ADA Septiembre Curso 2012/2013

Nombre: _____

Titulación; _____

Ejercicio 3

Con el fin de aumentar el rendimiento de los nuevos dispositivos móviles, se ha diseñado un nuevo sistema operativo basado en restricciones que permite predecir qué recursos serán usados en cada momento y, en función de esta predicción, minimizar el gasto de energía mediante el apagado y encendido de proveedores de información.

Entendemos por proveedor de información una aplicación, residente en el sistema operativo, que es capaz de aportar cierta información a las aplicaciones que se ejecutan en el dispositivo. Cada proveedor de información permite suministrar información a una o más aplicaciones. De este modo, un proveedor puede cubrir las necesidades de una serie de aplicaciones del terminal. En el ejemplo, podemos ver que el proveedor P_1 aporta la información necesaria para las aplicaciones A, B y D. De igual forma, el proveedor P_2 lo hace para las aplicaciones B, C y E.

Fijémonos que tanto P_1 como P_2 aportan información a la aplicación B, por lo que tan sólo uno de ellos sería necesario para dicha aplicación.

Sin embargo, esta información no es gratuita, sino que los proveedores consumen una cierta cantidad de energía. Esta energía medida en Ah/h es propia de cada proveedor de información.

El objetivo es encontrar, a través de un algoritmo de backtracking, una configuración de proveedores que permita dar cobertura a <u>todas</u> las aplicaciones con el menor consumo energético. Esto es, conseguir que con los proveedores seleccionados se puedan ejecutar la totalidad de aplicaciones que el sistema operativo está procesando.

Ejemplo: La siguiente tabla muestra un ejemplo de datos de entrada. Para el conjunto de actividades {A, B, C, D, E, F, G} y el conjunto de proveedores de información {P_1, P_2, P_3, P_4, P_5, P_6} con los consumos abajo indicados, la mejor configuración se proveedores seleccionados será {P_2, P_3, P_6}, con un coste energético total de 400 Ah/h.

Proveedor	Aplicaciones cubiertas	Gasto de batería (Ah/h)
P_1	**{A,B,D}**	**200**
P_2	{B,C,E}	**130**
P_3	{A,D}	**120**
P_4	{E,G}	**300**
P_5	{F}	**70**
P_6	{F,G}	**150**

Para solucionar el problema planteado, suponga que existen *n* proveedores de información (p_1, p_2,..., p_n). El tipo **Proveedor** posee tres propiedades con sus métodos de acceso correspondientes denominados **getProveedor**() (que devuelve un string con el nombre del proveedor) y **getConsumo**() (que devolverá un valor de tipo Integer con el gasto energético del proveedor). Por último, el método **getAplicaciones()**, que devuelve un conjunto de string (Set<String>) con el nombre de las aplicaciones a las que da soporte el proveedor.

Tenga en cuenta además que una aplicación satisface sus necesidades **si al menos existe un proveedor** de información seleccionado compatible con dicha aplicación.

El problema se modelará a través de la clase **ProblemaProveedoresImpl**, que implementa la interfaz `ProblemaRyP<Integer, Boolean, EstadoProveedores>`.

SE PIDE:

a) Escribir el código completo del método *ryp*() de la clase AlgoritmoRyP<List<Proveedor>, Boolean, EstadoProveedores>.

b) Completar la ficha del problema. Concretamente los apartados Tamaño, Propiedades del Estado y EstadoInicial.

c) Escribir el método *getAlternativas*() de la clase ProblemaProveedoresImpl. Tenga en cuenta que tal cómo se puede ver en la ficha proporcionada, las alternativas serán valores lógicos que indicarán si la actividad recibirá subvención o no.

d) Escribir el código completo de los métodos *add*(Boolean) y *remove*() de la clase EstadoProveedoresImpl, teniendo en cuenta la ficha proporcionada.

BT	
Problema Proveedores Móviles	
Técnica: Backtracking	
Tamaño: ***TODO***	
Propiedades Compartidas	***listadoProveedores****: List<Proveedor>* ***listadoActividades****: List<String>*
Propiedades del Estado	***TODO***
Solución: List<Proveedor>	
Estado inicial: ***TODO***	

```
	private void ryp() {
		S solucion = estado.getSolucion();
		if (problema.esSolucion(solucion)) {
		   soluciones.add(solucion);
		    fin = esFin();
		}
		if (!fin && problema.isAlternativa(estado)) {
		   Iterable<A> it = problema.getAlternativas(estado);
		   for (A alternativa: it){
				estado.add(alternativa);
				ryp();
				estado.remove();
			}
		}
	}

	public boolean add(Boolean e) {
		if (e){
			Actividad a =
listaActividades.get(actividadesProcesadas);
			actividadesAtendidas.add(a);
			presupuestoDisponible -= a.getCoste();
			habitantesTotal += a.getHabitantes();
		} actividadesProcesadas++;
		return true;
	}

	public Boolean remove() {
		actividadesProcesadas--;
		Actividad a = listaActividades.get(actividadesProcesadas);
		Boolean ret = actividadesAtendidas.contains(a);
		if (ret){
			actividadesAtendidas.remove(a);
			presupuestoDisponible += a.getCoste();
			habitantesTotal -= a.getHabitantes();
		}
		return ret;
	}

	public Iterable<Boolean> getAlternativas(EstadoActividades e) {
		List<Boolean> it = Lists.newLinkedList();
		it.add(false);
		if (e.getPresupuestoDisponible() >=
	e.getListaActividades().get(e.getActividadesProcesadas()).getCoste())
			it.add(true);
		return it;
	}

	public boolean isAlternativa(EstadoActividades e) {
		return e.getActividadesProcesadas() <
		e.getListaActividades().size() &&
		e.getPresupuestoDisponible() > 0;
	}
}
```