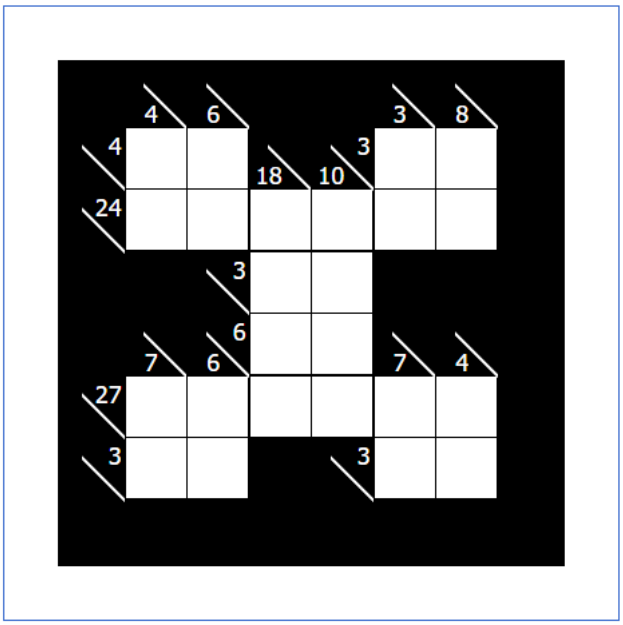
PROJET EN PROLOG

NGUYEN Viet 20006303

Mars 2021

Ce devoir est réalisé dans le carde du cours de
Programmation Déclarative et base de données.



Licence Informatique – Département STN

Première année

Université Paris 8

2020-2021

1. Présentation du Projet

a) *Introduction :*

- Le projet consiste à créer un programme en prolog permettant de résoudre une grille de Kakuro (puzzle game du Japon).

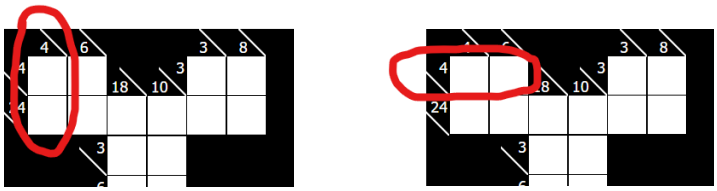
b) *Règle du Kakuro :*

- Nous avons une blanche carrée avec grille en n*n

- Cette planche carrée se compose de nombreuses parties :

+) La partie blanche vierge du tableau est utilisée pour la saisie des nombres.

+) La partie noire cache les carrés inutiles, les carrés contiennent "\" et les nombres aléatoires représentent les lignes et les colonnes du tableau. Par exemple, afficher les nombres pour les lignes et les colonnes:

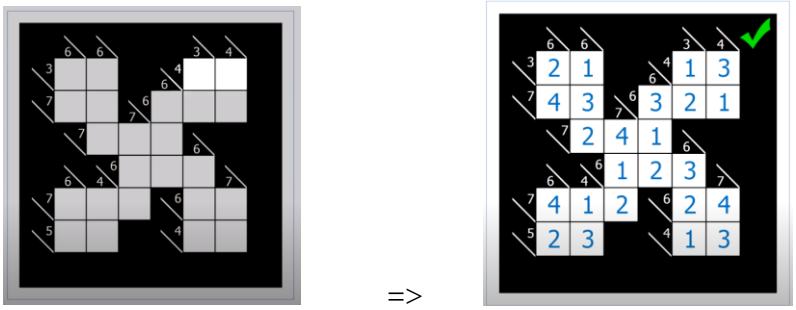


- Chaque cellule peut contenir des nombres de 1 à 9.

- Les indices dans les cellules noires indiquent la somme des nombres à côté de cet indice. (à droite ou en bas).

- Les nombres dans les cellules blanches consécutives doivent être uniques.

Par exemple, voici un Kakuro 6x6 facile :



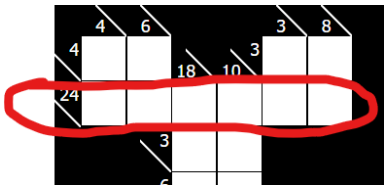
J'ai utilisé le site https://www.puzzle-kakuro.com/ pour trouver nos énoncés de puzzles et essayer les solutions que propose notre programme.

2. Un programme qui résout une form 6x6

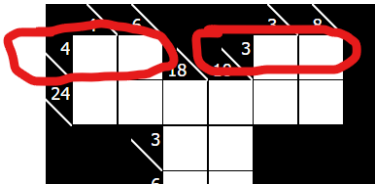
a) *Pour extraire les lignes du champ :*

```
/*Pour extraire les lignes du tableau*/
ex_lignes([], [], X, Y) :- maplist(reverse, X, Y).
ex_lignes([], Y, X, Z) :-
    not(Y = []),
    maplist(reverse, [Y | X], Z).
ex_lignes([X | Xs], ligne, acc, lignes) :-
    ground(X),
    (
        X = x ->
        ex_lignes(Xs, [], [ligne | acc], lignes)
    ;
    ex_lignes(Xs, [X], [ligne | acc], lignes)
    ).
ex_lignes([X | Xs], ligne, acc, lignes) :-
    var(X),
    ex_lignes(Xs, [X | ligne], acc, lignes).
```

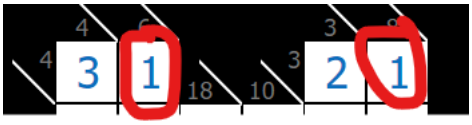
- En kakuro, il y a plusieurs carrés horizontaux sur la même rangée, mais ils seront considérés comme deux rangées différentes s'ils ne sont pas réunis comme sur l'image :



Donc, dans ce cas, ils sont comptés comme deux lignes



et nous n'avons pas à nous soucier de répéter le même nombre sur la même ligne.

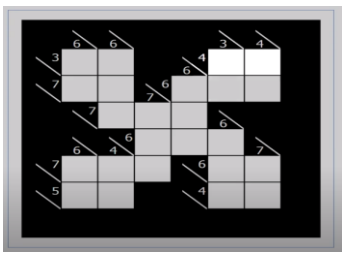


Et il en va de même pour certaines colonnes verticales dans tout le tableau.

b) *La form du tableau dans le jeu :*

```
tableau([
    [x  , 6/x, 6/x, x  , x  , 3/x, 4/x],
    [x/4, A  , B  , x  , 6/4, C  , D  ],
    [x/7, E  , F  , 7/6, H  , J  , K  ],
    [x  , x/7, L  , M  , N  , 6/x, x  ],
    [x  , 6/x, 4/6, O  , P  , Q  , 7/x],
    [x/7, R  , S  , T  , x/6, U  , V  ],
    [x/5, W  , X  , x  , x/4, Y  , Z  ]
]).
```

➔



```
/*Pour extraire les lignes du tableau*/
ex_lignes([], [], X, Y) :- maplist(reverse, X, Y).
ex_lignes([], Y, X, Z) :- not(Y = []), maplist(reverse, [Y | X], Z).
ex_lignes([X | Xs], ligne, acc, lignes) :- ground(X),
   (
      X = x ->
      ex_lignes(Xs, [], [ligne | acc], lignes)
   ;
   ex_lignes(Xs, [X], [ligne | acc], lignes)
   ).
ex_lignes([X | Xs], ligne, acc, lignes) :- var(X), ex_lignes(Xs, [X | ligne], acc, lignes).

/*Pour normaliser les lignes*/
changer_ligne([], X, Y) :- reverse(X, Y).
changer_ligne([X | Xs], Ys, Zs) :- ground(X), X = x/Y, changer_ligne(Xs, [Y | Ys], Zs).
changer_ligne([X | Xs], Ys, Zs) :-
   var(X),
   changer_ligne(Xs, [X | Ys], Zs).
changer_ligne(X, Y) :- changer_ligne(X, [], Y).

retourne([], Ys, Zs) :- reverse(Ys, Zs).
retourne([X | Xs], Ys, Zs) :-(var(X), ! ; X = x),retourne(Xs, [X | Ys], Zs).
retourne([X | Xs], Ys, Zs) :- ground(X),X = N/Y,retourne(Xs, [Y/N | Ys], Zs).
retourne(X, Y) :- retourne(X, [], Y).

mrg([], X, X).
mrg([X | Xs], Y, Z) :-mrg(Xs, [X | Y], Z).

changer([], X, X).
changer([E | Es], acc, res) :-
   (
      changer_ligne(E, NE) ->
      changer(Es, [NE | acc], res)
   ;
   changer(Es, acc, res)
   ).

lignes([], X, X).
lignes([ligne | lignes], acc, res) :-
```

```
   ex_lignes(ligne, [], [], extrait),
   (
      changer(extrait, [], change) ->
      (
         mrg(change, acc, Macc),
         lignes(lignes, Macc, res)
      )
   ;
   lignes(lignes, acc, res)
   ).

resoudre_somme([]).
resoudre_somme([somme | sommations]) :-
tout_distinct(sommations),somme(sommations, =, somme).

resoudre_sommes([]).
resoudre_sommes([X | Xs]) :- resoudre_somme(X), resoudre_sommes(Xs).

resoudre(tableau) :-
   term_variables(tableau, variables),
   lignes(tableau, [], lignes),
   change_pos(tableau, Ttableau),
   maplist(retourne, Ttableau, Rcolonne),
   lignes(Rcolonne, [], colonne),
   resoudre_sommes(lignes),
   resoudre_sommes(colonne),
   label(variables).

tableau([
   [x  , 6/x, 6/x, x  , x  , 3/x, 4/x],
   [x/4, A  , B  , x  , 6/4, C  , D  ],
   [x/7, E  , F  , 7/6, H  , J  , K  ],
   [x  , x/7, L  , M  , N  , 6/x, x  ],
   [x  , 6/x, 4/6, O  , P  , Q  , 7/x],
   [x/7, R  , S  , T  , x/6, U  , V  ],
   [x/5, W  , X  , x  , x/4, Y  , Z  ]
]).

kakuro :-
   tableau(tableau),
   resoudre(tableau),
```

```
format('bingo', [tableau]).
```