Ejercicio 5: Programación Dinámica

Se desea diseñar un algoritmo que gestione el termostato al que tienen que funcionar los sistemas de acondicionamiento de temperatura de un conjunto de oficinas (*OF*) de un edificio. Tenga en cuenta que:

1. Cada oficina ha indicado su temperatura preferida (*tempPref*). Dicho deseo se considerará satisfecho si la temperatura del termostato está en un rango *[tempPref-1,tempPref+1].* En cualquier caso, el termostato de cada oficina no podrá exceder del rango *[temPref-3, tempPref+3]*.
2. Por cada oficina, se dispone de una función (*getConsumo(int temp)*) que, dada una temperatura, devuelve la energía que consume dicha oficina.
3. Las posibles temperaturas de los termostatos pueden variar en el rango *[MinT, MaxT]* en pasos de 1 º.
4. Existe un consumo energético máximo del edificio (*CM*) que no puede ser superado. Es decir, la suma del consumo de todas las oficinas no puede superar CM.
5. Se busca encontrar la temperatura adecuada para cada oficina de manera que se maximice el número de oficinas satisfechas.

Ejemplo:

Para los siguientes datos:

	Temperatura Preferida	Función de Consumo (ud.)				
		26 º	**27 º**	**28 º**	**29 º**	**30 º**
0	28 º	6	5	4	3	1
1	26 º	10	6	2	1	0
2	27 º	6	3	2	1	1

minT	MaxT	CM
26 º	30 º	9 ud.

Una solución sería:

termostatos=[27º, 28º, 28º],
gastoEnergético=9ud.
oficinasSatisfechas=2

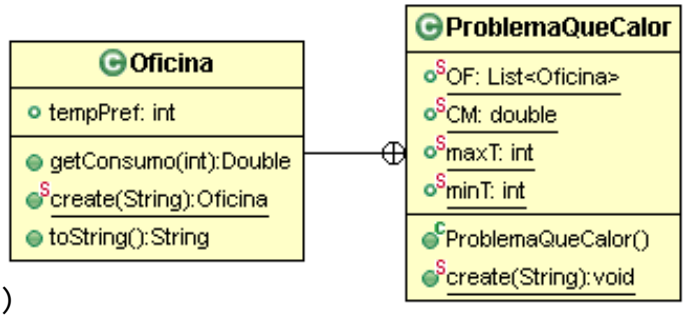
Como puede observarse, la oficina 1 no queda satisfecha por haber una diferencia *> 1* entre la temperatura preferida y la temperatura asignada por el termostato.

Ejemplo sin solución:

Para los mismos datos de oficinas pero cambiando los siguientes datos:

minT	MaxT	CM
26 º	30 º	2 ud.

No hay solución debido a la restricción de no poder estar a más de 3º de la temperatura preferida por lo que la oficina 1 no podría tener el termostato a 30º que es lo que sería necesario para llegar a una solución.



Se pide:

1. Completar la ficha.
2. Implemente:

a. `List<Integer> getAlternativas()`
b. `ProblemaPD getSubProblema(Integer a, int np)`
c. `Sp<Integer> combinaSolucionesParciales(Integer a,List<Sp> ls)`

Problema QueCalor	
Técnica: PD	
Prop. Compartidas:	OF, List<Oficinas>, Oficinas CM, double, Consumo máximo MinT, MaxT, int, mínima y máxima temperatura del termostato
Prop. Individuales:	i, int, índice en [0, |OF|] cRest, double, Consumo restante
Tamaño:	***TODO***
Solución:	List<Integer> lista de las temperaturas de cada oficina
Objetivo:	Encontrar las temperaturas que haga satisfacer al mayor número de oficinas
Alternativas:	$Ax = \{a \in [minT, .., maxT] \mid OF_i.getConsumo(a) \leq cRes \wedge \lvert OF_i.tempPref - a \rvert \leq 3$
Instanciación:	***TODO***
Problema generalizado:	***TODO***
SubProblema(a)	$(i+1, cRes - OF_i.getConsumo(a))$
cS(a, (a',v))	$(a, v + \lvert OF_i.tempPref - a \rvert \leq 1 ? 1 : 0)$

Problema QueCalor	
Técnica: PD	
Prop. Compartidas:	OF, List<Oficinas>, Oficinas CM, double, Consumo máximo MinT, MaxT, int, mínima y máxima temperatura del termostato
Prop. Individuales:	i, int, índice en [0, |OF|] cRest, double, Consumo restante
Tamaño:	**|OF|-i**
Solución:	List<Integer> lista de las temperaturas de cada oficina
Objetivo:	Encontrar las temperaturas que haga satisfacer al mayor número de oficinas
Alternativas:	$Ax = \{a \in [minT,..,maxT] \mid OF_i.getConsumo(a) \leq cRes \wedge \lvert OF_i.tempPref - a\rvert \leq 3$
Instanciación:	**i=0, cRest=CM *(TODO)***
Problema generalizado:	$\boldsymbol{aux(i,cRes)} = \begin{cases} \boldsymbol{(null,0)}, & \boldsymbol{i = \lvert OF\rvert} \\ \boldsymbol{\max_{a \in Ax}\{cS(a, aux(sp(a))\}}, & \boldsymbol{en\ otro\ caso} \end{cases}$ **(TODO)**
Sp(a)	$(i+1, cRes - OF_i.getConsumo(a))$
cS(a, (a',v))	$(a, v + \lvert OF_i.tempPref - a\rvert \leq 1 ? 1 : 0)$

```
a)public List<Integer> getAlternativas() {
	return=IntStream.rangeClosed(minT, maxT)
		.boxed()
		.filter(x -> cumpleRestricciones(x))
		.collect(Collectors.toList());
}

private boolean cumpleRestricciones(Integer temperatura){  //consumo a
gastar menor que consumo restante
	boolean res=(o.getConsumo(temperatura) <= cRes);
	//temperatura entre +-3 de la temp deseada
	res &= Math.abs(o.tempPref-temperatura) <=3;

	return res;
}

b)public ProblemaPD<SolucionProblemaQueCalor, Integer>
				getSubProblema(Integer a, int np) {
	double cResNext=cRes-OF.get(i).getConsumo(a);
	return new ProblemaQueCalorPD(i+1, cResNext);
}

c)public Sp<Integer> combinaSolucionesParciales(
				Integer a, List<Sp<Integer>> ls) {
	boolean cumple= Math.abs(OF.get(i).tempPref-a) <= 1;
	return Sp.create(a, ls.get(0).propiedad+(cumple?1:0));
}
```