Práctica Individual 4

Problema Número 3

Análisis y Diseño de Datos y Algoritmos
Grado de Ingeniería del Software
Curso 2º

Francisco Javier Huertas Vera (javiihv93@gmail.com)

Tutor: Luis Miguel Soria Morillo

Índice

1 Enunciado

Problema 3 – Problema de la Afinidad

Un centro de belleza necesita asignar un trabajador a cada uno de los clientes a los que tiene que tratar a lo largo del día. Cada cliente llega al centro en una franja horaria (un número entero) concreta y, aunque cualquier trabajador puede atender a cualquier cliente, sólo puede atender a 1 en cada franja horaria y, como mucho podrá atender a 3 clientes a lo largo del día. Además, los clientes han desarrollado una afinidad con algunos trabajadores, por lo que el objetivo es encontrar la asignación de trabajadores a clientes que, cumpliendo las restricciones, haga que exista el mayor número posible de asignaciones afines. Para ello, tendremos acceso en todo momento a la lista de clientes y a la lista de trabajadores. En cada problema individual considere todas las estructuras de datos necesarias para tener en cuenta las restricciones antes mencionadas, por ejemplo, los trabajadores que han sido ya asignados a cada franja horaria y el número de clientes que ya tiene cada trabajador asignado.

1.1 Cuestiones a resolver

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 sin filtro. 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 ejemplo 2 (se facilitan los datos en un fichero). La solución debe contener al menos la asignación con los nombres de los clientes a los nombres de los trabajadores
    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 (la imagen será muy grande, así que deberá reducirla). Además, añada dos recortes de dicha imagen, la primera mostrando el nodo raíz y la segunda mostrando el primer caso base.
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 maximización si no se desea hacer uso de una función de cota, la función relacionada debe devolver MAX_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). La solución debe contener al menos la asignación con los nombres de los clientes a los nombres de los trabajadores.

d. Modifique la solución anterior para realizar una función de cota adecuada al problema a resolver.

e. 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 ejemplo 2 (se facilitan los datos en un fichero). La solución debe contener al menos la asignación con los nombres de los clientes a los nombres de los trabajadores.

1.2 Lenguaje de programación asignado

Lenguaje de programación: Java
Entorno de programación: Eclipse

2 Seguimiento

2.1 Control de seguimiento y tutorías

El alumno debe listar y detallar las visitas a tutoría:

Fecha	Profesor	Detalles y dudas resueltas
25/05/17	Miguel Toro Bonilla	- Problemas con getSubProblemas(), no avanzaba el índice ni la afinidad.
26/05/17	Miguel Toro Bonilla	- Problemas con getSubProblemas() solucionados. Problema de la afinidad terminado.

2.2 Portafolio

Tuve problemas con el getSubProblemas(), a la hora de ejecutar el ejercicio me saltaba una excepción NullPointer, del método showAll() de AlgoritmoPD. Lo solucioné en tutoría con Miguel Toro y el problema estaba en que tenía que guardar las propiedades “índice” y “afinidad” en variables y así modificarlas.

3 Implementación

1. Resolver el problema por PD, para ello:

    a. Complete la ficha por la técnica PD:

Problema de la Afinidad	
Técnica: PD	
Tamaño: n = clientes.size() - indice	
Prop. Compartidas:	Clientes, List<Cliente> Trabajadores, List<String>
Prop. Individuales:	indice, int. nAfinidad, double, valor a maximizar. trabajadoresOcupados, Map<String, Set<Integer>>, map con los nombres de los trabajadores como Keys y las franjas horarias como values. asignación, Map<String, String>, nombre de los clientes como keys y nombre del trabajador asignado como value.
Solución:	Map<String,String>
Objetivo:	Encontrar la mejor solución posible que maximice el valor de nAfinidad.
Alternativas:	Cada trabajador que esté disponible, que no esté asignado ya a un cliente o que no trabaje dos veces en la misma franja horaria.
Instanciación:	
Problema generalizado:	Prob(i,af): (null,0.) si indice = n sA(cS(a, prob(i+1,af+1))) si hay afinidad sA(cS(a, prob(i+1,af))) si no hay afinidad sA: encontrar la alternativa “a” que maximice el valor nAfinidad.
Cs(a,(a’,v))	(a,(a’,v’)+v), sumar la solución de todas las alternativas(sumar las afinidades).

b. Complete el proyecto que se le entrega para resolver adecuadamente el problema indicado por PD sin filtro. Adicionalmente, puede añadir tantas clases, métodos y/o atributos como considere necesarios.

```
public class ProblemaAfinidadPD  implements ProblemaPD<Map<String, String>, Integer> {

    private ProblemaAfinidad problema;

    private int indice;
    private double nAfinidad;
    private Map<String,Set<Integer>> trabajadoresOcupados;
    private Map<String,String> asignacion;

    public static ProblemaAfinidadPD create(int indice, double nAfinidad,
            Map<String,Set<Integer>> trabajadoresOcupados, Map<String,String> asignacion) {
        return new ProblemaAfinidadPD(indice,nAfinidad,trabajadoresOcupados, asignacion);
    }

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

    public ProblemaAfinidadPD(){
        super();
        this.indice = 0;
        this.nAfinidad = 0;
        this.trabajadoresOcupados = new HashMap<String,Set<Integer>>();
        this.asignacion = new HashMap<String,String>();
    }

    public ProblemaAfinidadPD(int indice, double nAfinidad,
            Map<String,Set<Integer>> trabajadoresOcupados, Map<String,String> asignacion){
        super();
        this.indice = indice;
        this.nAfinidad = nAfinidad;
        this.trabajadoresOcupados = trabajadoresOcupados;
        this.asignacion = asignacion;
    }

//Tamaño del problema
public int size(){
    return getClientes().size()-this.indice;
}

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

public boolean esCasoBase(){

    return this.indice==getClientes().size();
}

public Sp<Integer> getSolucionCasoBase(){
    Sp<Integer> sol=Sp.create(null, 0.0);

    return sol;
}
public List<Integer> getAlternativas(){

    List<Integer> alt = new ArrayList<Integer>();
    Map<String,Set<Integer>> copiaTrab = new HashMap<>(trabajadoresOcupados);

    for(int i=0; i<getTrabajadores().size();i++){
        if(!copiaTrab.containsKey(getTrabajadores().get(i))){
            alt.add(i);
        }else{
            if(copiaTrab.get(getTrabajadores().get(i)).size() < 3
                    && !copiaTrab.get(getTrabajadores().get(i)).contains(getClientes().get(this.indice).franjaHoraria)){
                alt.add(i);
            }
        }
    }
    return alt;
}
```

```java
public Sp<Integer> seleccionaAlternativa(List<Sp<Integer>> ls){
    return ls.stream().max(Comparator.naturalOrder()).orElse(null);
}

public ProblemaPD<Map<String,String>,Integer> getSubProblema(Integer a, int i){

    ProblemaPD<Map<String,String>,Integer> p;

    Map<String,Set<Integer>> copia = new HashMap<>(trabajadoresOcupados);
    Map<String,String> copia2 = new HashMap<>(asignacion);
    int in = indice;
    double af = nAfinidad;
    in++;
    af++;
    copia2.put(getClientes().get(indice).nombre, getTrabajadores().get(a));


    copia.put(getTrabajadores().get(a), new HashSet<Integer>());
    copia.get(getTrabajadores().get(a)).add(getClientes().get(indice).franjaHoraria);



    if(getClientes().get(indice).trabajadoresAfines.contains(getTrabajadores().get(a))){

        p = new ProblemaAfinidadPD(in, af,copia, copia2);


    }else{
        p = new ProblemaAfinidadPD(in,nAfinidad,copia, copia2);

    }

    return p;
}

public Sp<Integer> combinaSolucionesParciales(Integer a, List<Sp<Integer>> ls) {
    Sp<Integer> sp = ls.get(0);
    Double num = sp.propiedad;
    if(getClientes().get(this.indice).trabajadoresAfines.contains(getTrabajadores().get(a))){
        num++;
    }

    return Sp.create(a, num);
}

@Override
public int getNumeroSubProblemas(Integer a) {
    // TODO Auto-generated method stub
    return 1;
}

@Override
public Map<String, String> getSolucionReconstruida(Sp<Integer> sp) {
    // TODO Auto-generated method stub
    Map<String, String> res = new HashMap<>();
    return res;
}

@Override
public Map<String, String> getSolucionReconstruida(Sp<Integer> sp, List<Map<String, String>> ls) {
    // TODO Auto-generated method stub
    Map<String,String> res = new HashMap<>(ls.get(0));
    asignacion.keySet().stream().forEach(cliente->res.put(cliente, getTrabajadores().get(sp.alternativa)));
    res.put(getClientes().get(this.indice).nombre, getTrabajadores().get(sp.alternativa));
    return res;
}
```

```
public Double getObjetivoEstimado(Integer a) {
    // TODO Auto-generated method stub
    return Double.MAX_VALUE;
}

@Override
public Double getObjetivo() {
    // TODO Auto-generated method stub
    return Double.MIN_VALUE;
}

private List<Cliente> getClientes(){
    return problema.clientes;
}
private List<String> getTrabajadores(){
    return problema.trabajadores;
}

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

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

@Override
public String toString() {
    return "ProblemaAfinidadPD [indice=" + indice + ", nAfinidad=" + nAfinidad + ", trabajadoresOcupados="
            + trabajadoresOcupados + ", asignacion=" + asignacion + "]";
}
```

c. Complete el test de prueba e indique qué solución obtiene para el problema del escenario indicado previamente en el ejemplo 2 (se facilitan los datos en un fichero). La solución debe contener al menos la asignación con los nombres de los clientes a los nombres de los trabajadores.

A continuación, muestro la salida del TestAfinidadPD, en la que efectivamente la solución es correcta:

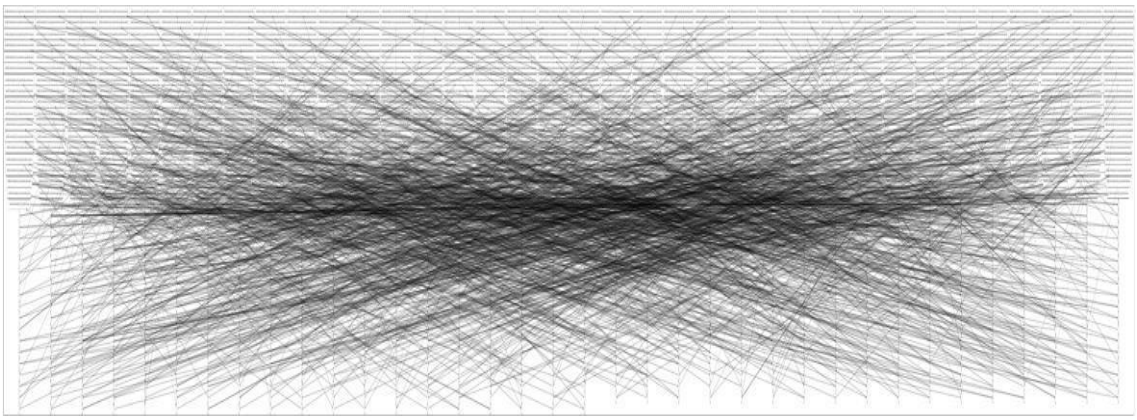
Juan -> Amparo: afinidad =
1 María -> Rosa: afinidad = 2
Sara -> Amparo: afinidad = 3
Andrés -> Rosa: afinidad = 4

Antonio -> Marco: afinidad = 5
Sonia -> Marco: afinidad = 6
Marta -> Rosa: afinidad = 6
Iván -> Amparo: afinidad = 7

```
Afinidad: 7.0
{Andres=Rosa, Marta=Rosa, Ivan=Amparo, Sara=Amparo, Juan=Amparo, Maria=Rosa, Antonio=Marco, Sonia=Marco}
```

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 (la imagen será muy grande, así que deberá reducirla). Además, añada dos recortes de dicha imagen, la primera mostrando el nodo raíz y la segunda mostrando el primer caso base.

A continuación, muestro una foto del grafo. La imagen no se puede apreciar a no ser que se amplíe y podemos ver como mucho más detalle. Adjunto la imagen en la carpeta ficheros del proyecto, junto al archivo de extensión ".gv".



2. Resolver el problema mediante BT, para ello:

a. Complete la ficha por la técnica BT.

Problema de la Afinidad	
Técnica: BT	
Tamaño: clientes.size() – índice	
Prop. Compartidas:	clientes, List<Cliente> trabajadores, List<String<
Prop. Del Estado:	Índice, int afinidad, int trabajadoresOcupados, Map<String,Set<Integer>>. Map de trabajadores trabajando Trabajador(key) – franjaHoraria(value)

	solución, Map<String,String>. Map de asignaciones Cliente(key) – Trabajador(value) cota, Integer. Guarda la afinidad del cliente actual (1 ò 0)
Solución:	Map<String, String>
Estado Inicial:	índice = 0
Estado Final:	índice = clientes.size()
Objetivo:	Encontrar el número mayor de afinidad entre clientes y trabajadores.
Alternativas:	Cada trabajador que esté disponible, que no esté asignado ya a un cliente o que no trabaje dos veces en la misma franja horaria.
Función de Cota:	afinidad + afinidad del cliente actual(1 ò 0) + nº clientes que quedan por visitar.
Avanza(a):	(índice+1,afinidad+1, solucion.put(clientes[indice],trabajadores[a]), trabajadoresOcupados.put(trabajadores[a],clientes[indice].franjaHoraria)) si existe afinidad entre cliente(índice) y trabajador(a) (índice+1,afinidad, solucion.put(clientes[indice],trabajadores[a]), trabajadoresOcupados.put(trabajadores[a],clientes[indice].franjaHoraria)) si **no** existe afinidad entre cliente(índice) y trabajador(a)
Retrocede(a):	(índice+1, afinidad-1, trabajadoresOcupados.remove(trabajadores[a],clientes[indice].franjaHoraria)) si existe afinidad entre cliente(índice) y trabajador(a) (índice+1, afinidad, trabajadoresOcupados.remove(trabajadores[a],clientes[indice].franjaHoraria)) si **no** existe afinidad entre cliente(índice) y trabajador(a)

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 maximización si no se desea hacer uso de una función de cota, la función relacionada debe devolver MAX_VALUE.

```
public class EstadoAfinidadBT implements EstadoBT<Map<String, String>,Integer>{

    List<Cliente> clientes;
    List<String> trabajadores;
    Map<String,Set<Integer>> trabajadoresOcupados;
    Map<String,String> solucion;
    private Integer indice;
    private Integer afinidad;

    public EstadoAfinidadBT(){
        this.clientes = ProblemaAfinidad.clientes;
        this.trabajadores = ProblemaAfinidad.trabajadores;
        this.trabajadoresOcupados = new HashMap<String,Set<Integer>>();
        this.solucion = new HashMap<String,String>();
        this.indice = 0;
        this.afinidad = 0;
    }

    public void avanza(Integer i){
        //añadir trabajador al set del cliente
        if(!trabajadoresOcupados.containsKey(trabajadores.get(i))){
            trabajadoresOcupados.put(trabajadores.get(i), new HashSet<Integer>());
            trabajadoresOcupados.get(trabajadores.get(i)).add(clientes.get(indice).franjaHoraria);
        }else{
            trabajadoresOcupados.get(trabajadores.get(i)).add(clientes.get(indice).franjaHoraria);
        }
        solucion.put(clientes.get(indice).nombre, trabajadores.get(i));
        if(clientes.get(indice).trabajadoresAfines.contains(trabajadores.get(i))){
            afinidad++;
        }
        indice++;
    }


 public void retrocede(Integer i){
     indice--;
     if(trabajadoresOcupados.containsKey(trabajadores.get(i))){
         trabajadoresOcupados.remove(trabajadores.get(i));
     }
     if(clientes.get(indice).trabajadoresAfines.contains(trabajadores.get(i))){
         afinidad--;
     }

 }

 public List<Integer> getAlternativas(){
     List<Integer> alternativas = new ArrayList<Integer>();

     for(int i=0; i<trabajadores.size();i++){

         if(trabajadoresOcupados.containsKey(trabajadores.get(i))
             && !trabajadoresOcupados.get(trabajadores.get(i)).contains(clientes.get(indice).franjaHoraria)){

             alternativas.add(i);

         }else{
             if(!trabajadoresOcupados.containsKey(trabajadores.get(i))){
                 alternativas.add(i);
             }
         }
     }
     return alternativas;
 }
```

```
public Map<String,String> getSolucion(){
    Map<String,String> res = new HashMap<String,String>(solucion);


    return res;
}

public boolean isFinal(){
    return indice == clientes.size();
}

public int size(){
    return clientes.size() - indice;
}

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

public Double getObjetivoEstimado(Integer a){
    return Double.MAX_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). La solución debe contener al menos la asignación con los nombres de los clientes a los nombres de los trabajadores.

A continuación, muestro los resultados obtenidos al ejecutar el test:

```
7.0 ,{Andres=Rosa, Marta=Rosa, Ivan=Amparo, Sara=Amparo, Juan=Amparo, Maria=Rosa, Antonio=Marco, Sonia=Marco}
```

d. Modifique la solución anterior para realizar una función de cota adecuada al problema a resolver.

```
public Integer getFuncionCota(){
       return size()+afinidad+cota;
}
```

e. 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 ejemplo 2 (se facilitan los datos en un fichero). La solución debe contener al menos la asignación con los nombres de los clientes a los nombres de los trabajadores.

A continuación, muestro los resultados obtenidos, con la función de cota, al ejecutar el test:

```
7.0 ,{Andres=Rosa, Marta=Rosa, Ivan=Amparo, Sara=Amparo, Juan=Amparo, Maria=Rosa, Antonio=Marco, Sonia=Marco}
```