Ejercicios 3 y 4 – Backtracking y GV/A*

Se tiene una lista de n alumnos (identificados de 0 a n-1) que se desea partir en 2 conjuntos (identificados por 0 y 1) de forma que alumnos incompatibles no pueden pertenecer al mismo conjunto, y se maximice el número de alumnos con dni par en el conjunto 0. Existe simetría en las incompatibilidades.

Alumno	DNI	Incompatibilidades
0	21352135	{1,6}
1	12341234	{0,3}
2	12787812	{5,3}
3	98765432	{1,2}
4	87654321	{6}
5	34567890	{2}
6	43679109	{0,4}

La solución para este ejemplo sería: [conjunto0 = {1, 2, 6}, conjunto1 = {0, 3, 4, 5}]. El problema de entrada viene modelado por la clase ProblemaAlumnos, y la solución por la clase SolucionAlumnos (ver diagrama al dorso).

SE PIDE:

Ejercicio 3. Para resolver el problema mediante BT, realice los siguientes métodos de la clase EstadoAlumnos (ver diagrama al dorso):

a) Tipo getTipo()
b) EstadoAlumnos getEstadoInicial()
c) EstadoAlumnos avanza(Integer a)
d) boolean esCasoBase()
e) List<Integer> getAlternativas()
f) Double getObjetivo()

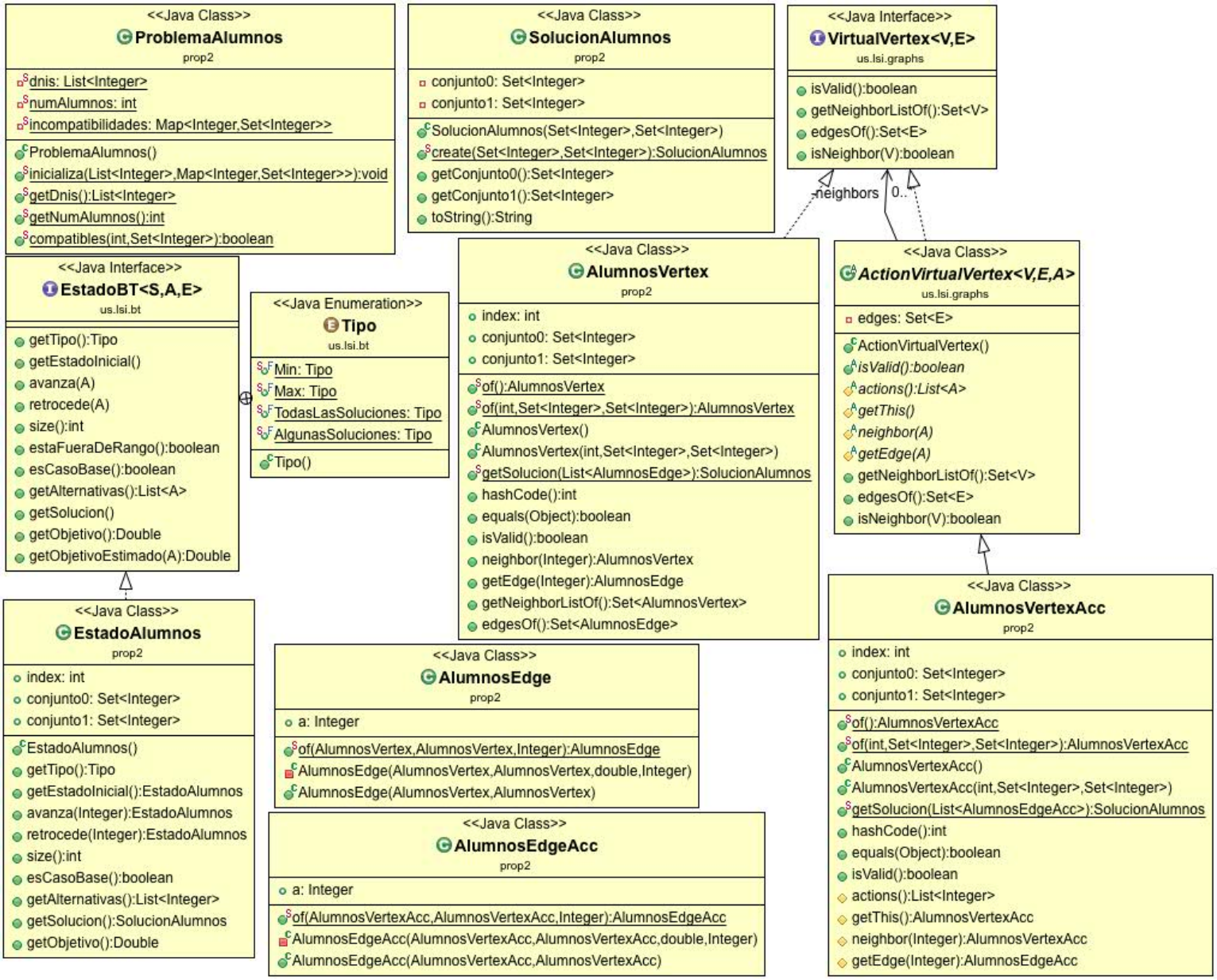
Ejercicio 4. Para resolver el problema mediante GV/A*, realice los siguientes métodos (ver diagrama al dorso). Puede decidir si resolverlo con acciones (apartados a.1, b.1, c.1 y d.1) o sin acciones (apartados a.2, b.2, c.2 y d.2):

a) 1. De la clase AlumnosVertexAcc, el método boolean isValid()
   2. De la clase AlumnosVertex, el método boolean isValid()
b) 1. De la clase AlumnosVertexAcc, el método AlumnosVertexAcc neighbor (Integer a)
   2. De la clase AlumnosVertex, el método AlumnosVertex neighbor(Integer a)
c) 1. De la clase AlumnosVertexAcc, el método List<Integer> actions()
   2. De la clase AlumnosVertex, el método Set<AlumnosVertex> getNeighborListOf()
d) Complete el siguiente método para crear una arista ponderada (el parámetro weight es el peso que debe tener la arista)
   1. De la clase AlumnosEdgeAcc,

```
AlumnosEdgeAcc of(AlumnosVertexAcc v1, AlumnosVertexAcc v2, Integer a) {
   //TODO
   return new AlumnosEdgeAcc(v1, v2, weight, a); }
```

   2. De la clase AlumnosEdge,

```
AlumnosEdge of(AlumnosVertex v1, AlumnosVertex v2, Integer a) {
   //TODO
   return new AlumnosEdge(v1, v2, weight, a); }
```



NOTA1: Como puede observar, las propiedades del estado y las propiedades del vértice son: index, conjunto0 y conjunto1.

NOTA2: El método compatibles(int i, Set<Integer> c) de la clase ProblemaAlumnos devuelve cierto en caso de que el alumno i sea compatible con todos los alumnos del conjunto c.

Solución - Ejercicio 3.

Apartado a)

```
Tipo getTipo() {
	return Tipo.Max;
}
```

Apartado b)

```
EstadoAlumnos getEstadoInicial() {
	index = 0;
	conjunto0 = new HashSet<Integer>();
	conjunto1 = new HashSet<Integer>();
	return this;
}
```

Apartado c)

```
EstadoAlumnos avanza(Integer a) {
	if(a == 0)
		conjunto0.add(index);
	else
		conjunto1.add(index);
	index++;
	return this;
}
```

Apartado d)

```
boolean esCasoBase() {
	return index == ProblemaAlumnos.getNumAlumnos();
}
```

Apartado e)

```
List<Integer> getAlternativas() {
	List<Integer> alternativas = new ArrayList<Integer>();
	if(ProblemaAlumnos.compatibles(index,conjunto0))
		alternativas.add(0);
	if(ProblemaAlumnos.compatibles(index,conjunto1))
		alternativas.add(1);
	return alternativas;
}
```

Apartado f)

```
Double getObjetivo() {
	Long res = conjunto0.stream().filter(e ->
(ProblemaAlumnos.getDnis().get(e)%2==0)).count();
	return res*1.0;
}}
```

Solución - Ejercicio 4

Apartado a) Igual con y sin acciones

```
boolean isValid() {
	return index>=0 && index<=ProblemaAlumnos.getNumAlumnos()
	&& conjunto0.stream().allMatch(i -> i <= index &&
ProblemaAlumnos.compatibles(i, conjunto0))
	&& conjunto1.stream().allMatch(i -> i <= index &&
ProblemaAlumnos.compatibles(i, conjunto1))
	&& !conjunto0.retainAll(conjunto1);
}
```

Apartado b) Igual con y sin acciones

```
AlumnosVertex neighbor(Integer a) {
	List<Integer> c0 = new ArrayList<Integer>();
	c0.addAll(this.conjunto0);
	List<Integer> c1 = new ArrayList<Integer>();
	c1.addAll(this.conjunto1);
	if(a.equals(0))
		c0.add(index);
	else
		c1.add(index);
	return AlumnosVertex.of(index+1,c0,c1);
}
```

Apartado c)

```
List<Integer> actions() {
	List<Integer> alternativas = new ArrayList<Integer>();
	if(ProblemaAlumnos.compatibles(index,conjunto0))
		alternativas.add(0);
	if(ProblemaAlumnos.compatibles(index,conjunto1))
		alternativas.add(1);
	return alternativas;
}

Set<AlumnosVertex> getNeighborListOf() {
	List<Integer> alternativas = new ArrayList<Integer>();
	if(ProblemaAlumnos.compatibles(index,conjunto0))
		alternativas.add(0);
	if(ProblemaAlumnos.compatibles(index,conjunto1))
		alternativas.add(1);
	Set<AlumnosVertex> res = new HashSet<AlumnosVertex>();
	alternativas.stream().forEach(a -> res.add(neighbor(a)));
	return res;
}
```

Apartado d) Igual con y sin acciones.

```
static AlumnosEdge of(AlumnosVertex v1, AlumnosVertex v2, Integer a) {
	Double weight = 0.0;
	if(a.equals(0) && ProblemaAlumnos.getDnis().get(v1.index)%2==0)
		weight = -1.0;
	return new AlumnosEdge(v1, v2, weight, a);
}
```