TD2– Programmation par Contraintes-Choco
4A STI

Exercice 1 :

1-Commenter ce code, en précisant chaque partie correspond à quelle étape de la déclaration et/ou résolution du problème en utilisant Choco.

```
public class exercice1 {

    public void modelAndSolve(){
        int n = 8; // On a 8 dames
        /*
         * Déclaré le Model
         */
        Model model = new Model(n + "-xxxx problem"); //Déclaré le Model
        IntVar[] vars = new IntVar[n];
        for(int q = 0; q < n; q++){
            vars[q] = model.intVar("Q_"+q, 1, n); // déclaré le domaine de chacun
variable Dc[i] = [1,8]
        }
        /*
         * Ici on déclare des contraints
         */
        for(int i  = 0; i < n-1; i++){
            for(int j = i + 1; j < n; j++){
                model.arithm(vars[i], "!=",vars[j]).post();  // 2 dames ne sont
pas sur la meme lignes
                model.arithm(vars[i], "!=", vars[j], "-", j - i).post(); // 2
dames ne sont pas sur la meme diago
                model.arithm(vars[i], "!=", vars[j], "+", j - i).post(); // 2
dames ne sont pas sur la meme diago
            }
        }
        // Ici model trouve la solution
        Solution solution = model.getSolver().findSolution();
        if(solution != null){
            System.out.println(solution.toString());
        }
    }

    public static void main(String[] args) {
        new exercice1().modelAndSolve();
    }
```

2-En déduire sa modélisation formelle. Reconnaissez-vous le problème xxxx?

Le problème xxx est le problème de 8 dames.
Variable : Q_0, Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7
Domaine : Q_i = {1…8} , i =[0,7]
Contraintes :

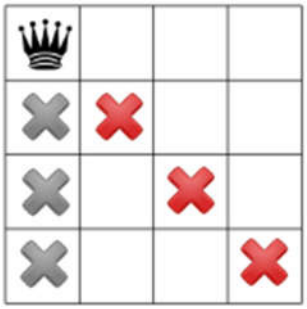
$Q\_[i] \neq Q\_[j]$ et $Q\_[i] \neq Q\_[j] \pm (j-i), i, j \in [0,7], i < j$

Deux dames ne sont ni sur la même colonnes, ni les lignes, ni la diago.

3-Donner une solution de ce problème, en supposant que le solveur suit une méthode de filtrage par arc (ordre croissant des variables et valeurs).

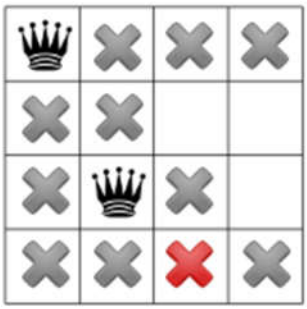
On consiste que dans le N = 4 :

1. D'abord , Q_0 = 1 :



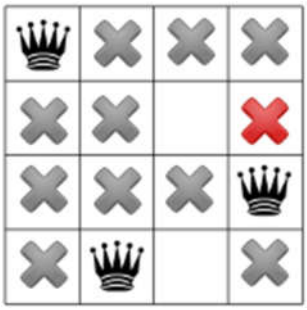
1.1.Si Q_[1] = 1 ou 2, ca ne marche pas , donc Q_[1] = 3 ou 4

1.1.1. Si Q_[1] = 3 :



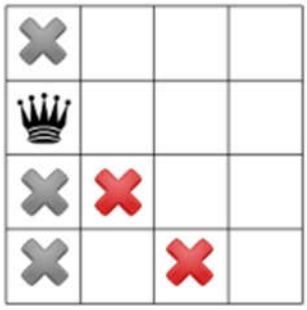
Selon l'image, c'est pas la place pour mettre la dame Q_[2] . En conclusion , dans le cas Q_[1] = 3 , on ne trouve jamais la solution de ce problème

1.1.2. Si Q_[1] = 4 :

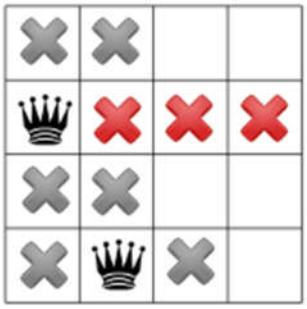


Selon l'image, Q_[3] = 3 est seul la solution . Mais dans ce cas-là, Q_[2]= 2 ou 4 , ca ne marche pas. Donc le cas Q_[1] = 4, on ne trouve jamais aussi la solution de ce problème.

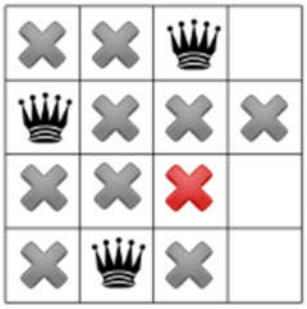
2. Si Q_[0] = 2 :



Selon l'image, Q_[1] = 4 est seul la solution.



Ensuit, on voit que Q_[2] = 1 est aussi seul la solution.



Enfin, on voit aussi que Q_[3] = 3 est seul la solution.

En conclusion, Q_[0] = 2 , Q_[1] = 4, Q_[2] = 1, Q_[3] = 3 est la solution de ce problème.

Dans ce cas n = 8, on fait la même

Une solution :

			Q				
					Q		
							Q
		Q					
Q							
						Q	
				Q			
	Q						

Exercice 2 :

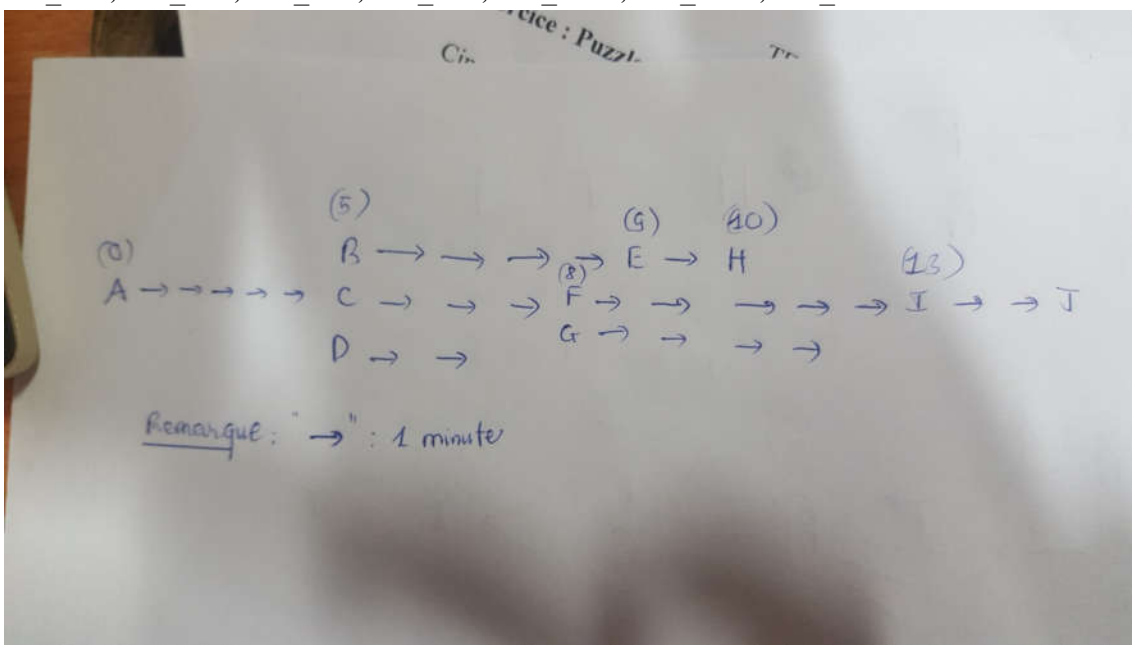
Implémenter le problème d'ordonnancement suivant avec Choco :

Il y a 10 tâches:

A, B,C,D,E,F,G,H,I,J à exécuter de durées respectives : 5,4,3,2,1,5,4,3,2,1,

en respectant les contraintes de précédences :
A avant B, C, D -- B avant E -- C avant F, G -- D avant F -- E avant H -- F avant I -- G avant I -- H avant J -- I avant J.
Toutes les tâches doivent commencer dans un délai de 15 minutes.

J'ai implémenté dans le exercice2.java et la solution est: *deb_0=0, deb_1=5, deb_2=5, deb_3=5, deb_4=9, deb_5=8, deb_6=8, deb_7=10, deb_8=13, deb_9=15*



Exercice 3 :
On note w1(2) l'écriture par la transaction 1 d'un objet 2 et r2(1) l'écriture par une transaction 2 d'un objet 1.
w1(1), r2(1), w1(2), w3(3), r2(3), r4(2), w2(4), w4(5), r5(4), w5(5)

1) Proposez un problème PSC qui cherche si une séquence d'opérations est sérialisable ou non. Implémenter sa solution en Choco.
2) Implémenter l'algorithme backtrack pour résoudre ce problème. (En Java)
3) Donner la taille max dont on peut résoudre le problème en moins d'une seconde.

Pour mesure la temps d'excution, on lancera au début du programme un chrono :

```
long start = System.currentTimeMillis();
```

Et à la fin du programme :

```
System.out.println("Temps d'exécution"+(System.currtentTimeMillis()-start));
```

REMARQUE :

Pour comprendre facilement mon code et compiler rapidement, je note que :

(i) Write est 0, Read est 1.

(ii) Comme vous me demande qu'on saisisse des ressources et des opérations par la variable input, mais cela prendra du temps pour tester. Donc, dans la partie main(), j'ai 2 parties : Partie 1 : Saisir et Partie 2 : Tester énonce (il est exemple qu'on fait : w1(1), r2(1), w1(2), w3(3), r2(3), r4(2), w2(4), w4(5), r5(4), w5(5)).
Si vous voudriez saisir, il faut commenter des codes dans le partie 2 et si vous voudriez tester l'énonce, il faut aussi commenter des codes dans le partie 1.

(iii) Une transaction est décrit comme ca :
Par ex : w1(1) → {0,1,1,1}, r2(1) -> {1,2,1,2} etc…

(iv) Dans le Java, l'index d'array commence par 0 , donc quand le résultat affiche que : « T_0 =1 » , c'est-à-dire que la ressource 1 est la première tâche.

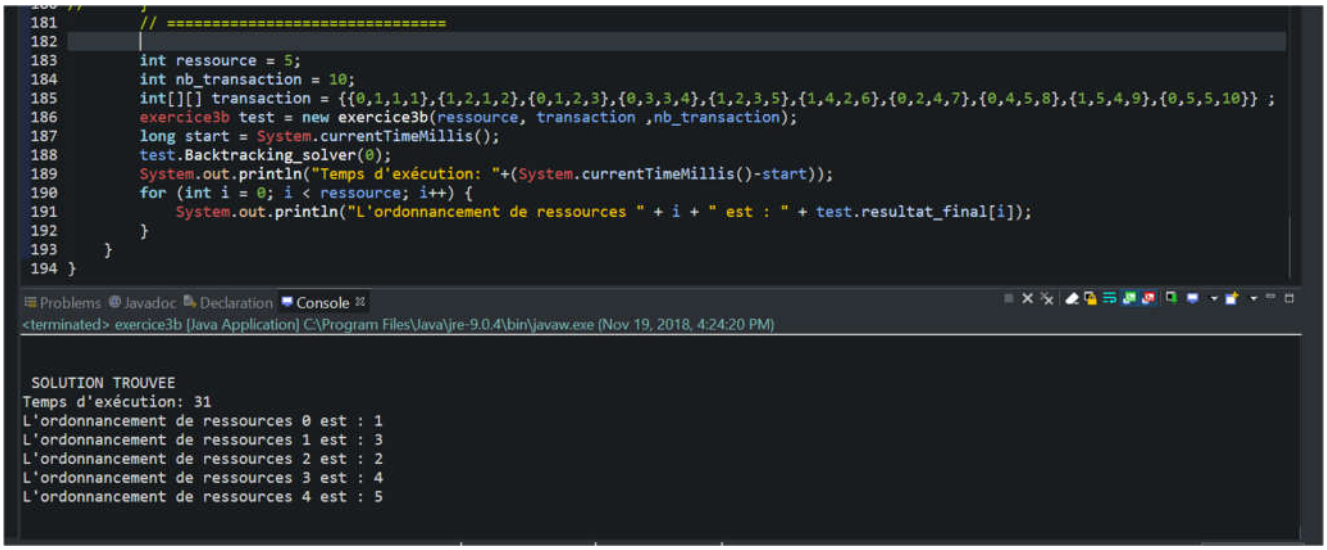
1) Voir le code dans le fichier exercice3.java

Résultat obtenu selon l'énoncé :

```
73 //          System.out.println("{"+transaction[i][0]+","+transaction[i][1]+","+transaction[i][2]+","+transaction[i][3]+"}");
74 //      }
75        long start = System.currentTimeMillis();
76        int ressource = 5;
77        int[][] transaction = {{0,1,1,1},{1,2,1,2},{0,1,2,3},{0,3,3,4},{1,2,3,5},{1,4,2,6},{0,2,4,7},{0,4,5,8},{1,5,4,9},{0,5,5,10}};
78        exercice3 test = new exercice3(ressource, transaction);
79        test.modelAndSolve();
80        System.out.println("Temps d'exécution: "+(System.currentTimeMillis()-start));
81    }
82 }
83
Problems  Javadoc  Declaration  Console
<terminated> exercice3 [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (Nov 19, 2018, 3:59:44 PM)
Solution: T_0=1, T_1=3, T_2=2, T_3=4, T_4=5,
Temps d'exécution: 186
```

2) Backtracking :
Voir le code dans le fichier exercice3b.java

Resultat obenu selon l''enonce :



```
        // ==============================

        int ressource = 5;
        int nb_transaction = 10;
        int[][] transaction = {{0,1,1,1},{1,2,1,2},{0,1,2,3},{0,3,3,4},{1,2,3,5},{1,4,2,6},{0,2,4,7},{0,4,5,8},{1,5,4,9},{0,5,5,10}} ;
        exercice3b test = new exercice3b(ressource, transaction ,nb_transaction);
        long start = System.currentTimeMillis();
        test.Backtracking_solver(0);
        System.out.println("Temps d'exécution: "+(System.currentTimeMillis()-start));
        for (int i = 0; i < ressource; i++) {
            System.out.println("L'ordonnancement de ressources " + i + " est : " + test.resultat_final[i]);
        }
    }
}
```

Problems Javadoc Declaration Console

<terminated> exercice3b [Java Application] C:\Program Files\Java\jre-9.0.4\bin\javaw.exe (Nov 19, 2018, 4:24:20 PM)

```
 SOLUTION TROUVEE
Temps d'exécution: 31
L'ordonnancement de ressources 0 est : 1
L'ordonnancement de ressources 1 est : 3
L'ordonnancement de ressources 2 est : 2
L'ordonnancement de ressources 3 est : 4
L'ordonnancement de ressources 4 est : 5
```

3) Tems d' exécution : 31 avec la taille ressource 5
   Donc la taille max dont on peut résoudre le problème en moins d'une seconde est :
   $\frac{1000(ms).5}{31(ms)} \sim 161$.