Ejercicio 4 – Programación Dinámica

Un proyecto software tiene *nProp* propiedades que es posible favorecer. En general, favorecer una propiedad puede contribuir negativamente a que ocurran ciertos riesgos del proyecto, pero puede contribuir positivamente a que ocurran otros. De este modo, la contribución de una propiedad *p* a un riesgo *r* puede ser 1, –1, o cero, indicando en este último caso que *p* no afecta a *r*.

Se desea aplicar la técnica de *Programación Dinámica* para hallar las propiedades que hay que favorecer de la lista *propiedades* de manera que se **minimice la contribución total** de dichas propiedades a los riesgos dados (lista *riesgos*), teniendo en cuenta que el máximo número de propiedades que se pueden favorecer es *maxPropAFavorecer* y que la contribución total de las propiedades seleccionadas a cada riesgo no puede exceder un determinado valor (*sumaMaxPorRiesgo*).

Ejemplo:

Propiedades	R0	R1	R2	R3	R4
ConocimientosEquipo	–1	1	–1	–1	0
Plazo	–1	1	0	0	–1
Escalabilidad	1	1	1	1	0
Seguridad	0	1	0	0	–1
RecursosParaCalidad	–1	0	–1	–1	0
Complejidad	0	1	0	0	1
Fiabilidad	1	1	0	–1	–1
Interoperabilidad	0	0	1	0	–1

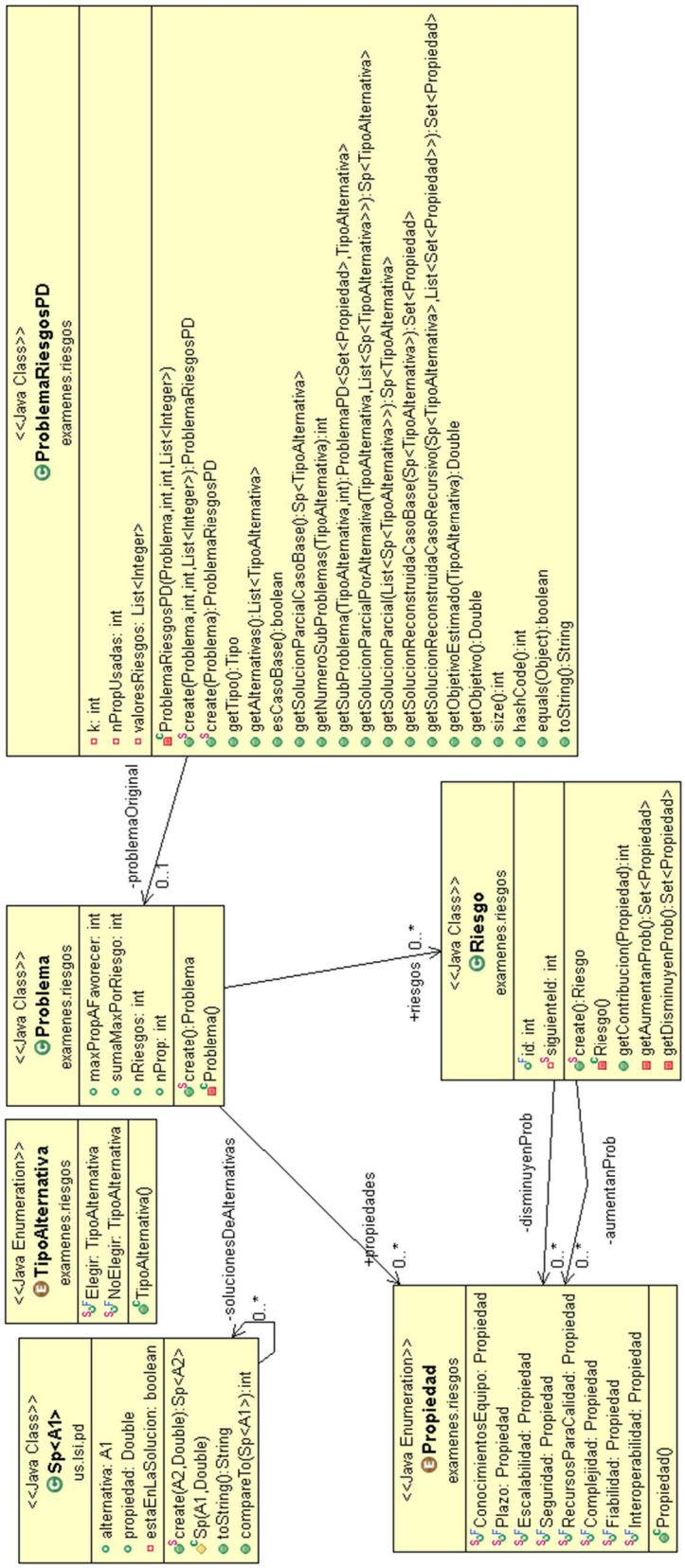
En la tabla de la izquierda, favorecer la propiedad "conocimientos del equipo de desarrollo" (*ConocimientosEquipo*) contribuye negativamente al riesgo 0 pero contribuye positivamente al riesgo 1.
Consideremos *maxPropAFavorecer*=4 y *sumaMaxPorRiesgo*=2. En tal caso, una solución óptima es favorecer las *ConocimientosEquipo, Plazo* y *RecursosParaCalidad*. La contribución total de dichas propiedades a los riesgos es –6, pues *ConocimientosEquipo* contribuye a los riesgos con -1+1-1-1+0=-2, *Plazo* contribuye con -1+1+0+0-1=-1 y *RecursosParaCalidad* contribuye con -1+0-1-1+0=-3. Además, la solución cumple la restricción de *sumaMaxPorRiesgo* ya que, conjuntamente, las tres propiedades seleccionadas contribuyen al riesgo 0 con -3, al riesgo 1 con 2, al riesgo 2 con -2, al riesgo 3 con -2, y al riesgo 4 con -1.

Se pide:

a) Complete los TODO de la ficha.
b) Implemente los métodos **getSolucionParcialCasoBase**, **getSolucionParcialPorAlternativa** y **getSolucionReconstruidaCasoRecursivo**.

Nota 1: no se pueden añadir propiedades individuales a la ficha distintas de las ya indicadas, ni atributos a la clase `ProblemaRiesgosPD` distintos de los ya indicados.

Nota 2: la contribución de un conjunto de propiedades a una lista de riesgos es la suma de las contribuciones de cada propiedad a cada riesgo. El método `int getContribucion (Propiedad p)` de la clase `Riesgo` devuelve la contribución de la propiedad `p` sobre un riesgo dado.


<<Java Class>>
ProblemaRiesgosPD
examenes.riesgos
k: int
nPropUsadas: int
valoresRiesgos: List<Integer>
ProblemaRiesgosPD(Problema,int,int,List<Integer>)
create(Problema,int,int,List<Integer>):ProblemaRiesgosPD
create(Problema):ProblemaRiesgosPD
getTipo():Tipo
getAlternativas():List<TipoAlternativa>
esCasoBase():boolean
getSolucionParcialCasoBase():Sp<TipoAlternativa>
getNumeroSubProblemas(TipoAlternativa):int
getSubProblema(TipoAlternativa,int):ProblemaPD<Set<Propiedad>,TipoAlternativa>
getSolucionParcialPorAlternativa(TipoAlternativa,List<Sp<TipoAlternativa>>):Sp<TipoAlternativa>
getSolucionParcial(List<Sp<TipoAlternativa>>):Sp<TipoAlternativa>
getSolucionReconstruidaCasoBase(Sp<TipoAlternativa>):Set<Propiedad>
getSolucionReconstruidaCasoRecursivo(Sp<TipoAlternativa>,List<Set<Propiedad>>):Set<Propiedad>
getObjetivoEstimado(TipoAlternativa):Double
getObjetivo():Double
size():int
hashCode():int
equals(Object):boolean
toString():String
-problemaOriginal
0..1
<<Java Class>>
Problema
examenes.riesgos
maxPropAFavorecer: int
sumaMaxPorRiesgo: int
nRiesgos: int
nProp: int
create():Problema
Problema()
+riesgos
0..*
<<Java Class>>
Riesgo
examenes.riesgos
id: int
siguienteId: int
create():Riesgo
Riesgo()
getContribucion(Propiedad):int
getAumentanProb():Set<Propiedad>
getDisminuyenProb():Set<Propiedad>
-disminuyenProb
-aumentanProb
<<Java Enumeration>>
TipoAlternativa
examenes.riesgos
Elegir: TipoAlternativa
NoElegir: TipoAlternativa
TipoAlternativa()
-solucionesDeAlternativas
+propiedades
<<Java Class>>
Sp<A1>
us.lsi.pd
alternativa: A1
propiedad: Double
estaEnLaSolucion: boolean
create(A2,Double):Sp<A2>
Sp(A1,Double)
toString():String
compareTo(Sp<A1>):int
<<Java Enumeration>>
Propiedad
examenes.riesgos
ConocimientosEquipo: Propiedad
Plazo: Propiedad
Escalabilidad: Propiedad
Seguridad: Propiedad
RecursosParaCalidad: Propiedad
Complejidad: Propiedad
Fiabilidad: Propiedad
Interoperabilidad: Propiedad
Propiedad()

ADDA 2ª Convocatoria-Septiembre Curso 2017/2018

APELLIDOS: _____
NOMBRE: _____

Problema de los Riesgos en los Proyectos Software	
Técnica*: Programación Dinámica*	
Tipos	• *S – Set<Propiedad>* • *A – [Elegir, NoElegir]*
Propiedades Compartidas	***riesgos****, List<Riesgo>, riesgos de un proyecto software, básica.* ***nRiesgos****, Integer, tamaño de la lista riesgos, derivada.* ***propiedades****, List<Propiedad>, propiedades disponibles, básica.* ***nProp****, tamaño de la lista propiedades, derivada.* ***maxPropAFavorecer****, Integer, máximo número de propiedades que se pueden favorecer, básica.* ***sumaMaxPorRiesgo****, Integer, máximo de la contribución total de las propiedades seleccionadas a cualquier riesgo, básica.* ***contribucion(p,r)****, Real, contribución que la propiedad p ejerce sobre el riesgo r, p en [0, nProp-1], r en [0, nRiesgos-1], básica.*
Propiedades Individuales	***k****, Integer* ***nPropUsadas****: Integer* ***valoresRiesgos****: List<Integer>, lista de tamaño nRiesgos.*
Solución: *Set<Propiedad>*	
Solución parcial (TODO): *(a,d)* *Siendo a la alternativa y d la contribución total de las propiedades seleccionadas a los riesgos.*	
Alternativas: $A = [Elegir, NoElegir]$	
Instanciación (TODO): $Inicial = (0, 0, [0, \dots, 0])$	
Casos base (TODO): *nPropUsadas=maxPropAFavorecer || k=nProp*	
Solución casos base (TODO): $\begin{cases} (?, 0.0), & si\ valoresRiesgos(i) \leq sumaMaxPorRiesgo, \quad i: 0..nRiesgos-1 \\ \bot, & e.o.c. \end{cases}$ *Otra opción, siendo* $x = \sum_{i=0}^{nRiesgos-1} valoresRiesgos(i)$ $\begin{cases} (?, x), & si\ valoresRiesgos(i) \leq sumaMaxPorRiesgo, \quad i: 0..nRiesgos-1 \\ \bot, & e.o.c. \end{cases}$	

Subproblemas:

$p = (k, nPropUsadas, valoresRiesgos) \xrightarrow{a=Elegir}$
$p_a = (k+1, nPropUsadas + 1, valoresRiesgos')$
$siendo\ valoresRiesgos'(i)$
$= valoresRiesgos(i) + contribucion(k,i), i: 0..nRiesgos - 1$

$p = (k, nPropUsadas, valoresRiesgos) \xrightarrow{a=NoElegir}$
$p_a = (k+1, nPropUsadas, valoresRiesgos)$

sA (***TODO***):

$$sA\big(a,(a',d)\big) = \begin{cases} (a, d + x), siendo\ x = \sum_{i=0}^{nRiesgos-1} contribucion(k,i) & si\ a = Elegir \\ (a,d) & e.o.c. \end{cases}$$

Si se ha elegido la segunda opción propuesta en el apartado ***Solución Casos Base****, entonces este apartado quedaría así:*

$sA\big(a,(a',d)\big) = (a,d)$

sP (TODO): *elige la solución con menor d.*

Solución reconstruida (TODO)

$sr((a,d)) = \{\}\ (caso\ base)$
$sr\big((a,d),s)\big) = s + propiedades(k)\ si\ a = Elegir$ *(caso recursivo)*
$sr\big((a,d),s)\big) = s \qquad si\ a = NoElegir$ *(caso recursivo)*

```java
@Override
public Sp<TipoAlternativa> getSolucionParcialCasoBase() {


      boolean hayRiesgosIncorrectos=
                  this.valoresRiesgos
                        .stream()
                        .anyMatch(r->
                        r > problemaOriginal.sumaMaxPorRiesgo);

            if (hayRiesgosIncorrectos)
                  return null;  //la seleccion de prop. no es solucion
            else {
                  TipoAlternativa daIgual= TipoAlternativa.NoElegir;
                  return Sp.create(daIgual, 0.0);
            }
}
```

Otra opción:

```java
@Override
public Sp<TipoAlternativa> getSolucionParcialCasoBase() {


      boolean hayRiesgosIncorrectos=
                  this.valoresRiesgos
                        .stream()
                        .anyMatch(r->
                        r > problemaOriginal.sumaMaxPorRiesgo);

            if (hayRiesgosIncorrectos)
                  return null;  //la seleccion de prop. no es solucion
            else {
                  TipoAlternativa daIgual= TipoAlternativa.NoElegir;
                  long x = IntStream
                        .range(0, problemaOriginal.nRiesgos)
                        .mapToLong (r->this.valoresRiesgos.get(r))
                        .sum();
                  return Sp.create(daIgual, (double)x);
            }
}
```

```
@Override
public Sp<TipoAlternativa>
            getSolucionParcialPorAlternativa (TipoAlternativa a,
                    List<Sp<TipoAlternativa>> ls) {


            Sp<TipoAlternativa> spHijo= ls.get(0);

            if (a==TipoAlternativa.Elegir) {
                  Propiedad pk= problemaOriginal.propiedades.get(k);
                  //Calculemos su contribucion.
                  long sumaContribuciones=
                  IntStream.range(0, problemaOriginal.nRiesgos)
                  .mapToLong(i->
                  problemaOriginal.riesgos.get(i).getContribucion(pk))
                  .sum();

                  return
                  Sp.create(a,sumaContribuciones+spHijo.propiedad);

            }
            else //a==TipoAlternativa.NoElegir
                  return Sp.create(a, spHijo.propiedad);

}
```

Si se ha elegido la segunda opción en el método getSolucionParcialCasoBase, este método quedaría así:

```
@Override
public Sp<TipoAlternativa>
            getSolucionParcialPorAlternativa (TipoAlternativa a,
                    List<Sp<TipoAlternativa>> ls) {


            Sp<TipoAlternativa> spHijo= ls.get(0);

            return Sp.create(a, spHijo.propiedad);

}
```

En cuanto al tercer método que se pide:

```
@Override
public Set<Propiedad>
     getSolucionReconstruidaCasoRecursivo
      (Sp<TipoAlternativa> sp, List<Set<Propiedad>> ls) {

            Set<Propiedad> result= ls.get(0);

            if (sp.alternativa==TipoAlternativa.Elegir)
                  result.add(problemaOriginal.propiedades.get(k));

            return result;
      }
```