Le model

Prenez le temps nécessaire pour bien comprendre l'architecture du projet et le modèle associé pour la conception des VM et PM.

Question 1

Nous disposons d'un centre de données avec disposant de ***nbMachines*** et ***nbVMs***. Chaque VM possède une priorité entre 1 et 10. 10 étant une VM hautement prioritaire. Pour des raisons d'administration nous ne souhaitons que il y ait au maximum ***sumOnOff*** VM en état running. (Nous avons donc ***nbVMs - sumOnOff*** VM éteintes). Cependant nous voulons que les VM allumées soient les plus prioritaires, ce qui en d'autres termes peut se résumer à maximiser la somme des priorités de VM allumée.

Max(somme(vm[i].priority*vm.state) i from 0 to ***nbVMs)***

Il vous est demandé de modifier le fichier PriorityConstraint.java.

Question 2

La suite de ces questions portent sur le fichier NRJConstraint.java

```
vmMemory[i] donne la taille mémoire de l'objet i.
vmIdMemory[i] donne l'identifiant VM de l'objet i.
```

```
pmMemory = new PackModeler(vmMemory, machines.size(), 32);
```

Construit un objet PackModeler composé d'un tableau d'item vmMemory. L'objectif est de placer tous les objets i dans des "sacs". Le nombre de "sac" est définit par machines.size(). La taille maximum du sac est 32.

```
model.addConstraints(Choco.pack(pmMemory));
```
La ligne précédente permet d'ajouter au solveur la contrainte spécifiée pour le packing.

la méthode `pmMemory.load[x]` permettra de récupérer la charge du "sac" x.
la méthode `pmMemory.bins[i]` donnera le numéro du sac ou a été placé l'objet i.
et `vmIdMemory[i]` donnera l'identifiant VM de l'objet i.

Q2.1 Faites la même chose pour ajouter une contrainte de packing sur le CPU.

Q2.2 Question connexe
Extrait de l'utilisation de la méthode pack avec sa signature de base, sans l'objet PackModeler.

`pack(items, load, bin, size)` states that a collection of items is packed into different bins, such that the total size of the items in each bin does not exceed the bin capacity:

$$\texttt{load}[b] = \sum_{i \in \texttt{items}[b]} \texttt{size}[i], \quad \forall \text{ bin } b$$

$$i \in \texttt{items}[b] \iff \texttt{bin}[i] = b, \quad \forall \text{ bin } b, \ \forall \text{ item } i$$

- `SetVariable[] items`: $\texttt{items}[b]$ is the set of items packed into bin b.
- `IntegerVariable[] load`: $\texttt{load}[b]$ is the total size of the items packed into bin b.
- `IntegerVariable[] bin`: $\texttt{bin}[i]$ is the bin where item i is packed into.
- `IntegerConstantVariable[] size`: $\texttt{size}[i]$ is the size of item i.

Par rapport à cette méthode, comment définit on le bin capacity de chacun des bins ?

Q2.3 Préciser le rôle de la boucle suivante :

```
for (Machine m : machines) {
		if (m.getMemory() != 32) {
			model.addConstraints(Choco.leq(pmMemory.loads[itm],
					m.getMemory()));
		}
		if (m.getCpu() != 16) {
			model.addConstraints(Choco.leq(pmCpu.loads[itm], m.getCpu()));
		}
		itm++;
}
```

Q2.4 Multi bin packing

Dans notre système nous avons mis en place deux contraintes de packing, l'une sur la RAM l'autre sur le CPU. De ce fait, une VM ayant pour identifiant id, placée sur la machine X par le premier packing (mémoire), doit forcément être placé sur la même machine X dans le second packing (cpu). Pour cela il faut relier les "sacs" entre les deux systèmes de packing (via `pmMemory.bins[i]` ...)

```
	for (int i = 0; i < vmIdMemory.length; i++) {
		for (int j = 0; j < vmIdCpu.length; j++) {
			if (vmIdMemory[i] == vmIdCpu[j]) {
				// Q2 : Lier le bin CPU avec le bin Memoire
				// A remplir : 1 ligne
			}
		}
		}
```