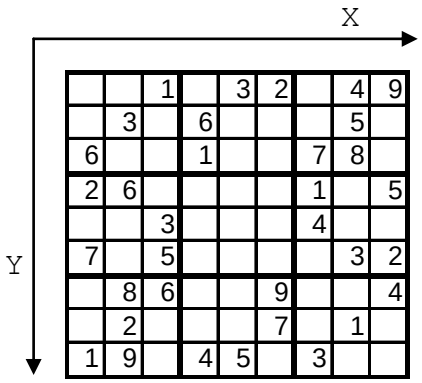
IA02 - FINAL P2014

Cours et TDs autorisés.

Le SUDOKU (15 points)

La grille de jeu est un carré de neuf cases de côté, subdivisé en autant de carrés identiques, appelés régions (voir Figure 1). Initialement, certaines cases contiennent un chiffre entre 1 et 9 tandis que d'autres cases sont vides. La règle du jeu est simple : il s'agit de remplir les cases de façon à ce que chaque ligne, colonne et région ne contiennent qu'une seule fois tous les chiffres de 1 à 9. Formulé autrement, chacun de ces ensembles doit contenir tous les chiffres de 1 à 9.



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

Figure 1 : une grille de Sudoku

Comme indiquée sur la Figure, les cases peuvent être identifiées par leurs coordonnées (X,Y).

Les lignes sont alors numérotées de haut en bas et les colonnes de gauche à droite.

```
[ [[x,x,1],[x,3,2],[x,4,9]],
  [[x,3,x],[6,x,x],[x,5,x]],
  [[6,x,x],[1,x,x],[7,8,x]],

  [[2,6,x],[x,x,x],[1,x,5]],

  [[x,x,3],[x,x,x],[4,x,x]],
  [[7,x,5],[x,x,x],[x,3,2]],

  [[x,8,6],[x,x,9],[x,x,4]],
  [[x,2,x],[x,x,7],[x,1,x]],
  [[1,9,x],[4,5,x],[3,x,x]]  ]
```

Figure 2 : une modélisation en PROLOG

On décide de représenter une grille de Sudoku en utilisant une liste de 9 éléments (voir Figure 2). Chaque élément représente une ligne complète de la grille. Chaque ligne est elle – même une liste de 3 éléments qui correspondent aux différentes régions que traverse la ligne. Chaque élément de la ligne est alors une liste de 3 chiffres entre 1 et 9. Si le chiffre n'est pas encore connu, il est remplacé par la lettre **x** (en minuscule !). Le but du problème suivant est d'écrire un ensemble de prédicats qui permet de manipuler une grille de Sudoku et de la résoudre.

Voici trois fonctions de {1, …, 9} dans {1,…,9} permettant de faire quelques calculs utiles. Les opérateurs « `mod` » et « `//` » sont respectivement les opérateurs « modulo » et « division entière » en prolog.

(N-1) mod 3 + 1
- associe aux valeurs N=1, 4 ou 7 la valeur 1
- associe aux valeurs N=2, 5 ou 8 la valeur 2
- associe aux valeurs N=3, 6 ou 9 la valeur 3

(N-1)//3 + 1
- associe aux valeurs N=1, 2 ou 3 la valeur 1
- associe aux valeurs N=4, 5 ou 6 la valeur 2
- associe aux valeurs N=7, 8 ou 9 la valeur 3

(N-1) //3 *3 + 1
- associe aux valeurs N=1, 2 ou 3 la valeur 1 ; associe aux valeurs N=4, 5 ou 6 la valeur 4 ;
- associe aux valeurs N=7, 8 ou 9 la valeur 7

Consignes : mis à part les prédicats de base (`write, nl, !, \+, =, \=, is, =:=, =\=, >=, ...`), on suppose qu'aucun prédicat n'est prédéfini. Tout autre prédicat utilisé devra être réécrit sauf indication contraire. Chaque question est indépendante : vous pouvez utiliser un prédicat d'une question (même sans y avoir répondu).

Partie I : Préliminaires (5 points)

1- (1 point) Écrire le prédicat **element(N,L,X)** qui unifie le Nième élément de la liste **L** avec la variable **X**. Ce prédicat doit aussi fonctionner en génération.

Ex : `element(3, [a,b,c,d,e], X).` renvoie `yes : X=c.`
`element(N, [a,b,c,d,e], X).` renvoie `yes : { N=1, X=a}; { N=2, X=b}; ... { N=5, X=e};`

2- (1 point) Écrire le prédicat **retirer_element_n(N,L,NL)** qui unifie **NL** avec la liste obtenue en enlevant le **N**e élément de la liste **L**.

Ex : `retirer_element_n(3, [a,b,c,d,e], NL).` renvoie `yes : NL=[a,b,d,e].`

3- (1 point) Écrire le prédicat **retirer_element_x(X,L,NL)** qui unifie avec la variable **NL** la liste obtenue en retirant l'élément unifié avec **X** de la liste **L**. Si l'élément **X** n'appartient pas à **L**, alors le prédicat n'échouera pas et **NL=L**. On suppose qu'il ne peut pas y avoir plusieurs occurrences de **X**.

Ex : `retirer_element_x(3,[1,2,3,4],NL).` renvoie `NL=[1,2,4].`
`retirer_element_x(8,[1,2,3,4],NL).` renvoie `NL=[1,2,3,4].`

4- (1 point) Écrire le prédicat **retirer_liste(E,L,NL)** qui unifie **NL** avec la liste **L** auquel on a retiré tous les éléments de la liste **E** (s'ils existent). On pourra utiliser le prédicat précédent.

Ex : `retirer_liste([3,2,5],[1,2,3,4],NL).` renvoie `yes : NL=[1,4].`

5- (1 point) Écrire le prédicat **set_element_n(N,L,X,NL)** qui remplace le **N**e élément de la liste **L** par l'élément unifié avec **X** et qui unifie le résultat avec **NL**.

Ex : `set_element_n(3,[1,2,3,4,5],7,NL).` renvoie `yes : NL=[1,2,7,4,5].`

Partie II : Les Possibles (4 points)

1- (1 point) Écrire le prédicat **get_ligne(N,G,L)** qui unifie la **N**e ligne d'une grille de Sudoku **G** dans la variable **L**. Les niveaux de crochets devront être enlevés (tous les éléments seront au même niveau).

Ex : `get_ligne(3,G,L).` renvoie `L=[6,x,x,1,x,x,7,8,x]` si `G` est unifiée avec l'exemple de la figure 2.

2- (1 point) Écrire le prédicat **get_colonne(N,G,C)** qui unifie la **N**e colonne d'une grille de Sudoku **G** dans la variable **C**. Les niveaux de crochets devront être enlevés (tous les éléments seront au même niveau).

Ex : `get_colonne(4,G,C).` renvoie `C=[x,6,1,x,x,x,x,x,4]` si `G` est unifiée avec l'exemple de la figure 2.

On suppose que le prédicat **get_region(X,Y,G,R)** qui unifie avec **R** une liste contenant tous les éléments (au même niveau) de la région à laquelle la case **(X,Y)** appartient est implémenté (vous n'avez pas à le refaire…).

Ex : `get_region(3,5,G,L).` renvoie `L=[2,6,x,x,x,3,x,7,5]` si `G` est unifiée avec l'exemple de la figure 2.

4- (2 points) Ecrire le prédicat **get_possibles(X,Y,G,P)** qui unifie avec **P** une liste qui contient toutes les valeurs que pourrait prendre la case de coordonnées **(X,Y)** compte-tenue de la ligne, la colonne et la région auxquelles cette case appartient. La valeur de cette case pourra être un chiffre ou un **x**. Si c'est une case **x**, il s'agira de savoir quelles sont les valeurs que peut prendre cette case. Dans le cas d'une case qui est un chiffre, cela nous permettra de savoir si cette valeur est possible (c'est le cas si elle appartient à la liste **P**) et donc de savoir si cette affectation est valide.

Ex : `get_possibles(5,4,G,L).` renvoie `L=[4,7,8,9]` si `G` est unifiée avec l'exemple de la figure 2.
`get_possibles(9,3,G,L).` renvoie `L=[3]` si `G` est unifiée avec l'exemple de la figure 2.
`get_possibles(2,4,G,L).` renvoie `L=[4,6]` si `G` est unifiée avec l'exemple de la figure 2.

Attention : on prendra grand soin à ne pas enlever de la liste des possibles l'élément de la case **(X,Y)** lui-même. Ainsi si on considère l'élément **(2,4)**, tous les éléments de la ligne **4**, c'est-à-dire : **[2,6,x,x,x,x,1,x,5]** (en ignorant les **x**) devraient être enlevés de la liste des possibles sauf le chiffre 6 qui est le deuxième élément de cette ligne donc l'élément de la case **(X,Y)** lui-même… On rappelle que « retirer_liste » n'échoue pas si on essaye d'enlever de la liste un élément qui n'existe (comme un **x** ou un chiffre que l'on aurait déjà enlevé par exemple…).

Partie III : Résolution (6 points)

1- (0.5 point) Ecrire le prédicat **get_valeur(X,Y,G,V)** qui unifie la valeur de la case **(X,Y)** d'une grille de Sudoku **G** dans la variable **V**. Cette valeur peut être un chiffre ou **x**. Ce prédicat pourra fonctionner en génération.

Ex : `get_valeur(6,7,G,V).` renvoie V=9 si G est unifiée avec l'exemple de la figure 2.
`get_valeur(X,Y,G,V).` renvoie {X=1, Y=1, V=x}; {X=1, Y=2, V=x}; {X=1, Y=2, V=6}; ... si G est unifiée avec l'exemple de la figure 2.

2- (1 point) Ecrire le prédicat **set_valeur(X,Y,G,V,NG)** qui change la valeur de la case **(X,Y)** d'une grille de Sudoku **G** avec la valeur unifiée avec **V** et qui unifie le résultat avec la variable **NG**.

3- (1 point) On dit qu'une **case est valide** si la liste des possibles de cette case est non vide et si la valeur de cette case est soit **x,** soit un chiffre appartenant à la liste des possibles. Ecrire le prédicat **valide(X,Y,G)** qui s'efface si la case **(X,Y)** d'une grille de Sudoku **G** est valide.

4- (1 point) On dit qu'une **grille est valide** si toutes ses cases sont valides. Ecrire le prédicat **valide(G)** qui s'efface si une grille de Sudoku **G** est valide.

Indice : une grille est valide s'il n'existe pas de case non valide. On pourra donc pour y arriver, écrire un prédicat qui cherche une case non valide.

5- (0.5 point) Ecrire le prédicat **get_coord_x(X,Y,G)** qui unifie les variables **X** et **Y** avec les coordonnées d'une case dont la valeur est **x**. Ce prédicat devra être capable de générer toutes les solutions possibles.

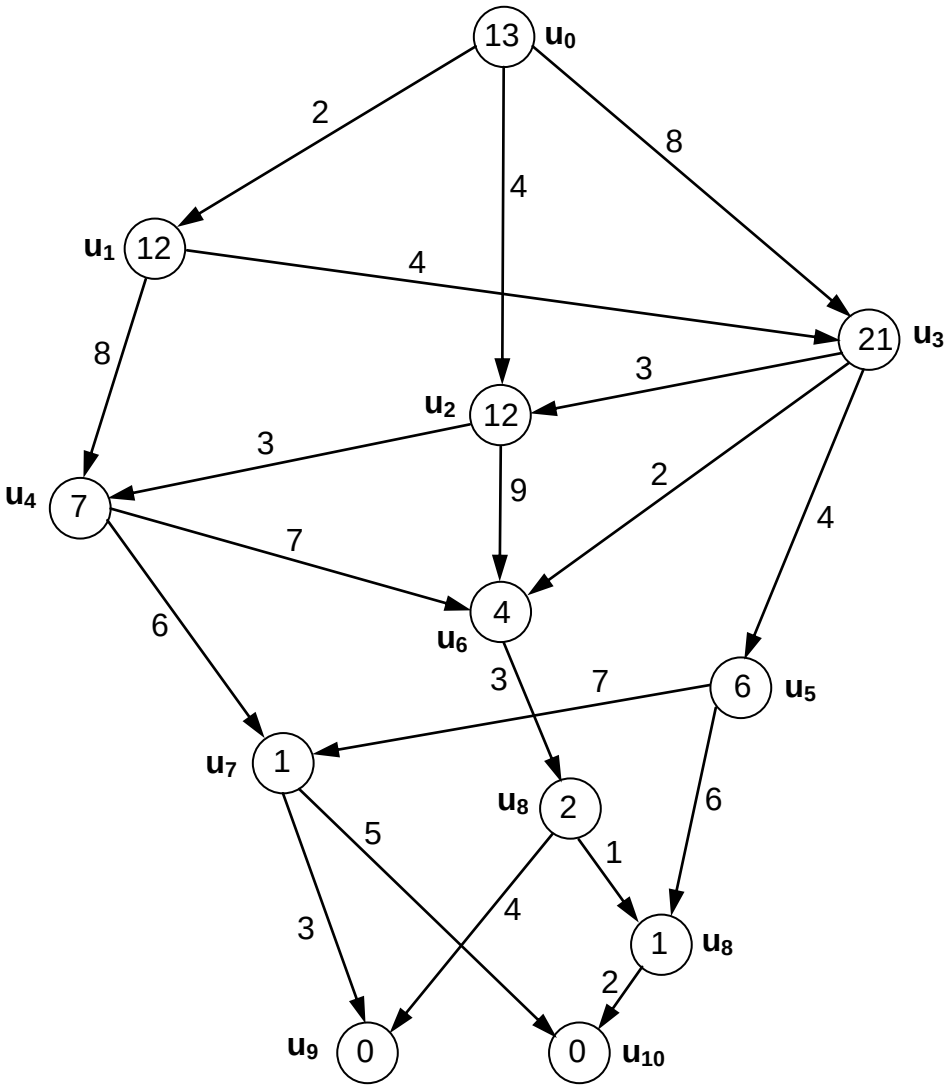
Ex : `get_coord_x(X,Y,G).` renvoie {X=1, Y=1 }; {X=1, Y=2}; {X=1, Y=5}; ... si G est unifiée avec l'exemple de la figure 2.

6- (0.5 point) Ecrire le prédicat **resolue(G)** qui s'efface si la grille de Sudoku **G** est valide et si toutes les cases ont une valeur qui est un chiffre (pas **x** !).

7- (1.5 point) Ecrire le prédicat **resoudre(G,NG)** qui résout (cherche un chiffre valide pour chaque case qui contient un **x**) la grille G de Sudoku et qui unifie le résultat avec **NG**.

Chemin (5 pts)

Dans le graphe d'états suivant, u_0 est l'état ou nœud initial, u_9 et u_{10} sont les états finaux ou nœuds solutions. La valeur de l'heuristique h(n) est précisée en chaque nœud n, le coût de chaque opérateur est indiqué directement sur chaque flèche.



Question 1 :

1.1 Appliquer l'algorithme A à ce graphe, évidemment jusqu'à ce que la condition de terminaison soit satisfaite. Vous développerez donc simultanément un graphe de recherche et un arbre de recherche, en partant bien sûr du nœud u_0. Vous préciserez lors de chaque expansion, exactement comme en cours et TD, les valeurs de `d` et de `h`, ainsi que le numéro d'ordre de chaque expansion. `(2 points)`

`Rq` **:** pour les flèches de l'arbre de recherche, vous pouvez éventuellement utiliser une couleur différente. Par ailleurs, quand vous êtes amené à changer la valeur de `d` ou à modifier l'emplacement d'une flèche de cet arbre, faites en sorte que ce changement soit bien visible.

1.2 Quel chemin avez-vous trouvé ? Quel est son coût ? Est-il optimal ? `(1 point)`

Question 2 :

2.1 Montrez que l'heuristique `h` n'est pas minorante. Indication : l'un des nœuds devrait attirer votre attention. Puis modifiez la valeur de `h` en ce même nœud de façon à obtenir une heuristique minorante. On appelle `h'` cette nouvelle heuristique. `(1 point)`

2.2 Appliquez alors l'algorithme A* muni de `h'`. Qu'obtenez-vous? `(1 point)`