TP1 : Programmation d'un algorithme de backtrack

Objectif

L'objectif de ce TP est de programmer l'algorithme de backtrack vu en cours en l'appliquant au problème CSP (« Constraint Satisfaction Problem »). Ce problème est décrit sous la forme de :

- variables,
- domaines (donnés en extension),
- contraintes (données en extension).

On ne vous demande pas de créer d'interface graphique, une exécution en mode console suffira. Les indications fournies ci-dessous sont relatives au langage Java, mais vous êtres libres d'utiliser le langage de programmation que vous voulez (à condition qu'il y ait un environnement de programmation adéquat en salle de TP).

Etape 1 : Représenter une instance du problème

On choisit de décrire un CSP de façon très simple (que vous pourrez améliorer par la suite) :

- une **variable** est représentée par une chaîne de caractères (String) : son « nom »
- une **valeur** possible pour une variable est un objet quelconque (Object)
- un **domaine** est un ensemble de valeurs (on le prendra ordonné pour permettre la mise en place éventuelle d'heuristiques sur le choix d'une valeur)
- on a besoin d'accéder rapidement au domaine d'une variable : ici, on a choisi une table de hachage (HashMap), qui associe à chaque variable son domaine
- une **contrainte** est décrite par son nom (qui peut être utilisé en affichage ; le nom est soit une chaîne de caractères quelconque, soit une chaîne formée de « C » + un numéro unique), un ensemble ordonné de variables, et un ensemble de tuples (qui sont eux-mêmes des suites de valeurs)
- un **CSP** est donné par la table associant variables et domaines, et un ensemble de contraintes.

➔ Lancez votre environnement Java favori (Eclipse par exemple) et créez un projet contenant les fichiers sources java (ainsi que .txt) que vous trouverez sur l'ENT.
➔ Observez le codage choisi pour décrire un CSP (classe CSP), et en particulier une contrainte (classe Constraint).

CSP :

```
private HashMap<String,TreeSet<Object>> varDom;
// table de hachage associant à chaque variable son domaine
private ArrayList<Constraint> constraints; // liste des contraintes
```

Constraint :

```
private static int num=0; //rang de la contrainte dans l'ensemble de contraintes
private String name; // nom de la contrainte
private ArrayList<String> varTuple; // ensemble ordonné de variables
private Set<ArrayList<Object>> valTuples; // ensemble de tuples de la contrainte
```

➔ Lire le code de la méthode `main` de la classe CSP : celle-ci créée une instance de CSP « vide » et la remplit par ajout successif de variables, domaines et contraintes. Exécutez cette méthode. Comprenez les messages dus aux contrôles effectués par les méthodes `addVariable`, `addValue` et `addConstraint`.

Etape 2 : Saisir et afficher une instance du problème

On veut pouvoir construire une instance de CSP à partir d'un fichier texte. On supposera que le fichier a une syntaxe conforme aux règles syntaxiques que vous vous donnerez. Il ne vous est donc pas demandé de vérifier sa syntaxe.

Exemple de codage texte commode (voir par exemple `example.txt`), où les éléments d'une liste sont séparés par des points-virgules :

```
nombre de variables
pour chaque variable :
nom de la variable ; liste des valeurs de la variable
nombre de contraintes
pour chaque contrainte :
liste des variables de la contrainte
nombre de tuples de la contrainte
      pour chaque tuple de la contrainte :
      liste des valeurs composant le tuple
```

➔ Ajoutez à la classe CSP un **constructeur** qui prenne en paramètre un fichier texte décrivant l'instance de CSP.
➔ Testez.

Indications Java

- Vous trouverez en annexe des rappels sur la lecture de fichiers texte en Java
- Pour traduire une `String` en un `int`, voir la méthode `Integer.parseInt`
- Pour découper une ligne lue, utiliser la classe `StringTokenizer` (penser à indiquer au constructeur quel caractère sert de séparateur)
- Tout chemin d'accès au fichier donné de façon non absolue est supposé relatif à la racine du projet (autrement dit, placez vos fichiers texte au premier niveau du répertoire du projet si vous voulez y accéder par leur nom sans préciser de chemin d'accès).

Etape 3 : Rechercher une solution

➔ La classe `Solver` a pour objectif de résoudre une instance de CSP. Complétez le squelette fourni en implémentant la méthode `backtrack`, qui retourne une solution s'il en existe une, et null sinon. Cette méthode fera typiquement appel à des méthodes de test de consistance que vous devrez ajouter dans les classes `CSP` et `Constraint`.

➔ Testez votre algorithme sur un exemple de CSP (un problème de coloration, les n reines avec n petit – le codage en extension étant très fastidieux, le puzzle du zèbre – là aussi, très fastidieux avec le codage en extension, …)

Etape 4 : Rechercher toutes les solutions

➔ Programmez une seconde version de backtrack qui calcule l'ensemble de toutes les solutions.

Annexe : rappels sur les fichiers texte en Java

/ Exemple de programme **lisant** un fichier texte nommé **"essai.txt"** */*

```
import java.io.*;
public class FicText2
{
      public static void main(String args []) throws IOException
      { BufferedReader lectureFichier = new BufferedReader( new
                        FileReader ("essai.txt"));
        System.out.println("Lecture du fichier essai.txt");
        String s = lectureFichier.readLine();

        /* readLine() retourne :
            -  la ligne lue jusqu'au retour chariot (lu mais non retourné)
               donc une chaîne vide si la ligne ne comporte qu'un RC
            -  la valeur null s'il n'y a rien à lire (fin du flux de données)
         */
         while (s!= null)
            { System.out.println(s);
              s = lectureFichier.readLine();
            }
       lectureFichier.close();
       System.out.println("Fin du fichier");
      }
}
```

/ Exemple de programme **créant** un fichier texte nommé **"essai.txt"** */*

```
import java.io.*;
public class FicTexte1
{
      public static void main(String args []) throws IOException
      { BufferedReader lectureClavier = new BufferedReader( new
                        InputStreamReader (System.in));
        BufferedWriter ecritureFichier = new BufferedWriter( new
                        FileWriter ("essai.txt"));
        System.out.println("Entrez des lignes (Return pour terminer)");
        String s = lectureClavier.readLine();
        while (s.length() != 0)
            { ecritureFichier.write(s); // TQ pas chaîne vide
              ecritureFichier.newLine();
              s = lectureClavier.readLine();
            }
        ecritureFichier.close(); // ferme le fichier associé
        System.out.println("Fin saisie");
      }
}
```