integer incrementoJarra1 = -
this.cantidadAcumuladaJarra1;
				Integer maximoIncremento =
jarra2.getCapacidad() - this.cantidadAcumuladaJarra2;
				Integer incrementoJarra2 = maximoIncremento <
this.cantidadAcumuladaJarra1 ? maximoIncremento
					: this.cantidadAcumuladaJarra1;

				Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
				alternativas.add(a);

			} else if (codigo == 2 &&
this.cantidadAcumuladaJarra1 != 0
```

```
                                && this.cantidadAcumuladaJarra2 !=
jarra2.getCapacidad()) {

                        Integer maximoIncremento =
jarra2.getCapacidad() - this.cantidadAcumuladaJarra2;
                        Integer incrementoJarra2 = (maximoIncremento <
this.cantidadAcumuladaJarra1) ? maximoIncremento
                                        : this.cantidadAcumuladaJarra1;
                        Integer incrementoJarra1 = -incrementoJarra2;

                        Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
                        alternativas.add(a);

                } else if (codigo == 3 &&
this.cantidadAcumuladaJarra1 != jarra1.getCapacidad()) {
                        Integer incrementoJarra2 = 0;
                        Integer incrementoJarra1 =
jarra1.getCapacidad() - this.cantidadAcumuladaJarra1;

                        Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
                        alternativas.add(a);

                } else if (codigo == 4 &&
this.cantidadAcumuladaJarra2 != jarra2.getCapacidad()) {
                        Integer incrementoJarra1 = 0;
                        Integer incrementoJarra2 =
jarra2.getCapacidad() - this.cantidadAcumuladaJarra2;

                        Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
                        alternativas.add(a);

                } else if (codigo == 5 && cantidadAcumuladaJarra2 !=
0) {

                        Integer incrementoJarra2 = -
this.cantidadAcumuladaJarra2;
                        Integer incrementoJarra1 = 0;
                        Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
                        alternativas.add(a);

                } else if (codigo == 6 &&
this.cantidadAcumuladaJarra2 != 0) {

                        Integer incrementoJarra2 = -
this.cantidadAcumuladaJarra2;
                        Integer maximoIncremento =
jarra1.getCapacidad() - this.cantidadAcumuladaJarra1;
                        Integer incrementoJarra1 = maximoIncremento <
this.cantidadAcumuladaJarra2 ? maximoIncremento
                                        : this.cantidadAcumuladaJarra2;
```

```
                        Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
                        alternativas.add(a);

                  } else if (codigo == 7 &&
this.cantidadAcumuladaJarra2 != 0
                              && this.cantidadAcumuladaJarra1 !=
jarra1.getCapacidad()) {

                        Integer maximoIncremento =
jarra1.getCapacidad() - this.cantidadAcumuladaJarra1;
                        Integer incrementoJarra1 = (maximoIncremento <
this.cantidadAcumuladaJarra2) ? maximoIncremento
                                    : this.cantidadAcumuladaJarra2;
                        Integer incrementoJarra2 = -incrementoJarra1;

                        Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
                        alternativas.add(a);
                  }
            }

            return alternativas;
      }

      public int getNumeroSubProblemas(Accion a) {
            return 1;
      }

      public SolucionJarra getSolucionReconstruida(Sp<Accion> sp) {
            List<Operacion> nueva = new LinkedList<Operacion>();
            return SolucionJarra.create(nueva);
      }

      public SolucionJarra getSolucionReconstruida(Sp<Accion> sp,
List<SolucionJarra> ls) {
            SolucionJarra solucion = ls.get(0);
            List<Operacion> noesnueva =
solucion.getListaOperaciones();
            Accion realizada = sp.alternativa;
            noesnueva.add(realizada.getOp());
            return SolucionJarra.create(noesnueva);

      }

      public Double getObjetivoEstimado(Accion a) {
            return Double.MIN_VALUE;
      }

      public Double getObjetivo() {
            return Double.MAX_VALUE;
      }

      public int hashCode() {
            final int prime = 31;
            int result = 1;
```

```
            result = prime * result + ((cantidadAcumuladaJarra1 ==
null) ? 0 : cantidadAcumuladaJarra1.hashCode());
            result = prime * result + ((cantidadAcumuladaJarra2 ==
null) ? 0 : cantidadAcumuladaJarra2.hashCode());
            result = prime * result + operacionesRealizadas;
            return result;
      }

      public boolean equals(Object obj) {
            if (this == obj)
                  return true;
            if (obj == null)
                  return false;
            if (getClass() != obj.getClass())
                  return false;
            ProblemaJarraPD other = (ProblemaJarraPD) obj;
            if (cantidadAcumuladaJarra1 == null) {
                  if (other.cantidadAcumuladaJarra1 != null)
                        return false;
            } else if
(!cantidadAcumuladaJarra1.equals(other.cantidadAcumuladaJarra1))
                  return false;
            if (cantidadAcumuladaJarra2 == null) {
                  if (other.cantidadAcumuladaJarra2 != null)
                        return false;
            } else if
(!cantidadAcumuladaJarra2.equals(other.cantidadAcumuladaJarra2))
                  return false;
            if (operacionesRealizadas != other.operacionesRealizadas)
                  return false;
            return true;
      }

}
```

Resolución del problema mediante BT:

Problema de las Jarras	
Técnica: BT	
Tamaño: numMaxOperaciones-operacionesRealizadas	
Prop. Compartidas:	numMaxOperaciones(Integer) jarra1(Jarra) jarra2(Jarra) operaciones(List<Operación>) cantidadFinalEnJarra1(Integer) cantidadFinalEnJarra2(Integer)
Prop. del Estado:	operacionesRealizadas(Integer) cantidadAcumuladaJarra1(Integer) cantidadAcumuladaJarra2(Integer) lista(solucionJarra)
Solución:	Solucion → s(List<Operación>)
E. Inicial:	ProblemaJarra(0,0,0)
E. Final:	ProblemaJarra(2,0,6)
Objetivo:	Minimizar el valor de operacionesRealizadas cantidadAcumuladaJarra1=cantidadFinalEnJarra1 && cantidadAcumuladaJarra2=cantidadFinalEnJarra2
Alternativas:	Todas las operaciones posibles .Inicialmente tenemos un problema y dependiendo de alternativa que elijamos el problema se convertirá en otro problema cuyas propiedades distaran de las del problema inicial y pasaran a tener un valor distinto dependiendo de la alternativa que elijamos.

	En el método implementado se ha utilizado un filtro para descartar acciones que no tenga sentido realizar con el fin de agilizar todo el proceso y llegar a la solución de forma más rápida.
Función de Cota:	Realizamos la operación que recibimos como parámetro y simulamos las 3 siguientes operaciones que a su vez recorrerán las operaciones que se realizan en cada operación. Si en algún momento se llega al caso base entonces devolveremos 1, 2,3...etc. En el caso de que no lleguemos al caso base devolveremos el MIN_VALUE para que siga iterando hasta que encuentre la mejor solución.

Avanza(a):	cantidadAcumuladaJarra1 += a.getIncJarra1();
	cantidadAcumuladaJarra2 += a.getIncJarra2();
	operacionesRealizadas += 1;
	List<Operacion> copia = Lists.*newArrayList*(lista.getListaOperaciones());
	copia.add(a.getOp());
	lista = SolucionJarra.*create*(copia);

Retrocede(a):	cantidadAcumuladaJarra1 -= a.getIncJarra1();
	cantidadAcumuladaJarra2 -= a.getIncJarra2();
	operacionesRealizadas -= 1;
	List<Operacion> copia = Lists.*newArrayList*(lista.getListaOperaciones());
	copia.remove(lista.getListaOperaciones().size()-1);
	lista = SolucionJarra.*create*(copia);

Clase EstadoJarra:

```
public class EstadoJarra implements EstadoBT<SolucionJarra, Accion> {

	public static Double mejorValorObtenido = Double.MAX_VALUE;
	public static SolucionJarra mejorSolucionObtenida = null;

	public static EstadoJarra create() {
		return new EstadoJarra();
	}

	public static EstadoJarra create(List<Operacion> lista) {
		return new EstadoJarra(0, 0, lista);
```

```
	}

	private SolucionJarra lista;
	private static List<Operacion> operaciones;
	private static int numMaxOperaciones;
	private int operacionesRealizadas;
	private int cantidadAcumuladaJarra1;
	private int cantidadAcumuladaJarra2;
	private static int cantidadFinalEnJarra1;
	private static int cantidadFinalEnJarra2;
	private static Jarra jarra2;
	private static Jarra jarra1;

	private EstadoJarra() {
		super();
		jarra1 = ProblemaJarra.jarra1;
		jarra2 = ProblemaJarra.jarra2;

		cantidadFinalEnJarra1 =
ProblemaJarra.cantidadFinalEnJarra1;
		cantidadFinalEnJarra2 =
ProblemaJarra.cantidadFinalEnJarra2;
		operaciones = ProblemaJarra.operaciones;
		numMaxOperaciones = ProblemaJarra.numMaxOperaciones;

		this.cantidadAcumuladaJarra1 = 0;
		this.cantidadAcumuladaJarra2 = 0;
		this.operacionesRealizadas = 0;
		this.lista = SolucionJarra.create(Lists.newArrayList());

	}

	private EstadoJarra(Integer ac1, Integer ac2, List<Operacion>
lista) {
		super();

		this.cantidadAcumuladaJarra1 = ac1;
		this.cantidadAcumuladaJarra2 = ac2;
		this.operacionesRealizadas = lista.size();
		this.lista = SolucionJarra.create(lista);
	}

	public void avanza(Accion a) {

		this.cantidadAcumuladaJarra1 += a.getIncJarra1();
		this.cantidadAcumuladaJarra2 += a.getIncJarra2();
		this.operacionesRealizadas += 1;
		List<Operacion> copia =
Lists.newArrayList(lista.getListaOperaciones());
		copia.add(a.getOp());
		this.lista = SolucionJarra.create(copia);

	}

	public void retrocede(Accion a) {
```

```
            this.cantidadAcumuladaJarra1 -= a.getIncJarra1();
            this.cantidadAcumuladaJarra2 -= a.getIncJarra2();
            this.operacionesRealizadas -= 1;
            List<Operacion> copia =
Lists.newArrayList(lista.getListaOperaciones());
            copia.remove(lista.getListaOperaciones().size() - 1);
            this.lista = SolucionJarra.create(copia);
      }

      public int size() {
            return numMaxOperaciones - this.operacionesRealizadas;
      }

      public boolean isFinal() {
            Boolean casobase1 = this.cantidadAcumuladaJarra1 ==
cantidadFinalEnJarra1
                        && this.cantidadAcumuladaJarra2 ==
cantidadFinalEnJarra2;
            return casobase1;

      }

      public List<Accion> getAlternativas() {
            List<Accion> alternativas = new LinkedList<Accion>();
            Boolean casobase2 = numMaxOperaciones <=
this.operacionesRealizadas;

            if (casobase2) {
                  return alternativas;
            }

            for (Operacion o : operaciones) {
                  Integer codigo = o.getCodigo();
                  // Vaciar J1
                  if (codigo == 0) {

                        Integer incrementoJarra1 = -
this.cantidadAcumuladaJarra1;
                        Integer incrementoJarra2 = 0;
                        Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
                        alternativas.add(a);

                        // Volcar J1 en J2
                  } else if (codigo == 1) {

                        Integer incrementoJarra1 = -
this.cantidadAcumuladaJarra1;
                        Integer maximoIncremento =
jarra2.getCapacidad() - this.cantidadAcumuladaJarra2;
                        Integer incrementoJarra2 = maximoIncremento <
this.cantidadAcumuladaJarra1 ? maximoIncremento
                                    : this.cantidadAcumuladaJarra1;

                        if (incrementoJarra1 == 0 && incrementoJarra2
== 0)
```

```
                        continue;
                    Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
                    alternativas.add(a);

                    // Echar J1 en J2
                } else if (codigo == 2) {

                    Integer maximoIncremento =
jarra2.getCapacidad() - this.cantidadAcumuladaJarra2;
                    Integer incrementoJarra2 = (maximoIncremento <
this.cantidadAcumuladaJarra1) ? maximoIncremento
                                    : this.cantidadAcumuladaJarra1;
                    Integer incrementoJarra1 = -incrementoJarra2;

                    if (incrementoJarra1 == 0 && incrementoJarra2
== 0)
                        continue;
                    Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
                    alternativas.add(a);

                    // Llenar J1
                } else if (codigo == 3) {

                    Integer incrementoJarra2 = 0;
                    Integer incrementoJarra1 =
jarra1.getCapacidad() - this.cantidadAcumuladaJarra1;

                    if (incrementoJarra1 == 0 && incrementoJarra2
== 0)
                        continue;
                    Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
                    alternativas.add(a);

                    // Llenar J2
                } else if (codigo == 4) {
                    Integer incrementoJarra1 = 0;
                    Integer incrementoJarra2 =
jarra2.getCapacidad() - this.cantidadAcumuladaJarra2;

                    if (incrementoJarra1 == 0 && incrementoJarra2
== 0)
                        continue;
                    Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
                    alternativas.add(a);

                    // Vaciar J2
                } else if (codigo == 5) {

                    Integer incrementoJarra2 = -
this.cantidadAcumuladaJarra2;
                    Integer incrementoJarra1 = 0;
```

```
                        if (incrementoJarra1 == 0 && incrementoJarra2
== 0)
                              continue;
                        Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
                        alternativas.add(a);

                        // Volcar J2 en J1
                  } else if (codigo == 6) {

                        Integer incrementoJarra2 = -
this.cantidadAcumuladaJarra2;
                        Integer maximoIncremento =
jarra1.getCapacidad() - this.cantidadAcumuladaJarra1;
                        Integer incrementoJarra1 = maximoIncremento <
this.cantidadAcumuladaJarra2 ? maximoIncremento
                                    : this.cantidadAcumuladaJarra2;

                        if (incrementoJarra1 == 0 && incrementoJarra2
== 0)
                              continue;
                        Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
                        alternativas.add(a);

                        // Echar J2 en J1
                  } else if (codigo == 7) {

                        Integer maximoIncremento =
jarra1.getCapacidad() - this.cantidadAcumuladaJarra1;
                        Integer incrementoJarra1 = (maximoIncremento <
this.cantidadAcumuladaJarra2) ? maximoIncremento
                                    : this.cantidadAcumuladaJarra2;
                        Integer incrementoJarra2 = -incrementoJarra1;

                        if (incrementoJarra1 == 0 && incrementoJarra2
== 0)
                              continue;
                        Accion a = new Accion(o, incrementoJarra1,
incrementoJarra2);
                        alternativas.add(a);
                  }
            }

            return alternativas;
      }

      public SolucionJarra getSolucion() {
            return this.lista;
      }

      public Double getObjetivo() {
            return (double) operacionesRealizadas;
      }

      public Double getObjetivoEstimado(Accion a) {
```

```
            return Double.MIN_VALUE;
      }
```

Si hacemos uso de la cota debemos implementar los métodos que aparecen a continuación y realizar la siguiente modificación en el método getObjetivoEstimado():

```
public Double getObjetivoEstimado(Accion a) {
            return (double)cotaEstimada(a);
      }

public Integer cotaEstimada(Accion a) {

            Integer copiaCantidadAcumulada1 =
this.cantidadAcumuladaJarra1;
            Integer copiaCantidadAcumulada2 =
this.cantidadAcumuladaJarra2;

            copiaCantidadAcumulada1 += a.getIncJarra1();
            copiaCantidadAcumulada2 += a.getIncJarra2();

            if (copiaCantidadAcumulada1 == cantidadFinalEnJarra1 &&
copiaCantidadAcumulada2 == cantidadFinalEnJarra2) {
                  return 1;
            }

            for (Operacion op1 : operaciones) {
                  for (Operacion op2 : operaciones) {
                        for (Operacion op3 : operaciones) {

                              Map<String, Integer> m1 =
obtieneIncremento(op1, copiaCantidadAcumulada1,
copiaCantidadAcumulada2);
                              copiaCantidadAcumulada1 +=
m1.get("incremento1");
                              copiaCantidadAcumulada2 +=
m1.get("incremento2");

                              if (copiaCantidadAcumulada1 ==
cantidadFinalEnJarra1
                                          && copiaCantidadAcumulada2
== cantidadFinalEnJarra2) {
                                    return 2;
                              }

                              Map<String, Integer> m2 =
obtieneIncremento(op2, copiaCantidadAcumulada1,
copiaCantidadAcumulada2);
                              copiaCantidadAcumulada1 +=
m2.get("incremento1");
                              copiaCantidadAcumulada2 +=
m2.get("incremento2");
```

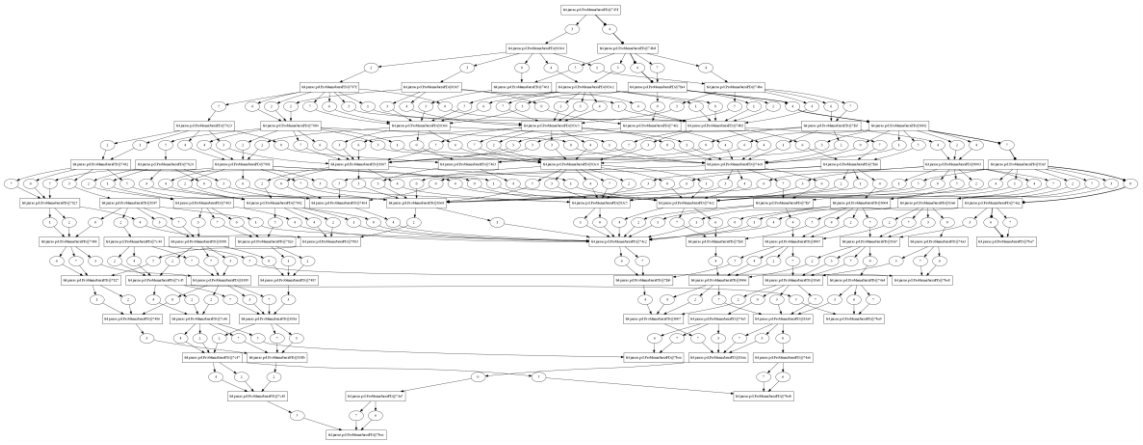
```
                                if (copiaCantidadAcumulada1 ==
cantidadFinalEnJarra1
                                            && copiaCantidadAcumulada2
== cantidadFinalEnJarra2) {
                                      return 3;
                                }

                                Map<String, Integer> m3 =
obtieneIncremento(op3, copiaCantidadAcumulada1,
copiaCantidadAcumulada2);
                                copiaCantidadAcumulada1 +=
m3.get("incremento1");
                                copiaCantidadAcumulada2 +=
m3.get("incremento2");

                                if (copiaCantidadAcumulada1 ==
cantidadFinalEnJarra1
                                            && copiaCantidadAcumulada2
== cantidadFinalEnJarra2) {
                                      return 4;
                                }
                          }
                    }
              }
              return (int) Double.MIN_VALUE;
        }

        private Map<String, Integer> obtieneIncremento(Operacion o,
Integer acumuladoJarra1, Integer acumuladoJarra2) {

              Integer maxIncr = 0;
              Integer incJarra1 = 0;
              Integer incJarra2 = 0;
              Map<String, Integer> mapa = new HashMap<String,
Integer>();

              Integer codigo = o.getCodigo();

              // Vaciar J1
              if (codigo == 0) {
                    incJarra1 = -acumuladoJarra1;
                    incJarra2 = 0;
              }

              // Volcar J1 en J2
              if (codigo == 1) {
                    incJarra1 = -acumuladoJarra1;
                    maxIncr = jarra2.getCapacidad() - acumuladoJarra2;
                    incJarra2 = maxIncr < incJarra1 ? maxIncr :
incJarra1;
              }

              // Echar J1 en J2
              if (codigo == 2) {
                    maxIncr = jarra2.getCapacidad() - acumuladoJarra2;
```

```
                  incJarra2 = (maxIncr < acumuladoJarra1) ? maxIncr :
acumuladoJarra1;
                  incJarra1 = -incJarra2;
            }

            // Llenar J1
            if (codigo == 3) {
                  incJarra1 = jarra1.getCapacidad() - acumuladoJarra1;
                  incJarra2 = 0;
            }

            // Llenar la J2
            if (codigo == 4) {
                  incJarra1 = 0;
                  incJarra2 = jarra2.getCapacidad() - acumuladoJarra2;
            }

            // Vaciar J2
            if (codigo == 5) {
                  incJarra2 = -acumuladoJarra2;
                  incJarra1 = 0;
            }

            // Volcar J2 en J1
            if (codigo == 6) {
                  incJarra2 = -acumuladoJarra2;
                  maxIncr = jarra1.getCapacidad() - acumuladoJarra1;
                  incJarra1 = maxIncr < acumuladoJarra2 ? maxIncr :
acumuladoJarra2;
            }

            // Echar J2 en J1
            if (codigo == 7) {
                  maxIncr = jarra1.getCapacidad() - acumuladoJarra1;
                  incJarra1 = (maxIncr < acumuladoJarra2) ? maxIncr :
acumuladoJarra2;
                  incJarra2 = -incJarra1;
            }

            mapa.put("incremento1", incJarra1);
            mapa.put("incremento2", incJarra2);
            return mapa;
      }
```

4 Test y Pruebas

Resultado obtenido al ejecutar el test que emplea el algoritmo PD:

```
------
Jarra [codigo=1, capacidad=4]
Jarra [codigo=2, capacidad=3]
[Operacion [codigo=0, operacion=vaciar J1, descripcion=Vacía
completamente el contenido de J1 al suelo]
, Operacion [codigo=1, operacion=volcar J1 en J2, descripcion=Vuelca
el contenido de la jarra J1 en la J2. Si ésta se llena el agua
restante de la J1 cae al suelo y desaparece. J1 siempre queda vacía.]
, Operacion [codigo=2, operacion=echar J1 en J2, descripcion=Echa el
contenido de la jarra J1 en la J2. Si ésta se llena el agua restante
de la J1 se conserva en dicha jarra (no cae al suelo).]
, Operacion [codigo=3, operacion=llenar J1, descripcion=Llena
completamente la jarra J1 desde el grifo.]
, Operacion [codigo=4, operacion=llenar J2, descripcion=Llena
completamente la jarra J2 desde el grifo.]
, Operacion [codigo=5, operacion=vaciar J2, descripcion=Vacía
completamente el contenido de J2 al suelo.]
, Operacion [codigo=6, operacion=volcar J2 en J1, descripcion=Vuelca
el contenido de la jarra J2 en la J1. Si ésta se llena el agua
restante de la J2 cae al suelo y desaparece. J2 siempre queda vacía.]
, Operacion [codigo=7, operacion=echar J2 en J1, descripcion=Echa el
contenido de la jarra J2 en la J1. Si ésta se llena el agua restante
de la J2 se conserva en dicha jarra (no cae al suelo).]
]
NumOps: 6.0
Solucion: SolucionJarra [numOp=6, ls=[Operacion [codigo=6,
operacion=volcar J2 en J1, descripcion=Vuelca el contenido de la jarra
J2 en la J1. Si ésta se llena el agua restante de la J2 cae al suelo y
desaparece. J2 siempre queda vacía.]
, Operacion [codigo=0, operacion=vaciar J1, descripcion=Vacía
completamente el contenido de J1 al suelo]
, Operacion [codigo=7, operacion=echar J2 en J1, descripcion=Echa el
contenido de la jarra J2 en la J1. Si ésta se llena el agua restante
de la J2 se conserva en dicha jarra (no cae al suelo).]
, Operacion [codigo=4, operacion=llenar J2, descripcion=Llena
completamente la jarra J2 desde el grifo.]
, Operacion [codigo=6, operacion=volcar J2 en J1, descripcion=Vuelca
el contenido de la jarra J2 en la J1. Si ésta se llena el agua
restante de la J2 cae al suelo y desaparece. J2 siempre queda vacía.]
, Operacion [codigo=4, operacion=llenar J2, descripcion=Llena
completamente la jarra J2 desde el grifo.]
]]
```



Resultado obtenido al ejecutar el test que emplea el algoritmo BT:

```
Jarra [codigo=2, capacidad=3]
[Operacion [codigo=0, operacion=vaciar J1, descripcion=Vacia
completamente el contenido de J1 al suelo]
, Operacion [codigo=1, operacion=volcar J1 en J2, descripcion=Vuelca
el contenido de la jarra J1 en la J2. Si esta se llena el agua
restante de la J1 cae al suelo y desaparece. J1 siempre queda vacia.]
, Operacion [codigo=2, operacion=echar J1 en J2, descripcion=Echa el
contenido de la jarra J1 en la J2. Si esta se llena el agua restante de
la J1 se conserva en dicha jarra (no cae al suelo).]
, Operacion [codigo=3, operacion=llenar J1, descripcion=Llena
completamente la jarra J1 desde el grifo.]
, Operacion [codigo=4, operacion=llenar J2, descripcion=Llena
completamente la jarra J2 desde el grifo.]
, Operacion [codigo=5, operacion=vaciar J2, descripcion=Vacia
completamente el contenido de J2 al suelo.]
, Operacion [codigo=6, operacion=volcar J2 en J1, descripcion=Vuelca
el contenido de la jarra J2 en la J1. Si esta se llena el agua
restante de la J2 cae al suelo y desaparece. J2 siempre queda vacia.]
, Operacion [codigo=7, operacion=echar J2 en J1, descripcion=Echa el
contenido de la jarra J2 en la J1. Si esta se llena el agua restante
de la J2 se conserva en dicha jarra (no cae al suelo).]
]
Solucion: SolucionJarra [numOp=6, ls=[Operacion [codigo=4,
operacion=llenar J2, descripcion=Llena completamente la jarra J2 desde
el grifo.]
, Operacion [codigo=6, operacion=volcar J2 en J1, descripcion=Vuelca
el contenido de la jarra J2 en la J1. Si esta se llena el agua
restante de la J2 cae al suelo y desaparece. J2 siempre queda vacia.]
, Operacion [codigo=4, operacion=llenar J2, descripcion=Llena
completamente la jarra J2 desde el grifo.]
```

, Operacion [codigo=7, operacion=echar J2 en J1, descripcion=Echa el contenido de la jarra J2 en la J1. Si esta se llena el agua restante de la J2 se conserva en dicha jarra (no cae al suelo).]
, Operacion [codigo=0, operacion=vaciar J1, descripcion=Vacia completamente el contenido de J1 al suelo]
, Operacion [codigo=6, operacion=volcar J2 en J1, descripcion=Vuelca el contenido de la jarra J2 en la J1. Si esta se llena el agua restante de la J2 cae al suelo y desaparece. J2 siempre queda vacia.]
]]