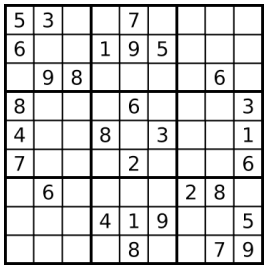
TP projet

Exercice 1 - Gestion d'ensembles d'entiers

Un ensemble sera représenté par une liste triée d'entiers. Chaque élément de l'ensemble n'apparaît qu'une seule fois dans la liste.

1. Nous souhaitons écrire la fonction `CreerEnsemble` qui, à partir d'une liste d'entiers, renvoie l'*ensemble* issu de cette liste. Cette fonction utilisera la fonction `Trier` qui permet le tri d'une liste de nombres selon l'algorithme de tri par insertion du minimum et la fonction `RetirerDoublon` qui permet de supprimer les doublons d'une liste triée de nombres.
2. Ecrire la fonction `Union` qui renvoie l'*ensemble* correspondant à l'union de deux ensembles. Nous rappelons que l'ensemble résultat doit être trié et ne doit pas contenir de doublon.
3. Ecrire la fonction `Intersection` qui renvoie l'*ensemble* correspondant à l'intersection de deux ensembles.
4. Ecrire la fonction `Difference` qui renvoie l'*ensemble* correspondant à la différence de deux ensembles A et B (A privé des éléments de B).

Exercice 2 - Sudoku



Un Sudoku est une grille constituée de n lignes et de n colonnes contenant des chiffres allant de 1 à n. Une **ligne** est l'ensemble des cases contigües horizontalement. Une **colonne** est l'ensemble des cases contigües verticalement. Une **région** est un ensemble de $\sqrt{n} \times \sqrt{n}$ cases contigües.

Les lignes sont numérotées de 1 à n du haut vers le bas. Les colonnes sont numérotées de 1 à n de gauche à droite. Les régions sont numérotées de 1 à n, du haut vers le bas et de gauche à droite.

Certaines cases de la grille sont remplies au départ et l'objectif du jeu est de remplir les cases vides. Le remplissage de la grille doit être fait de façon à ce qu'une ligne ne contienne qu'une seule occurrence de chaque chiffre, qu'une colonne ne contienne qu'une seule occurrence de chaque chiffre, et qu'une région ne contienne qu'une seule occurrence de chaque chiffre.

1. Explication de la méthode de résolution simple du Sudoku

La méthode la plus simple permettant de résoudre certains Sudoku (niveau facile) est la suivante :

1. Calcul de l'*ensemble* des candidats potentiels de chaque case vide. Ainsi, pour chaque case vide appartenant à une ligne i, une colonne j et une région k :
   1- Calcul de l'*ensemble* I des éléments présents dans la ligne i.
   2- Calcul de l'*ensemble* J des éléments présents dans la colonne j.
   3- Calcul de l'*ensemble* K des éléments présents dans la région k.
   4- Calcul de l'*ensemble* des candidats impossibles : union des ensembles I, J et K.
   5- Enfin, l'*ensemble* des candidats potentiels correspond alors à la différence entre l'ensemble contenant les n premiers entiers et l'ensemble des candidats impossibles.
2. Si une case a un *ensemble* de candidats vide, on s'arrête car il y a une erreur dans la résolution.
3. Remplir chaque case n'ayant qu'un seul candidat potentiel avec ce candidat.
4. Recommencer à l'étape 1 (appel récursif) si le Sudoku n'est pas fini, c'est-à-dire si toutes les cases de la grille ne sont pas remplies.

2. Mise en œuvre en Scheme

En Scheme, un Sudoku sera représenté par une liste de n sous-listes. Chaque sous-liste contiendra les valeurs d'une ligne. On utilisera la valeur 0 pour une case vide. Ainsi la grille présentée précédemment pour n=9 sera représentée par la liste d'entiers suivante :

```
'((5 3 0 0 7 0 0 0 0)
  (6 0 0 1 9 5 0 0 0)
  (0 9 8 0 0 0 0 6 0)
  …)
```

1. Définir la grille de Sudoku S de l'exemple précédent et les variables globales `sudokuSize` et `regionSize` correspondant à la taille de la grille et à la taille d'une région, ainsi que la variable globale `Candidats` qui correspond à l'*ensemble* des candidats possibles pour une case de la grille du Sudoku (cela nécessite de définir une fonction utilisant `sudokuSize)`.

2. Ecrire la fonction `Ligne` qui, à partir d'un Sudoku S et d'un indice de ligne i, renvoie l'*ensemble* des valeurs des cases non-vides de la $i^{ème}$ ligne de S. Pour cela, il sera utile de définir une fonction permettant de récupérer le $i^{ème}$ élément d'une liste, ainsi qu'une fonction permettant de supprimer les éléments d'un *ensemble* correspondant à la représentation des cases vides de S.

3. Ecrire la fonction `Colonne` qui, à partir d'un Sudoku S et d'un indice de colonne i, renvoie l'*ensemble* des valeurs des cases non-vides de la $i^{ème}$ colonne de S.

4. Ecrire la fonction `IndiceRegion` qui, à partir d'un indice de ligne i et d'un indice de colonne j, renvoie l'indice de la région dans laquelle se trouve la case (i,j).

5. Ecrire la fonction `RegionIndice` qui, à partir d'un indice de région, renvoie l'indice de la ligne et de la colonne de la première case de la région.

6. Ecrire la fonction `Region` qui, à partir d'un Sudoku S et d'un indice de région i, renvoie l'*ensemble* des valeurs des cases non-vides de la $i^{ème}$ région de S. Pour cela, il sera utile de définir une fonction permettant de renvoyer les éléments d'une liste compris entre les indices n et m.

7. Ecrire la fonction `Possibles` qui, à partir d'une case de la grille du Sudoku, renvoie l'*ensemble* des valeurs possibles pour cette case. Si la case n'est pas vide, la fonction renvoie un ensemble réduit à la valeur de la case.

8. Ecrire la fonction `solve1step` qui, à partir d'un Sudoku S, effectue une itération de la méthode de résolution du Sudoku. L'idée est de remplacer dans un premier temps la valeur de chacune des cases de la grille du Sudoku (vide ou non) par l'*ensemble* des candidats possibles, puis si un seul candidat est possible, de retenir ce candidat, sinon de mettre la valeur 0 dans la case.

9. Enfin, écrire la fonction `Resoudre` qui, à partir d'un Sudoku S, renvoie le Sudoku après résolution.

3. Stratégies plus complexes pour aller plus loin (bonus)

1. *Méthode d'essais et erreurs* – Si aucune case ne présente qu'un seul candidat possible, traitez les cases qui possèdent deux solutions en testant l'une puis l'autre solution si nécessaire.

2. *Stratégie des chiffres exclusifs* – Si à l'intérieur d'une région r, des chiffres figurent uniquement dans la ligne l (resp. colonne c) et qu'ils ne figurent pas ailleurs dans la région, il est alors possible de supprimer ce chiffre sur la ligne l (resp. colonne c) des autres régions puisqu'ils seront obligatoirement positionnés sur la ligne l (resp. colonne c) de la région r.

3. *Stratégie des chiffres exclusifs dans une région* – Si à l'intérieur d'une région r, des chiffres figurent sur la ligne l (resp. colonne c) et qu'ils ne figurent pas ailleurs sur la ligne l (resp. colonne c) des autres régions, il est alors possible de supprimer ce chiffre sur les autres lignes de la région r, puisqu'ils seront obligatoirement positionnés sur la ligne l (resp. colonne c) de la région r.

4. *Stratégie des paires exclusives* – Si dans une région r se trouvent deux cases c1 et c2 dans lesquelles la même paire figure, alors on peut supprimer ces deux chiffres dans les autres cases de la région r, puisqu'ils seront obligatoirement mis dans les cases c1 et c2.

Implémentez en Scheme une ou plusieurs de ces stratégies permettant de traiter des grilles plus complexes telle que la grille suivante.

5				2		7		
			3		8			
4	2						8	3
	5			4	9			
9				1			4	
		1	8	5				
						3		
6		2	7					
					6	4		7

4. Évaluation du projet

À la fin de chacune des deux séances du TP projet (TP7 et TP9), vous devrez déposer sur TOMUSS votre code source commenté et testé au fur et à mesure.

Lors de la séance d'évaluation du TP projet (lundi 14 décembre), vous devrez faire une **démonstration** de votre programme à votre enseignant. Il vous demandera notamment d'expliquer les fonctions que vous avez implantées. Vous déposerez également sur TOMUSS la version finale de votre **code source qui doit être commenté**, et qui doit comporter des **appels pertinents aux différentes fonctions** afin de les tester.

Ce projet sera noté selon plusieurs critères :

- travail pendant les séances encadrées,
- présentation/démonstration du code réalisé,
- qualité du code (en particulier la complexité),
- commentaires et établissement de tests pertinents.