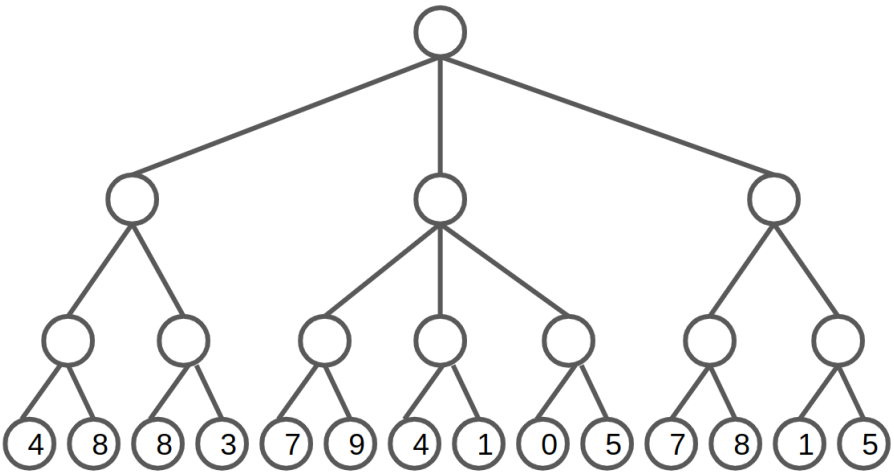
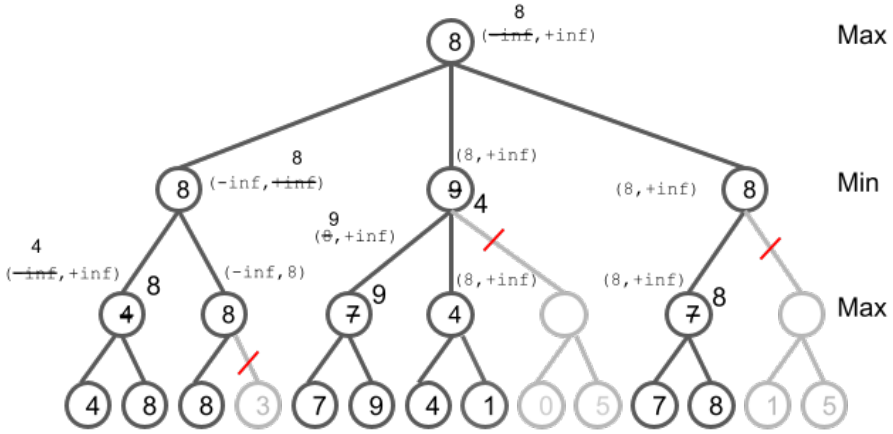
Algorithmes pour l'IA – TD 2
Minimax AlphaBeta + Problèmes de satisfaction de contraintes

T1: Appliquez l'algorithme minimax avec élagage AlphaBeta sur l'arbre suivant. Les valeurs aux feuilles correspondent aux valeurs de la fonction d'évaluation pour les états du jeu correspondants. Le joueur Max commence.

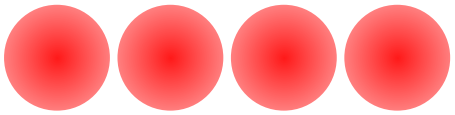


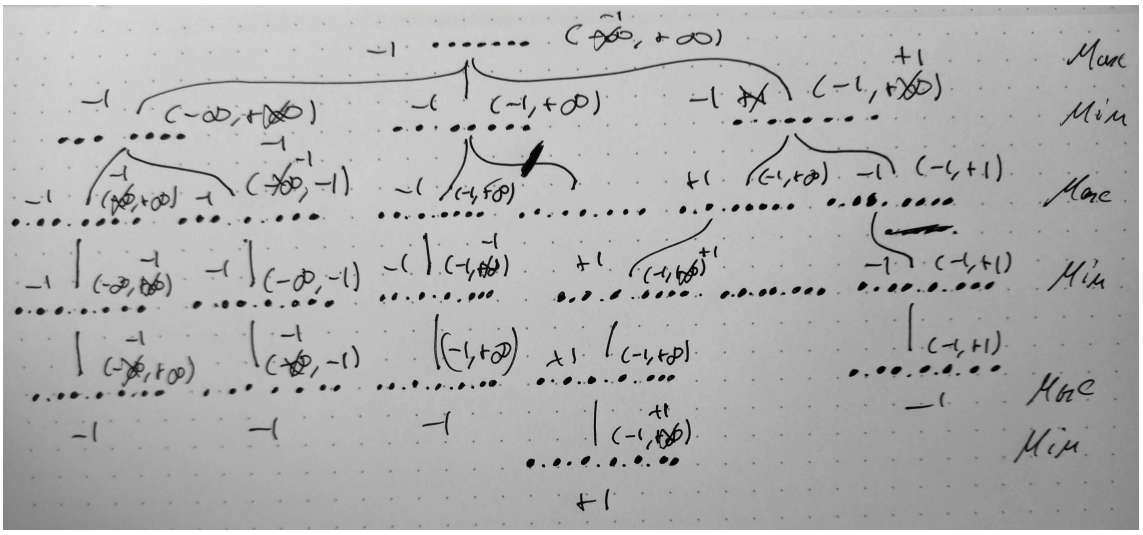


T2: On considère le jeu à deux joueurs suivant : On commence avec une pile de 7 jetons. Pendant le jeu, plusieurs piles vont être créées. Chaque joueur doit diviser une pile en deux piles non vides et de tailles différentes. S'il ne peut plus jouer, le joueur a perdu.

On évalue à +1 la valeur d'une position de victoire pour le joueur max, et -1 une position de victoire pour le joueur min.

1. Appliquez l'algorithme MiniMax avec élagage AlphaBeta à ce jeu sachant que le joueur Max commence.
2. Qui gagne ce jeu ?

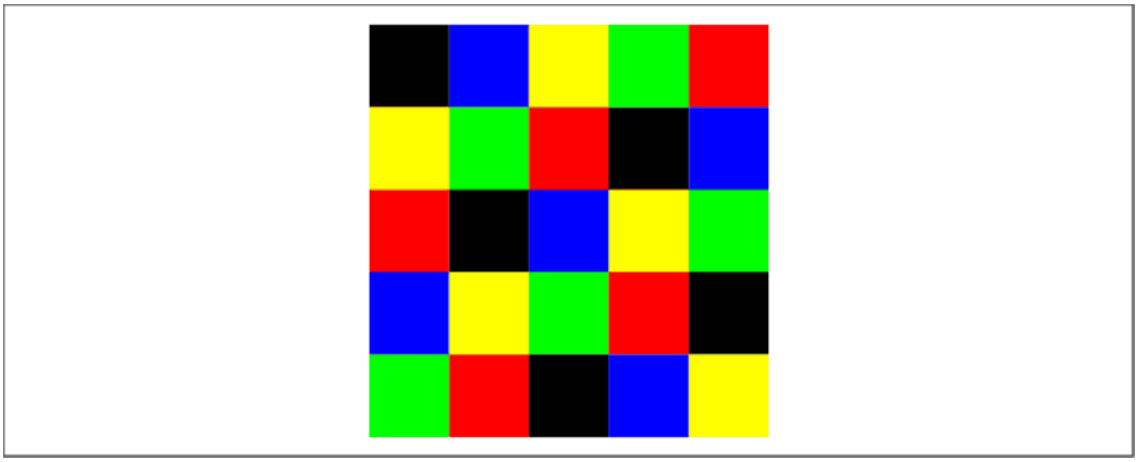




T3: Étant donnée n couleurs, un carré latin d'ordre n est un carré n x n colorié tel que :

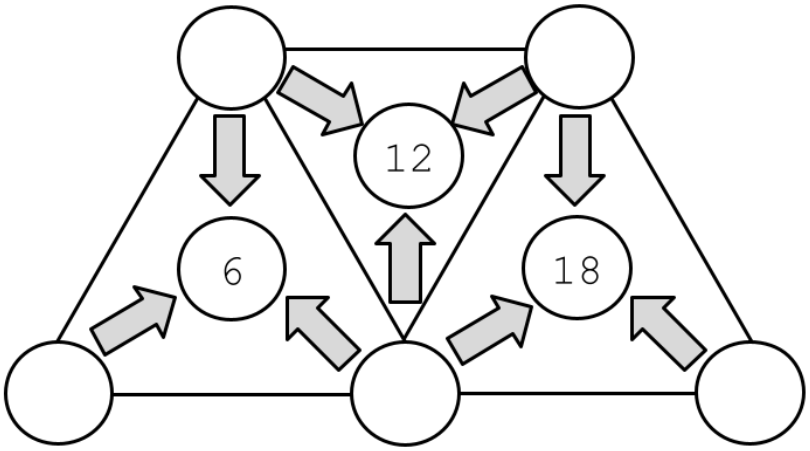
- Toute cellule est coloriée
- Chaque couleur apparaît exactement une fois sur chaque ligne
- Chaque couleur apparaît exactement une fois sur chaque colonne

Modélisez ce problème comme un problème de satisfaction de contrainte.



- **Variables :** X_{ij} pour chaque ligne $i \in [1, n]$ et colonne $j \in [1, n]$
- **Domaines :** nombre entier $c \in [1, n]$ correspondant aux couleurs
- **Contraintes :**
  - pour tout $i \in [1, n]$, $X_{i1} \neq X_{i2} \neq \ldots \neq X_{in}$
  - pour tout $j \in [1, n]$, $X_{1j} \neq X_{2j} \neq \ldots \neq X_{nj}$

T4: Modélisez et résolvez en utilisant l'algorithme d'anticipation le problème suivant :



Trouver les nombres (entiers>0, tous différents) aux sommets des triangles pour obtenir les sommes aux centres des triangles.

- **Variables :** Les sommets des triangles : $X1, X2, X3, X4, X5$
- **Domaines :** nombre entier $s \in [1, 18]$ (aucun ne peut être plus grand que 18)
- **Contraintes :**
  - $X1 \neq X2 \neq X3 \neq X4 \neq X5$
  - $X1 + X2 + X3 = 6$
  - $X2 + X3 + X4 = 12$
  - $X3 + X4 + X5 = 18$

Si on utilise l'heuristique consistant à prendre la variable la plus contrainte pour le choix de la variable à affecter en premier, on prend $X3$. On aura ensuite le choix entre $X2$ et $X4$. On choisira $X4$ ici, pour rendre les choses plus difficiles.

Pour le choix de la valeur, on prend la première dans la liste.

Pour voir comment l'algorithme se déroule, on utilisera ici une méthode trop simple pour calculer les valeurs possibles restantes (directement à partir d'une affectation ou d'une valeur restantes pour d'autres variables).

1. $X3 = 1; X1 \in [2, 3]; X2 \in [2, 3]; X4 \in [2, 9]; X5 \in [2, 15]$
2. $X4 = 2; X1 \in [3]; X2 \in []; X5 \in [4, 15]$ retour à la ligne 1

3. $X4 = 3; X1 \in [2]; X2 \in []; X5 \in [4, 14]$ retour à la ligne 1
4. $X4 = 4; X1 \in [2, 3]; X2 \in [2, 3]; X5 \in [2, 3] \cup [5, 13]$
5. $X2 = 2; X1 \in [3]; X5 \in [5, 13]$ retour à la ligne 4
6. $X2 = 3; X1 \in [2]; X5 \in [5, 13]$ retour à la ligne 4
7. $X4 = 5; X1 \in [2, 3]; X2 \in [2, 3]; X5 \in [2, 4] \cup [6, 12]$
8. ...
9. $X4 = 8; X1 \in [2, 3]; X2 \in [2, 3]; X5 \in [2, 3] \cup [5, 9]$
10. $X2 = 2; X1 \in [3]; X5 \in [3] \cup [5, 9]$ retour à la ligne 9
11. $X2 = 3; X1 \in [2]; X5 \in [3] \cup [5, 9]$
12. $X1 = 2; X5 \in [5, 9]$
13. $X5 = 5$; retour à la ligne 12
14. ...
15. $X5 = 9$

Si on avait propagé les valeurs possible correctement, ce qui est plus dur, on aurait eu comme première ligne:

1. $X3 = 1; X1 \in [2, 3]; X2 \in [2, 3]; X4 \in [8, 9]; X5 \in [8, 9]$

et on aurait finit bien plus vite.

T5: Résoudre au moins un des problèmes ci-dessus (si carré magique, avec $n = 4$) en utilisant OR-tools.

Carré Latin :

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

carre = []
for i in range(1,5):
  carre.append([])
  for j in range(1,5):
    carre[i-1].append(model.NewIntVar(1,4,'X'+str(i)+str(j)))

for i in range(1,5):
    model.AddAllDifferent(carre[i-1])

carreT = []

for i in range(1,5):
  carreT.append([])
```

```
  for j in range(1,5):
    carreT[i-1].append(carre[j-1][i-1])

for i in range(1,5):
    model.AddAllDifferent(carreT[i-1])

solver = cp_model.CpSolver()
status = solver.Solve(model)

if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
  for i in range(1,5):
    print("")
    for j in range(1,5):
      print(solver.Value(carre[i-1][j-1]), " ", end="")
```

Résultat :

```
3  2  4  1
1  4  2  3
4  1  3  2
2  3  1  4
```

Sommes :

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

X1 = model.NewIntVar(1, 18, 'X1')
X2 = model.NewIntVar(1, 18, 'X2')
X3 = model.NewIntVar(1, 18, 'X3')
X4 = model.NewIntVar(1, 18, 'X4')
X5 = model.NewIntVar(1, 18, 'X5')

model.AddAllDifferent([X1,X2,X3,X4,X5])

model.Add(X1 + X2 + X3 == 6)
model.Add(X2 + X3 + X4 == 12)
model.Add(X3 + X4 + X5 == 18)

solver = cp_model.CpSolver()
status = solver.Solve(model)

if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
    print(f"X1={solver.Value(X1)}")
    print(f"X2={solver.Value(X2)}")
    print(f"X3={solver.Value(X3)}")
    print(f"X4={solver.Value(X4)}")
    print(f"X5={solver.Value(X5)}")
```

Résultat :

```
X1=2
X2=1
X3=3
X4=8
X5=7
```