

# ADDA - Practica Individual 2

## Problema 3

Juan Arteaga Carmona  
(juan.arteaga41567@gmail.com)  
2º Curso - TI-2

3 de junio de 2018

### 1. Completar la ficha de descripción del problema

- Tipos:
  - S - SolucionAlimentos
  - A - Integer  $\in [0,1000]$  (El numero de gramos que se seleccionan de cada ingrediente)
- Propiedades compartidas:
  - nutrientes - List<Nutriente>- Lista con los ingredientes del problema
  - ingredientesActivos - List<IngredienteActivo>- Lista con los ingredientes activos del problema
- Solucion:

Determinar cuantos gramos de cada ingrediente activo es necesario incluir por kilo de alimento para minimizar costes, teniendo en cuenta que cada kilo de alimento debe contener una cantidad mínima de nutrientes concreta.
- Propiedades:
  - listaIngredientes
  - listaNutrientes
  - minimos
- Restricciones:
  - minimos.stream().allMatch(x ->x<0.)
- Solución óptima:

min coste

### 2. Resolver el problema por PD, para ello:

#### 2.1. Completar la ficha de descripcion de la solución mediante programacion dinámica.

- Problema:
  - Problema de los alimentos

- Descripción:
  - Determinar cuantos gramos de cada ingrediente activo es necesario incluir por kilo de alimento para minimizar costes, teniendo en cuenta que cada kilo de alimento debe contener una cantidad mínima de nutrientes concreta.
- Técnica:
  - Programacion dinámica
- Tipos:
  - S - SolucionAlimentos
  - A - Integer  $\in [0,1000]$  (El numero de gramos que se seleccionan de cada ingrediente)
- Propiedades compartidas:
  - nutrientes - List<Nutriente>- Lista con los ingredientes del problema
  - ingredientesActivos - List<IngredienteActivo>- Lista con los ingredientes activos del problema
  - numeroIngredientes - Derivada - ingredientesActivos.size()
- Propiedades individuales:
  - a - Alternativa elegida
  - index - Integer - Indice indicador del alimento por el que vamos
  - costeIngrediente - Derivada - ingredientesActivos.get(index).getCoste()
- Solución:
  - SolucionAlimentos
- Tamaño:
  - numeroIngredientes - index
- Solucion parcial:
  - (a, c)

Donde: a es la alternativa y c el coste acumulado
- Objetivo:
  - min c
- Alternativas:
  - $A = a \in [1,1000]$
- Instanciación:
  - (0,0)
- ProblemaGeneralizado:
  - (i,j)
- Caso base:
  - $i = \text{numeroIngredientes}$
- Solución casos base:
  - (?, 0.)
- Numero de subproblemas:
  - 1000
- Subproblemas:
  - $p = (i, j) \rightarrow p_1 = (i + 1, j + a * \text{costeIngrediente})$

- Esquema recursivo:
  - $sr(i, coste) = \begin{cases} (?, 0) & i = \text{numeroIngredientes} \\ sA(cS, sp(p_a)) & \text{e.c.o.c} \end{cases}$
- sA:
  - $sA(A, (a', v')) = (A, v')$
- sP:
  - Elegir la alternativa con menor coste.
- Solucion reconstruida:
  - `List<Integer>res = new ArrayList<>();`
  - `memoriaAlternativas.stream().foreach(x ->res.add(x))`
  - `return SolucionAlimentos.create(res);`

## 2.2. Escriba un archivo denominado ‘alimentos.txt’ con los datos del escenario de entrada de forma similar a como se ha realizado en las clases de prácticas para otros problemas.

A continuacion se incluye el archivo con los datos iniciales del problema propuesto en el enunciado de la práctica individual. Cabe destacar que todas las cantidadesMinimas-PorKilo de cada nutriente se escriben todas en una linea y que la cantidadNutrientes y coste se escriben en una linea distinta por cada ingrediente activo.

```
#La cantidad minima de cada nutriente en esta linea#
Nutrientes,90,50,20,2
```

```
#Cada ingrediente activo en una linea distinta#
IngredienteActivo,0.1-0.08-0.04-0.01,0.04
IngredienteActivo,0.2-0.15-0.02-0,0.06
```

## 2.3. Desarrolle un proyecto que resuelva el problema especificado por la técnica sin hacer uso en principio de una función de cota. Dicho proyecto debe incluir un test de prueba que genere la solución para el escenario previamente descrito en el enunciado.

La solución a este problema esta compuesta de la clase ProblemaAlimentosPD2 y TestAlimentosPD.

El código especificado en los anexos, a pesar de estar practicamente implementado, no resuelve este problema, por lo que se puede considerar que no se ha cumplido el objetivo de esta actividad.

- 2.4. El test de prueba debe generar el archivo "GrafoAlimentos.gv"(grafo and/or relacionado de la busqueda llevada a cabo) que debe entregar incluido en el proyecto y en la memoria.**

No se ha podido crear el archivo GrafoAlimentos.gv, ya que no se ha realizado correctamente el ejercicio anterior.

- 2.5. Modifique la solucion anterior para realizar una función de cota adecuada al problema a resolver. Ejecute de nuevo el test de prueba sobre el proyecto modificado e indique qué solución obtiene para el problema del escenario indicado previamente en el enunciado. el test de prueba debe generar el archivo "GrafoAlimentosFiltro.gv"que debe entregar incluido en el proyecto y en la memoria.**

En el codigo fuente existen los métodos necesarios para poder ejecutar el algoritmo con cota. GetObjetivo y getObjetivoEstimado.

Para utilizarlos habria que especificar:

```
AlgoritmoPD.calculaMetricas();  
AlgoritmoPD.isRandomize = true;  
AlgoritmoPD.conFiltro = true;
```

en la clase TestAlimentosPD

### **3. Resolver el problema mediante BT, para ello:**

- 3.1. Completar la ficha de descripción de la solucion mediante BT.**

- Problema:
  - Problema de los alimentos
- Descripción:
  - Determinar cuantos gramos de cada ingrediente activo es necesario incluir por kilo de alimento para minimizar costes, teniendo en cuenta que cada kilo de alimento debe contener una cantidad mínima de nutrientes concreta.
- Técnica:
  - Vuelta Atrás
- Tipos:
  - S - SolucionAlimentos
  - A - Integer  $\in [0,1000]$  (El numero de gramos que se seleccionan de cada ingrediente)
- Propiedades compartidas:
  - nutrientes - List<Nutriente>- Lista con los ingredientes del problema

- ingredientesActivos - List<IngredienteActivo>- Lista con los ingredientes activos del problema
- numeroIngredientes - Derivada - ingredientesActivos.size()
- Propiedades individuales del estado:
  - index - Integer - Indice indicador del alimento por el que vamos
  - memoriaAlternativas - List<Integer>- Lista con las alternativas que hemos seleccionado hasta este estado.
  - minimos - List<Double>- Memoria que se inicializa con los minimos que hay que cumplir y se va restando. Si todo es <0 se han cumplido los minmos.
- Solución:
  - SolucionAlimentos
- Tamaño:
  - numeroIngredientes - index
- Alternativas:
  - $A = a \in [1, 1000]$
- Instanciación:
  - (0,0)
- Estado final:
  - index == numeroIngredientes (Devuelve true si hemos llegado al último ingrediente)
- Avanza:
  - (index, coste) ->(index+1, coste+a\*costeIngrediente)
- Retrocede:
  - (index, coste) ->(index-1, coste-a\*costeIngrediente)
- Solución (para calcular el estado final):
  - $s() = sr(0, null)$
  - $sr(i, coste) = \begin{cases} coste & \text{index} = \text{numeroIngredientes} \\ sr(index + 1, coste + a * costeIngrediente) & \text{index} < \text{numeroIngredientes} \end{cases}$
- Objetivo:
  - Min coste

### 3.2. Desarrolle un proyecto que resuelva el problema especificado por técnica indicada sin hacer uso en principio de una función de cota. Dicho proyecto debe incluir un test de prueba que genere la solución para el escenario previamente descrito en el enunciado.

El código fuente que resuelve este ejercicio se encuentra disponible en el anexo. En concreto, se trataría de el paquete andalu30.PracticaIndividual2.bt. También se puede encontrar un volcado de pantalla de la salida de la consola al ejecutar la clase TestAlimentosBT, en la figura que se encuentra en la página 21.

### 3.3. Modifique la solución anterior para realizar una función de cota adecuada al problema a resolver. Ejecute de nuevo el test de prueba sobre el proyecto modificado e indique qué solución obtiene para el problema del escenario indicado previamente en el enunciado.

Al igual que en el apartado anterior, el código fuente de este ejercicio se encuentra disponible en el anexo. Para ejecutar el algoritmo teniendo en cuenta la cota es necesario especificarlo en la clase TestAlimentosBT con:

```
AlgoritmoBT.conFiltro = true;  
AlgoritmoBT.isRandomize = true;
```

Se puede encontrar el volcado de pantalla de esta solución en la figura de la página 21.

## 4. Anexos

### 4.1. Paquete andalu30.PracticaIndividual2.common

#### 4.1.1. IngredienteActivo.java

```
package andalu30.PracticaIndividual2.common;  
  
import java.util.List;  
  
public class IngredienteActivo {  
    private List<Double> cantidadNutrientes;  
    private Double coste;  
  
    public IngredienteActivo(List<Double> cantidadNutrientes, Double coste) {  
        super();  
        this.cantidadNutrientes = cantidadNutrientes;  
        this.coste = coste;  
    }  
  
    public List<Double> getCantidadNutrientes() {  
        return cantidadNutrientes;  
    }  
  
    public void setCantidadNutrientes(List<Double> cantidadNutrientes) {  
        this.cantidadNutrientes = cantidadNutrientes;  
    }  
  
    public Double getCoste() {  
        return coste;  
    }  
  
    public void setCoste(Double coste) {  
        this.coste = coste;  
    }  
  
    @Override  
    public int hashCode() {  
        final int prime = 31;  
        int result = 1;
```

```

        result = prime * result + ((cantidadNutrientes == null) ? 0 :
        ↪ cantidadNutrientes.hashCode());
        result = prime * result + ((coste == null) ? 0 : coste.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        IngredienteActivo other = (IngredienteActivo) obj;
        if (cantidadNutrientes == null) {
            if (other.cantidadNutrientes != null)
                return false;
        } else if (!cantidadNutrientes.equals(other.cantidadNutrientes))
            return false;
        if (coste == null) {
            if (other.coste != null)
                return false;
        } else if (!coste.equals(other.coste))
            return false;
        return true;
    }

    @Override
    public String toString() {
        return "IngredienteActivo [cantidadNutrientes=" + cantidadNutrientes +
        ↪ ", coste=" + coste + "];"
    }
}

```

#### 4.1.2. Nutriente.java

```

package andalu30.PracticaIndividual2.common;

public class Nutriente {
    private Integer cantidadMinimaPorKilo;

    public Nutriente(Integer cantidadMinimaPorKilo) {
        super();
        this.cantidadMinimaPorKilo = cantidadMinimaPorKilo;
    }

    public Integer getCantidadMinimaPorKilo() {
        return cantidadMinimaPorKilo;
    }

    public void setCantidadMinimaPorKilo(Integer cantidadMinimaPorKilo) {
        this.cantidadMinimaPorKilo = cantidadMinimaPorKilo;
    }
}

```

```

@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((cantidadMinimaPorKilo == null) ? 0 :
        ↪ cantidadMinimaPorKilo.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Nutriente other = (Nutriente) obj;
    if (cantidadMinimaPorKilo == null) {
        if (other.cantidadMinimaPorKilo != null)
            return false;
    } else if (!cantidadMinimaPorKilo.equals(other.cantidadMinimaPorKilo))
        return false;
    return true;
}

@Override
public String toString() {
    return "Nutriente [cantidadMinimaPorKilo=" + cantidadMinimaPorKilo +
        ↪ "]";
}

}

```

#### 4.1.3. ProblemaAlimentos.java

```

package andalu30.PracticaIndividual2.common;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;

public class ProblemaAlimentos {
    private static List<Nutriente> nutrientes;
    private static List<IngredienteActivo> ingredientesActivos;

    public ProblemaAlimentos() {
        super();
    }

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

```



```

}

public static ProblemaAlimentos create(String path) {
    ProblemaAlimentos pa = new ProblemaAlimentos();
    pa.generaDatosIniciales(path);
    return pa;
}

public static List<Nutriente> getNutrientesProblema(){
    return nutrientes;
}

public static List<IngredienteActivo> getIngredientesActivosProblema(){
    return ingredientesActivos;
}

```

```

@SuppressWarnings("static-access")
public void generaDatosIniciales(String path){
    this.nutrientes = generaDatosInicialesNutrientes(path);
    this.ingredientesActivos =
        ↳ generaDatosInicialesIngredientesActivos(path);

    if
        ↳ (nutrientes.size()!=ingredientesActivos.get(0).getCantidadNutrientes().size())
        ↳ {
        System.err.println("Se ha producido un error al cargar los
            ↳ datos iniciales. O faltan/sobran nutrientes o la
            ↳ cantidadNutrientes de los ingredientes no es correcta");
        System.exit(-1);
    }
}

```

```

private static List<Nutriente> generaDatosInicialesNutrientes(String path){
    List<Nutriente> res = new ArrayList<>();

    try {
        File archivo = new File(path);
        FileReader fr = new FileReader(archivo);
        BufferedReader br = new BufferedReader(fr);

        String linea;
        String[] div = null;
        while((linea=br.readLine())!=null) {
            if (linea.startsWith("Nutrientes,")) {
                div = linea.split(",");
            }
        }
    }
}

```

```

        for (int i = 1; i < div.length; i++) {
            Nutriente n = new Nutriente(new
                ↪ Integer(div[i]));
            res.add(n);
        }
    }
    fr.close();

} catch (Exception e) {
    System.err.println("Se ha producido un error al inicializar
        ↪ los datos de los nutrientes: "+e.getMessage());
}
return res;
}

private static List<IngredienteActivo>
    ↪ generaDatosInicialesIngredientesActivos(String path){
    List<IngredienteActivo> res = new ArrayList<>();

    try {
        File archivo = new File(path);
        FileReader fr = new FileReader(archivo);
        BufferedReader br = new BufferedReader(fr);

        String linea;
        String[] div = null;
        while((linea=br.readLine())!=null) {
            if (linea.startsWith("IngredienteActivo,")) {
                div = linea.split(",");
                String[] nut = div[1].split("-");

                List<Double> cantidadNutrientes = new
                    ↪ ArrayList<>();
                for (String s : nut) {
                    cantidadNutrientes.add(new
                        ↪ Double(s));
                }

                IngredienteActivo ia = new
                    ↪ IngredienteActivo(cantidadNutrientes,
                    ↪ new Double(div[2]));
                res.add(ia);
            }
        }
        fr.close();
    } catch (Exception e) {
        System.err.println("Se ha producido un error al
            ↪ inicializar los datos de los ingredientes activos:
            ↪ "+e.getMessage());
    }
    return res;
}

//Metodos equals toString hashCode
@Override

```

```

        public boolean equals(Object obj) {
            return super.equals(obj);
        }

        @Override
        public String toString() {
            return super.toString();
        }

        @Override
        public int hashCode() {
            return super.hashCode();
        }
    }
}

```

## 4.2. Paquete andalu30.PracticaIndividual2.pd

### 4.2.1. ProblemaAlimentosPD2.java

```

package andalu30.PracticaIndividual2.pd;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
import andalu30.PracticaIndividual2.common.IngredienteActivo;
import andalu30.PracticaIndividual2.common.Nutriente;
import andalu30.PracticaIndividual2.common.SolucionAlimentos;
import us.lsi.pd.AlgoritmoPD.Sp;
import us.lsi.pd.ProblemaPD;
import us.lsi.pd.ProblemaPDR;

public class ProblemaAlimentosPD2 implements ProblemaPDR<SolucionAlimentos, Integer>
↳ {

    private int indexingrediente;
    private List<IngredienteActivo> ingredientesActivos;
    private List<Nutriente> nutrientes;
    private List<Double> minimos;
    private List<Integer> memoriaAlternativas;
    private Double costeAcumulado;

    public static ProblemaAlimentosPD2 create(Integer index,
↳ List<IngredienteActivo> ling, List<Nutriente> nuts, List<Double> minimos,
↳ List<Integer> memoriaAlternativas) {
        return new ProblemaAlimentosPD2(index, ling, nuts, minimos,
↳ memoriaAlternativas);
    }
}

```

```

public static ProblemaAlimentosPD2 create(Integer index,
    ↪ List<IngredienteActivo> ling, List<Nutriente> nuts, List<Double> minimos)
    ↪ {
        List<Integer> memoriaAlternativas = new ArrayList<>();
        for (int i = 0; i < ling.size(); i++) {
            memoriaAlternativas.add(0);
        }
        return new ProblemaAlimentosPD2(index, ling, nuts, minimos,
            ↪ memoriaAlternativas);
    }

private ProblemaAlimentosPD2(int indexAlimento, List<IngredienteActivo> ling,
    ↪ List<Nutriente> nuts, List<Double> minimos, List<Integer>
    ↪ memoriaalternativas) {
    this.indexingrediente = indexAlimento;
    this.ingredientesActivos = ling;
    this.nutrientes = nuts;
    this.minimos = minimos;
    this.memoriaAlternativas = memoriaalternativas;
    this.costeAcumulado = 0.;
}

@Override
public Tipo getTipo() {
    return Tipo.Min;
}

@Override
public int size() {
    return this.ingredientesActivos.size()-this.indexingrediente;
}

@Override
public boolean esCasoBase() {
    System.out.println("Caso
        ↪ base:
        ↪ "+(this.ingredientesActivos.size()-this.indexingrediente));
    return this.ingredientesActivos.size()==this.indexingrediente;
}

@Override
public Sp<Integer> getSolucionParcialCasoBase() {
    return Sp.create(null, 0.);
}

@Override
public Sp<Integer> getSolucionParcial(List<Sp<Integer>> ls) {
    Sp<Integer> s = Collections.min(ls);
    return s;
}

@Override
public List<Integer> getAlternativas() {
    List<Integer> ret=IntStream.range(1, 1000)
        .boxed()

```

```

        .collect(Collectors.toList());
    return ret;
}

@Override
public ProblemaPD<SolucionAlimentos, Integer> getSubProblema(Integer a) {

    //Memoria alternativas
    this.memoriaAlternativas.set(indexingrediente, a);
    System.out.println(tl

    return
        ↪ ProblemaAlimentosPD2.create(this.indexingrediente+1,this.ingredientesActivos,thi
}

@Override
public Sp<Integer> getSolucionParcialPorAlternativa(Integer a, Sp<Integer> sp)
    ↪ {
    Sp<Integer> res = Sp.create(a,
        ↪ sp.propiedad+a*this.ingredientesActivos.get(this.indexingrediente).getCoste());
    return res;
}

@Override
public SolucionAlimentos getSolucionReconstruidaCasoBase(Sp<Integer> sp) {
    System.out.println("SolucionReconstruidaCasoBase");
    return SolucionAlimentos.create(new
        ↪ ArrayList<>(),this.ingredientesActivos);
}

@Override
public SolucionAlimentos getSolucionReconstruidaCasoRecursoivo(Sp<Integer> sp,
    ↪ SolucionAlimentos ls) {
    System.out.println("SolucionReconstruidaCasoRecursoivo");

    List<Integer> gramos = this.memoriaAlternativas;
    gramos.set(indexingrediente, sp.alternativa);

    return SolucionAlimentos.create(gramos, this.ingredientesActivos) ;
}

//---HashCode equals toString---

@Override
public int hashCode() {
    final int prime = 31;

```

```

    int result = 1;
    result = prime * result + ((costeAcumulado == null) ? 0 :
        ↪ costeAcumulado.hashCode());
    result = prime * result + indexing ingrediente;
    result = prime * result + ((ingredientesActivos == null) ? 0 :
        ↪ ingredientesActivos.hashCode());
    result = prime * result + ((memoriaAlternativas == null) ? 0 :
        ↪ memoriaAlternativas.hashCode());
    result = prime * result + ((minimos == null) ? 0 :
        ↪ minimos.hashCode());
    result = prime * result + ((nutrientes == null) ? 0 :
        ↪ nutrientes.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    ProblemaAlimentosPD2 other = (ProblemaAlimentosPD2) obj;
    if (costeAcumulado == null) {
        if (other.costeAcumulado != null)
            return false;
    } else if (!costeAcumulado.equals(other.costeAcumulado))
        return false;
    if (indexing ingrediente != other.indexing ingrediente)
        return false;
    if (ingredientesActivos == null) {
        if (other.ingredientesActivos != null)
            return false;
    } else if (!ingredientesActivos.equals(other.ingredientesActivos))
        return false;
    if (memoriaAlternativas == null) {
        if (other.memoriaAlternativas != null)
            return false;
    } else if (!memoriaAlternativas.equals(other.memoriaAlternativas))
        return false;
    if (minimos == null) {
        if (other.minimos != null)
            return false;
    } else if (!minimos.equals(other.minimos))
        return false;
    if (nutrientes == null) {
        if (other.nutrientes != null)
            return false;
    } else if (!nutrientes.equals(other.nutrientes))
        return false;
    return true;
}

@Override
public String toString() {
    return "ProblemaAlimentosPD2 [indexing ingrediente=" + indexing ingrediente +
        ↪ ", ingredientesActivos="

```

```

        + ingredientesActivos + ", nutrientes=" + nutrientes +
        ↪ ", minimos=" + minimos + ", memoriaAlternativas="
        + memoriaAlternativas + ", costeAcumulado=" +
        ↪ costeAcumulado + "];
    }

}

```

#### 4.2.2. TestAlimentosPD.java

```

package andalu30.PracticaIndividual2.pd;

import java.util.List;
import java.util.stream.Collectors;

import andalu30.PracticaIndividual2.common.ProblemaAlimentos;
import andalu30.PracticaIndividual2.common.SolucionAlimentos;
import us.lsi.algoritmos.Algoritmos;
import us.lsi.pd.AlgoritmoPD;

public class TestAlimentosPD {

    @SuppressWarnings("static-access")
    public static void main(String[] args) {

        String path = "ficheros/alimentos.txt";

        ProblemaAlimentos pa = ProblemaAlimentos.create(path);
        System.out.println("Problema Alimentos: "+pa);

        List<Double> minimos = pa.getNutrientesProblema().stream().map(x ->
            ↪ new
            ↪ Double(x.getCantidadMinimaPorKilo())).collect(Collectors.toList());

        ProblemaAlimentosPD2 p = ProblemaAlimentosPD2.create(0,
            ↪ pa.getIngredientesActivosProblema(),
            ↪ pa.getNutrientesProblema(), minimos);
        System.out.println("ProblemaAlimentosPD2: "+p);

        AlgoritmoPD.calculaMetricas();
        AlgoritmoPD.isRandomize = false;
        AlgoritmoPD.conFiltro = false;

        AlgoritmoPD<SolucionAlimentos, Integer> a = Algoritmos.createPD(p);
        System.out.println("Algoritmo PD: "+a+"\n");

        a.ejecuta();

        System.out.println("Algoritmo ejecutado");
        //a.showAllGraph("ficheros/pruebaAlimentosSinFiltro.gv", "Alimentos",
            ↪ p);

        List<Integer> solucionAlgoritmo = a.getSolucion(p).getGramos();
    }
}

```

```

        System.out.println("\n-----\nSolucion
        ↪ algoritmo: "+solucionAlgoritmo);

        System.out.println("\nSolución:");
        for (int i = 0; i < solucionAlgoritmo.size(); i++) {
            System.out.println("Alimento #" + i + ":
            ↪ "+solucionAlgoritmo.get(i) + " gramos.");
        }
        System.out.println("-----");
    }
}

```

## 4.3. Paquete andalu30.PracticaIndividual2.bt

### 4.3.1. ProblemaAlimentosBT.java

```

package andalu30.PracticaIndividual2.bt;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
import java.util.stream.Stream;
import andalu30.PracticaIndividual2.common.IngredienteActivo;
import andalu30.PracticaIndividual2.common.Nutriente;
import andalu30.PracticaIndividual2.common.ProblemaAlimentos;
import us.lsi.bt.EstadoBT;

public class ProblemaAlimentosBT implements EstadoBT<SolucionAlimentos,Integer>{

    private Integer index; //El alimento por el que vamos.

    //Propiedades
    private List<IngredienteActivo> listIngredientesActivos;
    private List<Nutriente> listNutriente;
    private Integer numeroIngredientes; //Derivada
    private List<Integer> memoriaAlternativas; //Lista con las alternativas que
    ↪ hemos cogido (Los gramos).
    private List<Double> minimos; //Memoria que se inicializa con los minimos que
    ↪ hay que cumplir y se va restando. Si todo es <0 se han cumplido los
    ↪ minimos

    //Constructores
    public static ProblemaAlimentosBT create(Integer index,
    ↪ List<IngredienteActivo> listIngredientesActivos,
        List<Nutriente> listNutriente) {
        return new ProblemaAlimentosBT(index,

                                                                 ↪ listIngredientesActivos,
                                                                 ↪ listNutriente,

```



```

        ↪ listNutriente.stream()
        ↪ -> new
        ↪ Double(x.getCantidadM

    }

    public static ProblemaAlimentosBT create(ProblemaAlimentos pa) {
        return new
            ↪ ProblemaAlimentosBT(0,ProblemaAlimentos.getIngredientesActivosProblema(),
                                                    ↪ ProblemaAliment
                                                    ↪ ProblemaAliment
                                                    ↪ ->
                                                    ↪ new
                                                    ↪ Double(x.getCar

    }

    private ProblemaAlimentosBT(Integer index, List<IngredienteActivo>
        ↪ listIngredientesActivos, List<Nutriente> listNutriente, List<Double>
        ↪ minimos) {
        super();
        this.index = index;
        this.listIngredientesActivos = listIngredientesActivos;
        this.listNutriente = listNutriente;
        this.numeroIngredientes = listIngredientesActivos.size();

        this.memoriaAlternativas = new ArrayList<>();
        this.minimos = minimos;
    }

    //Metodos de BT
    @Override
    public ProblemaAlimentosBT getEstadoInicial() {
        return
            ↪ ProblemaAlimentosBT.create(0,this.listIngredientesActivos,this.listNutriente);
    }

    @Override
    public Tipo getTipo() {
        return Tipo.Min;
    }

    @Override
    public int size() {
        return numeroIngredientes-this.index;
    }

    @Override
    public ProblemaAlimentosBT avanza(Integer a) {

        //Añadimos la alternativa a la memoria de las alternativas
        this.memoriaAlternativas.add(a);

        //Reducimos los minimos
        for (int i = 0; i < this.listNutriente.size(); i++) {

```

```

        this.minimos.set(i,
            ↪ this.minimos.get(i)-a*this.listIngredientesActivos.get(index).getCantida
    }

    this.index++;
    return this;
}

@Override
public ProblemaAlimentosBT retrocede(Integer a) {

    //Reducimos el index
    this.index--;

    //Quitar la ultima alternativa añadida.
    this.memoriaAlternativas.remove(this.memoriaAlternativas.size()-1);

    //aumentamos los minimos (nutrientes)
    for (int i = 0; i < this.listNutriente.size(); i++) {
        this.minimos.set(i,
            ↪ this.minimos.get(i)+a*this.listIngredientesActivos.get(index).getCantida
    }

    return this;
}

@Override
public List<Integer> getAlternativas() {
    Stream<Integer> s = IntStream.range(1, 1000).boxed();
    return s.collect(Collectors.toList());
}

@Override
public boolean esCasoBase() {
    return this.index==numeroIngredientes;
}

@Override
public SolucionAlimentos getSolucion() {
    List<Integer> ls = new ArrayList<>();

    for (Integer alt: this.memoriaAlternativas) {
        ls.add(alt);
    }
    //Solucion nula si no se cumplen las caracteristicas
    if (this.minimos.stream().anyMatch(x -> x>0.)) {
        return null;
    }else {
        return SolucionAlimentos.create(ls,
            ↪ this.listIngredientesActivos);
    }
}

@Override
public String toString() {
    return "ProblemaAlimentosBT [index=" + index + ",
        ↪ listIngredientesActivos=" + listIngredientesActivos

```

```

        + ", listNutriente=" + listNutriente + "];
    }

    //Con cota.
    public Double getObjetivoEstimado(Integer a) {
        return
            ↪ this.minimos.stream().min(Comparator.naturalOrder()).orElse(0.)+
            ↪ cota(a);
    }

    public Double cota(Integer a) {
        return new
            ↪ Double(a*this.listIngredientesActivos.get(index).getCoste());
    }
}

}

```

#### 4.3.2. TestAlimentosBT.java

```

package andalu30.PracticaIndividual2.bt;

import java.util.List;

import andalu30.PracticaIndividual2.common.ProblemaAlimentos;
import us.lsi.algoritmos.Algoritmos;
import us.lsi.bt.AlgoritmoBT;

public class TestAlimentosBT {

    @SuppressWarnings("static-access")
    public static void main(String[] args) {
        AlgoritmoBT.calculaMetricas();
        AlgoritmoBT.numeroDeSoluciones = 1;

        AlgoritmoBT.conFiltro = false;
        AlgoritmoBT.isRandomize = false;

        //Creacion de problemas y algoritmo
        System.out.println("Generando problema a partir del archivo
            ↪ 'alimentos.txt'");
        ProblemaAlimentos pa =
            ↪ ProblemaAlimentos.create("./ficheros/alimentos.txt");

        System.out.println("Generando AlgoritmoBT a partir del problema");
        ProblemaAlimentosBT p = ProblemaAlimentosBT.create(pa);
        AlgoritmoBT<SolucionAlimentos, Integer> a = Algoritmos.createBT(p);
    }
}

```

```

        if (AlgoritmoBT.conFiltro==true) {
            System.out.println("Ejecutando el algoritmo con cota: Por
                ↪ favor, espere.");
        }else {
            System.out.println("Ejecutando el algoritmo: Por favor,
                ↪ espere.");
        }
        a.ejecuta();

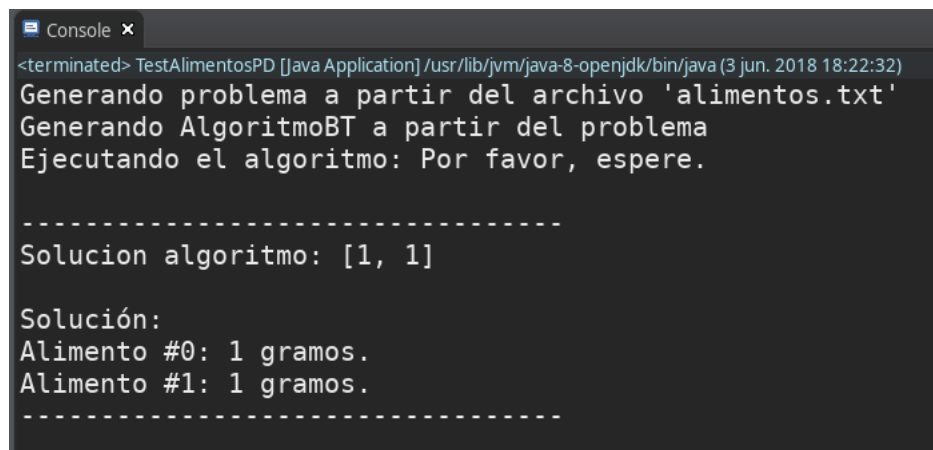
        //Solucion
        List<Integer> solucionAlgoritmo = a.getSolucion().getGramos();
        System.out.println("\n-----\nSolucion
            ↪ algoritmo: "+solucionAlgoritmo);

        System.out.println("\nSolución:");
        for (int i = 0; i < solucionAlgoritmo.size(); i++) {
            System.out.println("Alimento #" + i + ":
                ↪ "+solucionAlgoritmo.get(i) + " gramos.");
        }
        System.out.println("-----");
    }
}

```

#### 4.4. Volcado de pantalla de los resultados obtenidos por cada prueba realizada

##### 4.4.1. PD



```

Console x
<terminated> TestAlimentosPD [Java Application] /usr/lib/jvm/java-8-openjdk/bin/java (3 jun. 2018 18:22:32)
Generando problema a partir del archivo 'alimentos.txt'
Generando AlgoritmoBT a partir del problema
Ejecutando el algoritmo: Por favor, espere.

-----
Solucion algoritmo: [1, 1]

Solución:
Alimento #0: 1 gramos.
Alimento #1: 1 gramos.
-----

```

Figura 1: Solución incorrecta del algoritmo PD

```
Console x
<terminated> TestAlimentosPD [Java Application] /usr/lib/jvm/java-8-openjdk/bin/java (3 jun. 2018 18:22:32)
Generando problema a partir del archivo 'alimentos.txt'
Generando AlgoritmoBT a partir del problema
Ejecutando el algoritmo: Por favor, espere.

-----
Solucion algoritmo: [1, 1]

Solución:
Alimento #0: 1 gramos.
Alimento #1: 1 gramos.
-----
```

Figura 2: Solución incorrecta del algoritmo PD con cota

#### 4.4.2. BT

```
Console x
<terminated> TestAlimentosBT (1) [Java Application] /usr/lib/jvm/java-8-openjdk/bin/java (1 jun. 2018 11:30:29)
Generando problema a partir del archivo 'alimentos.txt'
Generando AlgoritmoBT a partir del problema
Ejecutando el algoritmo: Por favor, espere.

-----
Solucion algoritmo: [368, 266]

Solución:
Alimento #0: 368 gramos.
Alimento #1: 266 gramos.
-----
```

Figura 3: Volcado de pantalla de la terminal al ejecutar la clase TestAlimentosBT

```
Console x
<terminated> TestAlimentosBT (1) [Java Application] /usr/lib/jvm/java-8-openjdk/bin/java (1 jun. 2018 11:27:11)
Generando problema a partir del archivo 'alimentos.txt'
Generando AlgoritmoBT a partir del problema
Ejecutando el algoritmo con cota: Por favor, espere.

-----
Solucion algoritmo: [368, 266]

Solución:
Alimento #0: 368 gramos.
Alimento #1: 266 gramos.
-----
```

Figura 4: Volcado de pantalla de la terminal al ejecutar la clase TestAlimentosBT especificando la opcion de cotas

### 4.5. Código fuente y licencia

Todo el código fuente del trabajo se podrá encontrar en [www.github.com/Andalu30/US-ADDA-PracticaIndividual2/](http://www.github.com/Andalu30/US-ADDA-PracticaIndividual2/) a partir del lunes 4 de junio de 2018, fecha limite de entrega de este trabajo. No se recomienda el uso de la totalidad o de parte de este trabajo en caso de que no se cambie la actividad en años posteriores debido a que la universidad

usa software de detección de copias. Además, este trabajo no está realizado correctamente, la parte de PD no funciona correctamente. El contenido de este trabajo es libre y se encuentra licenciado bajo una licencia GPL v3.

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.