

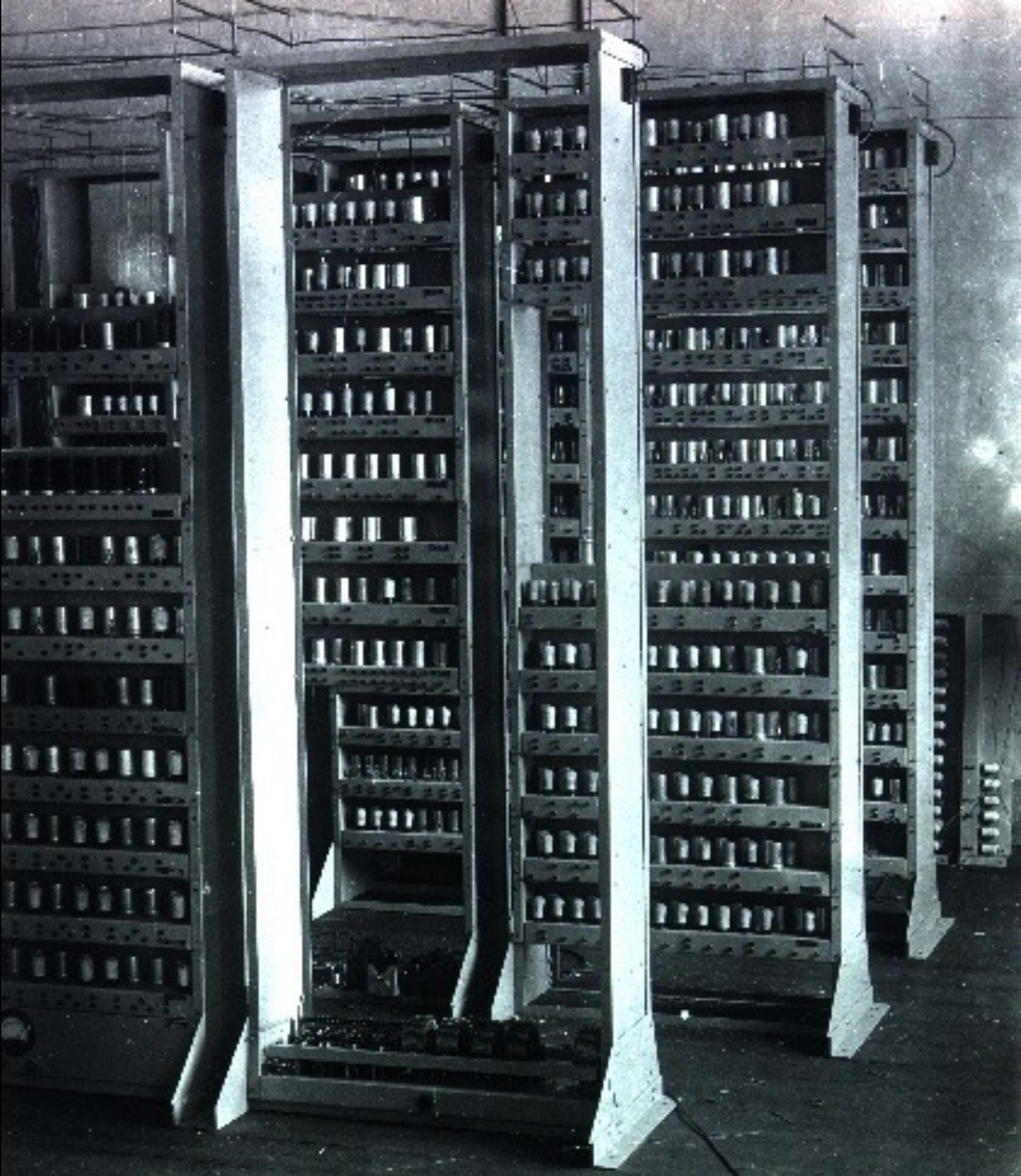
## 2.5. Minimax



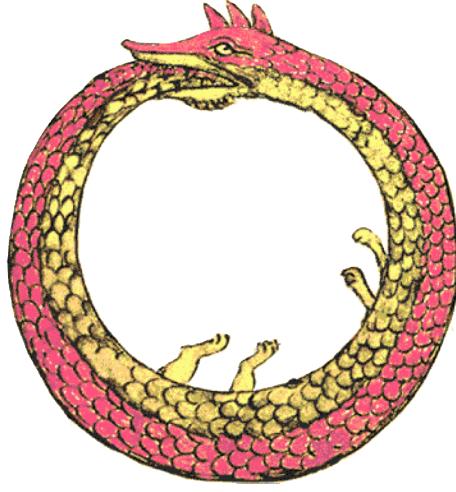
# Tic Tac Toe



- Premier jeu sur ordinateur en 1952
- Electronic Delay Storage Automatic Calculator (EDSAC)



# Principe de l'algorithme

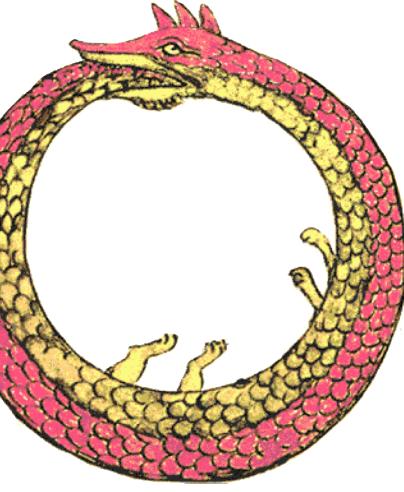


Pour un coup donné :

Prédire en simulant récursivement qui gagnera, si chaque joueur joue parfaitement.

- 
- └ je joue parfaitement
  - └ l'adversaire joue parfaitement
  - └ je joue parfaitement
  - └ l'adversaire joue parfaitement
  - └ je joue parfaitement
  - ...
  - Jusqu'à *fin de partie***

Jouer parfaitement ? Jouer le coup qui maximise mon gain ou minimise ma perte.



# Idée de l'algorithme

Pour *calculer qui gagne*, pour un coup donné :

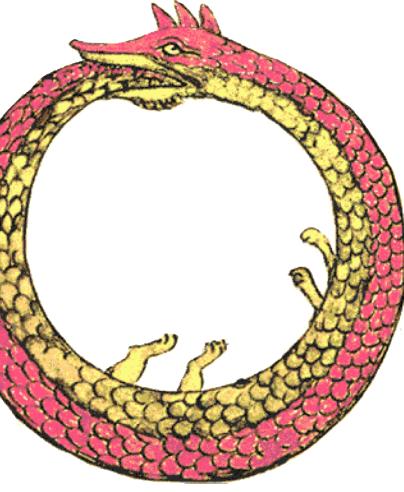
Jouer ce coup en tant que **moi**.

Calculer le meilleur coup à jouer pour **l'autre**.

]

*Simulation du meilleur coup de l'autre.*

Retourner qui gagne si l'autre joue ce coup.



# Idée de l'algorithme

Pour *calculer qui gagne*, pour un coup donné :

Jouer ce coup en tant que **moi**.

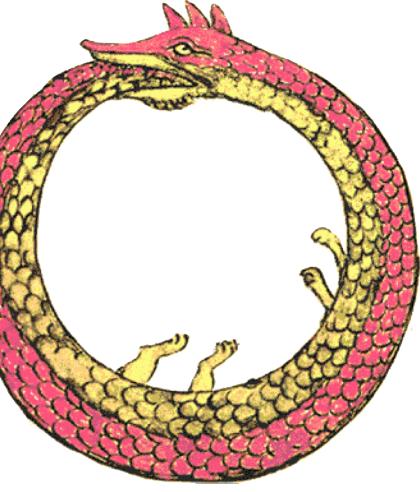
Pour chaque coup que **l'autre** peut faire

*Calculer qui gagne* si l'autre joue ce coup.

Noter le coup qui favorise **l'autre**.

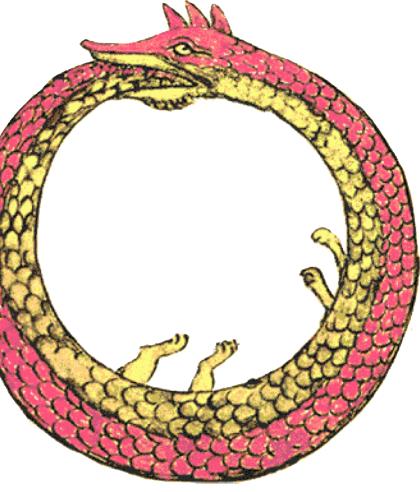
*Simulation du meilleur coup de l'autre.*

Retourner qui gagne si l'autre joue ce coup.



# Tic Tac Toe: principe de l'algorithme

- Comment choisir le prochain coup du joueur X (resp. O)?
  - donner un score à chaque case jouable
  - choisir la case avec le meilleur score
- Comment donner un score à une case?
  - 2 cas triviaux arrêtent la récursion
    - 3 X ou 3 O alignés : victoire -> score positif
    - 9 cases remplies : match nul -> score nul
  - Le cas général :
    - L'adversaire joue le tour qui maximise ses chances, récursivement
    - Nos objectifs sont opposés. Sa victoire est ma défaite et vice-versa.



**fonction calculeScore( case, joueur )**

marquer la case

si la grille est gagnante pour joueur, alors

score  $\leftarrow +1$

sinon si la grille est pleine, alors

score  $\leftarrow 0$

sinon

scoreAdverse  $\leftarrow -\infty$

pour toute case vide c

scoreAdverse  $\leftarrow \max(scoreAdverse,$   
*calculeScore(c,adversaire))*

fin pour

score  $\leftarrow -1 * scoreAdverse$

fin si

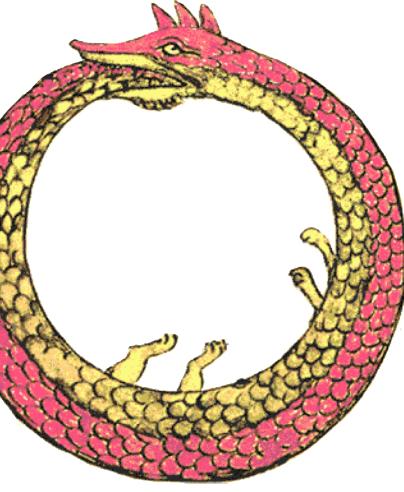
effacer la marque dans la case

retourner score

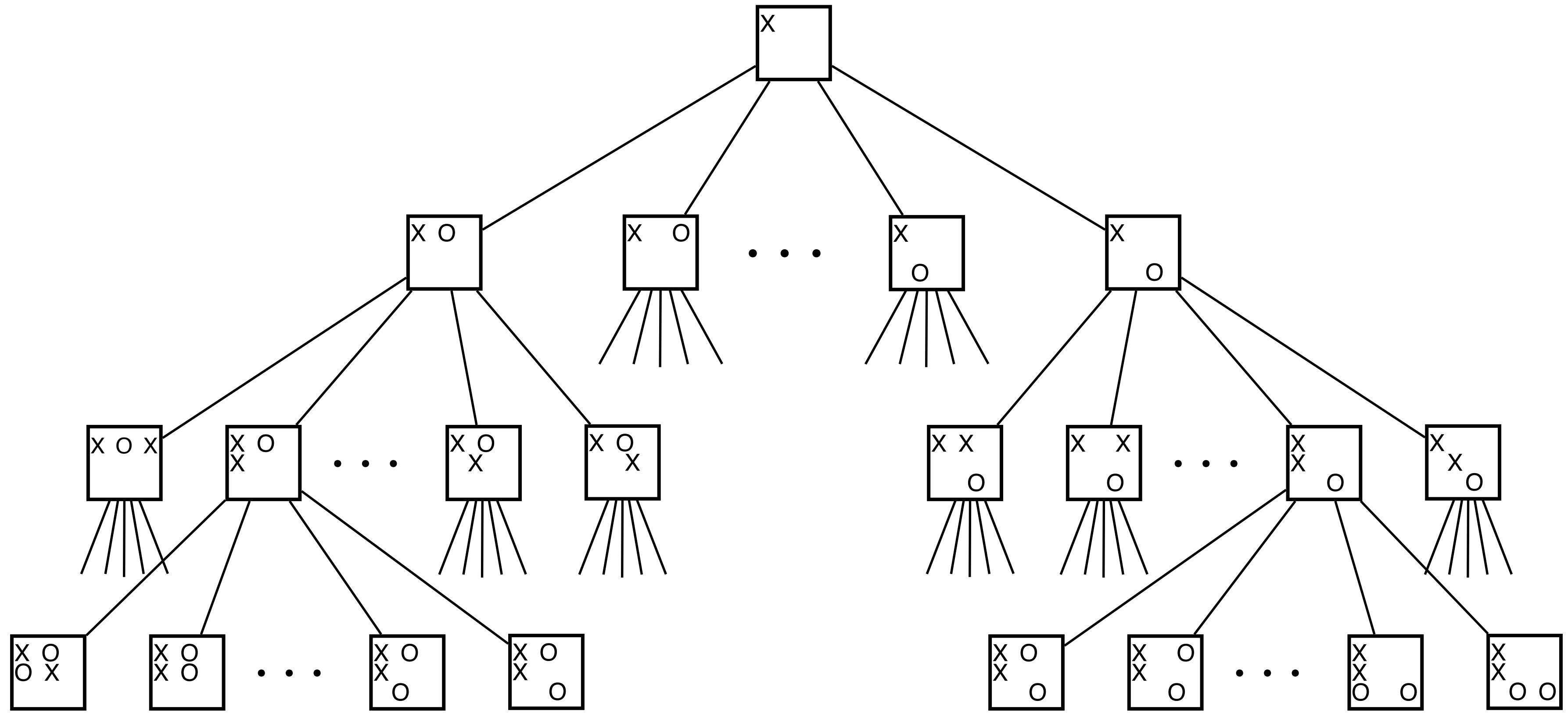
Cas triviaux

Cas général

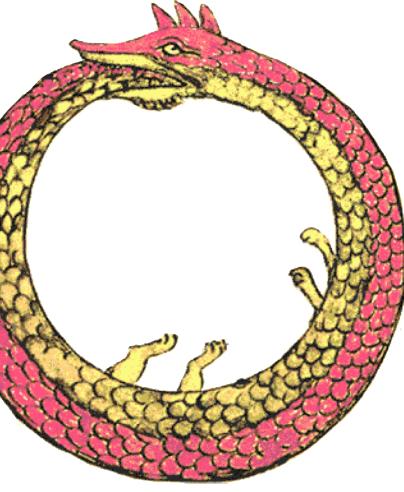
Voir algorithme permutations



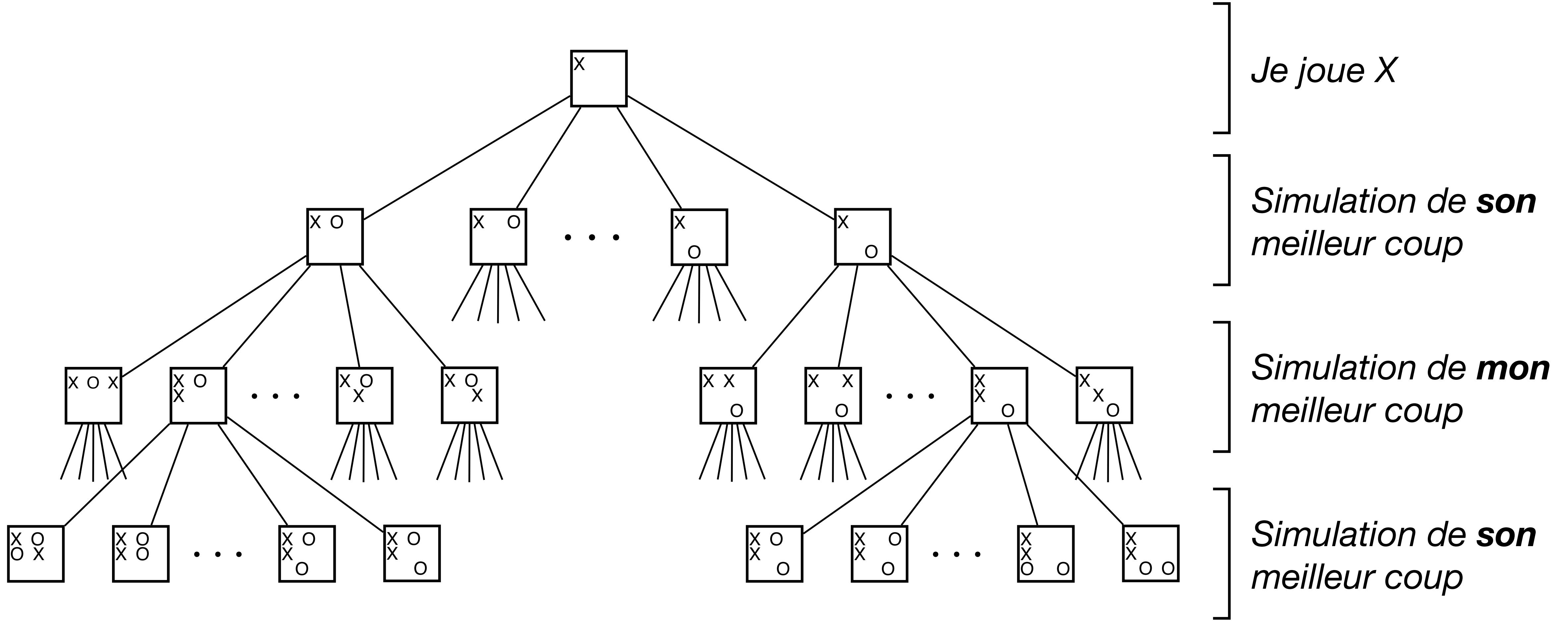
# Pourquoi “minimax” ?

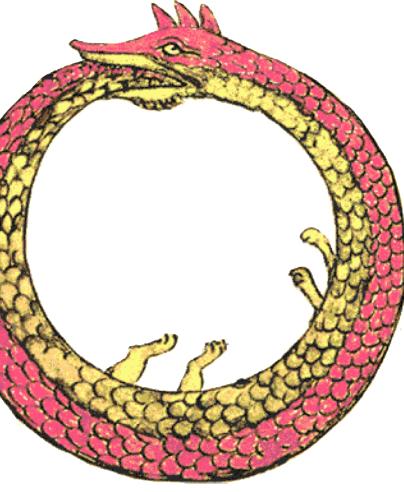


...

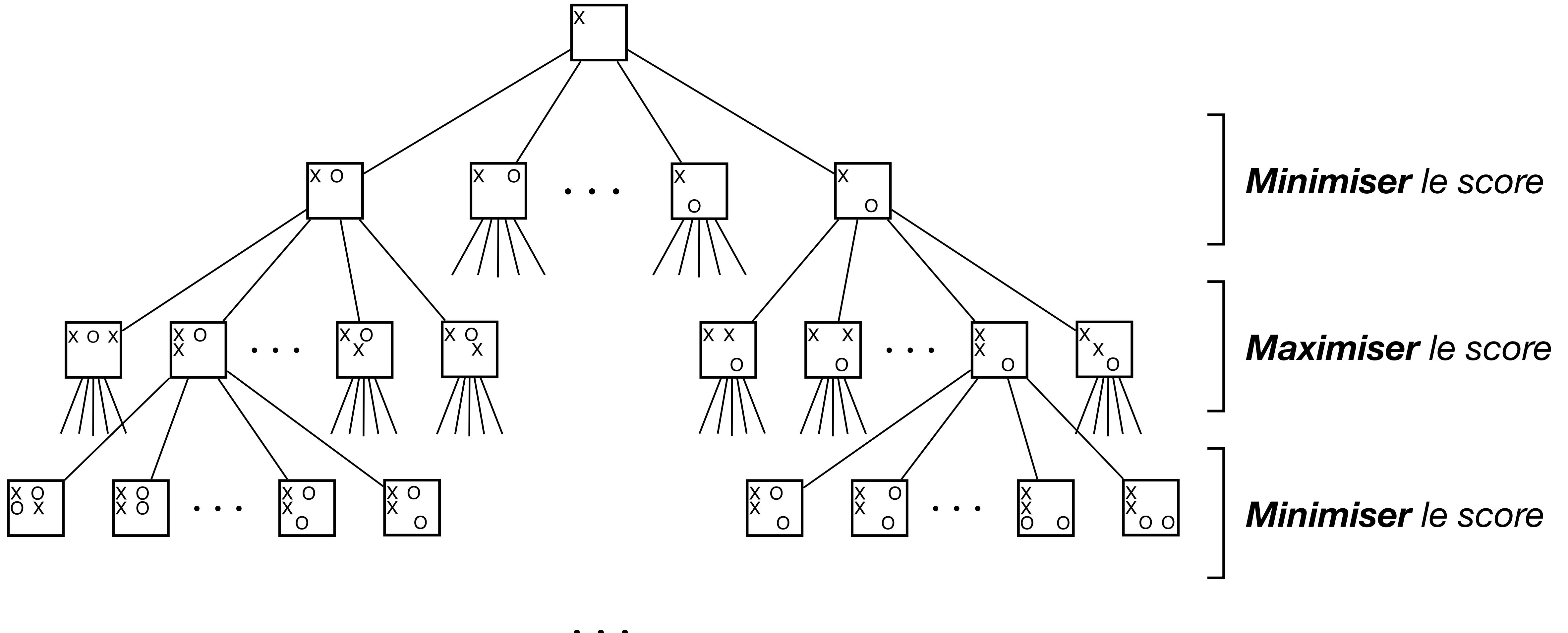


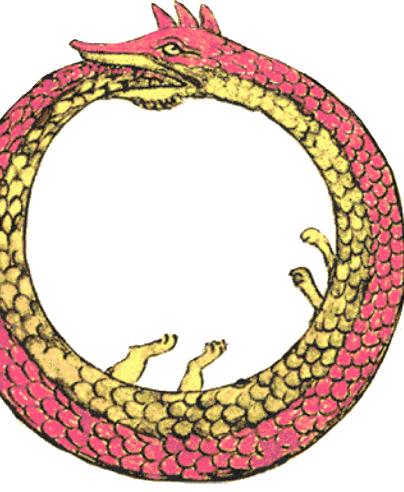
# Pourquoi “minimax” ?



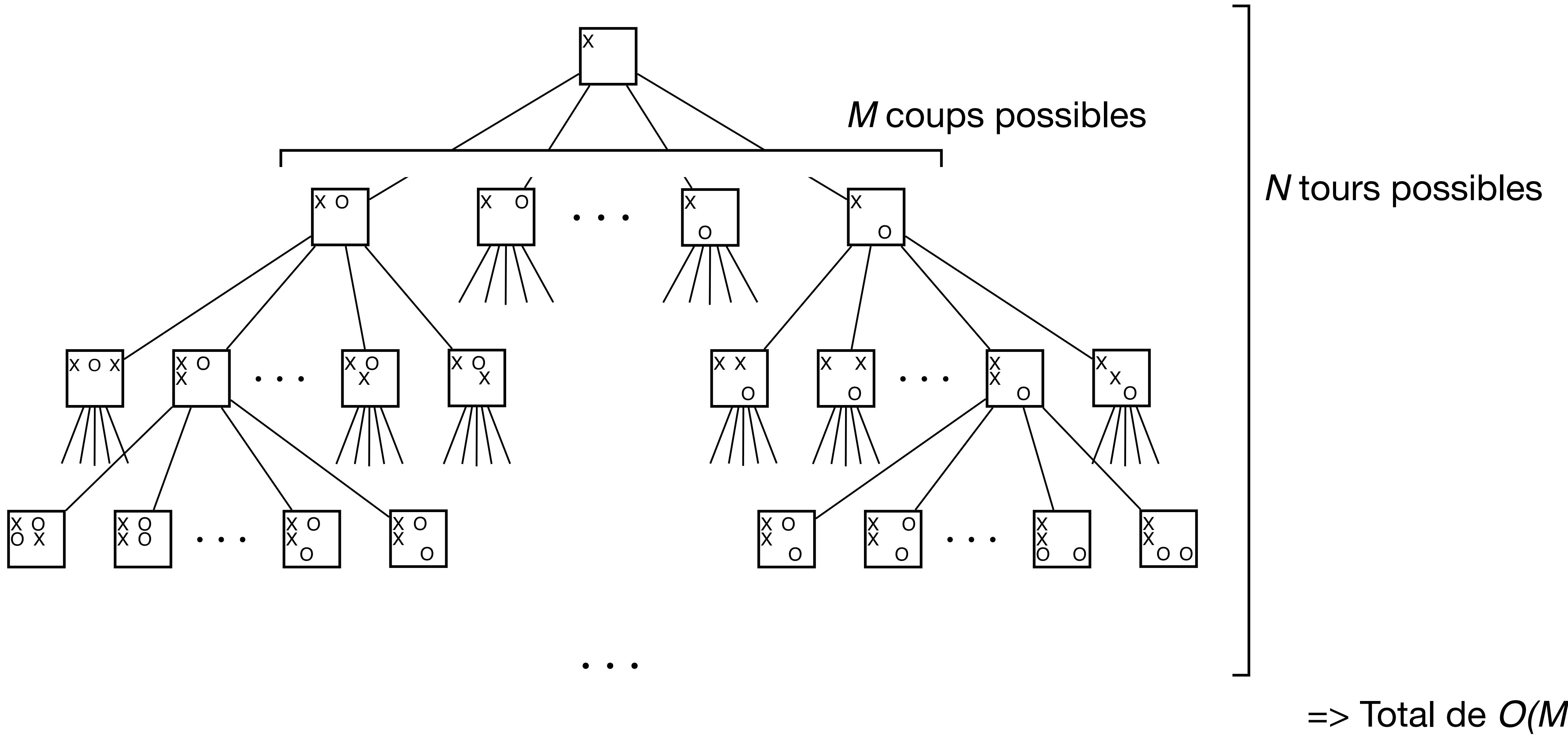


# Pourquoi “minimax” ?

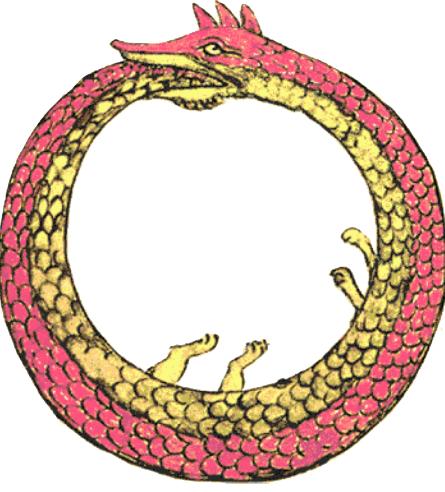




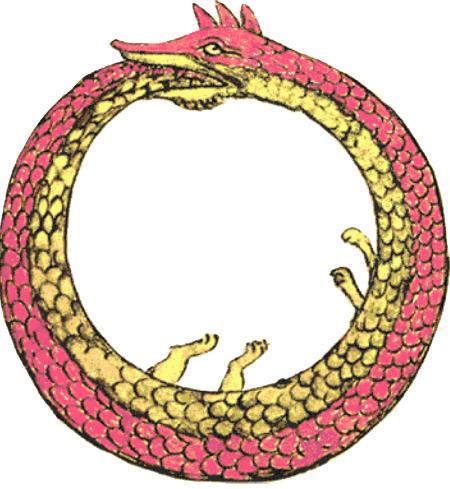
# Complexité ?



# Complexité de ce type d'algorithme ?



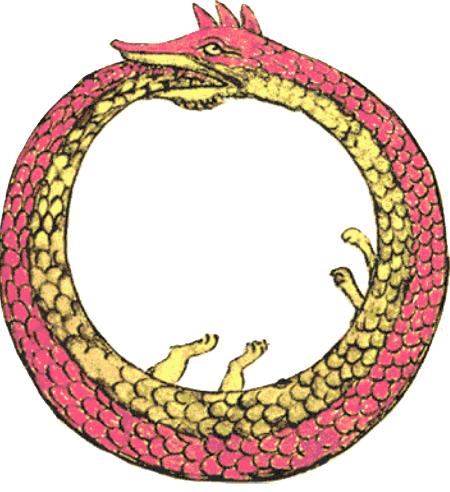
- Le joueur peut choisir entre  $M$  coups, chacun requérant un appel récursif.
- On explore  $N$  coups successifs
- il y aura  $O(M^N)$  appels récursifs.
- $M$  et  $N$  dépendent
  - des règles du jeu
  - de la position dans le jeu.



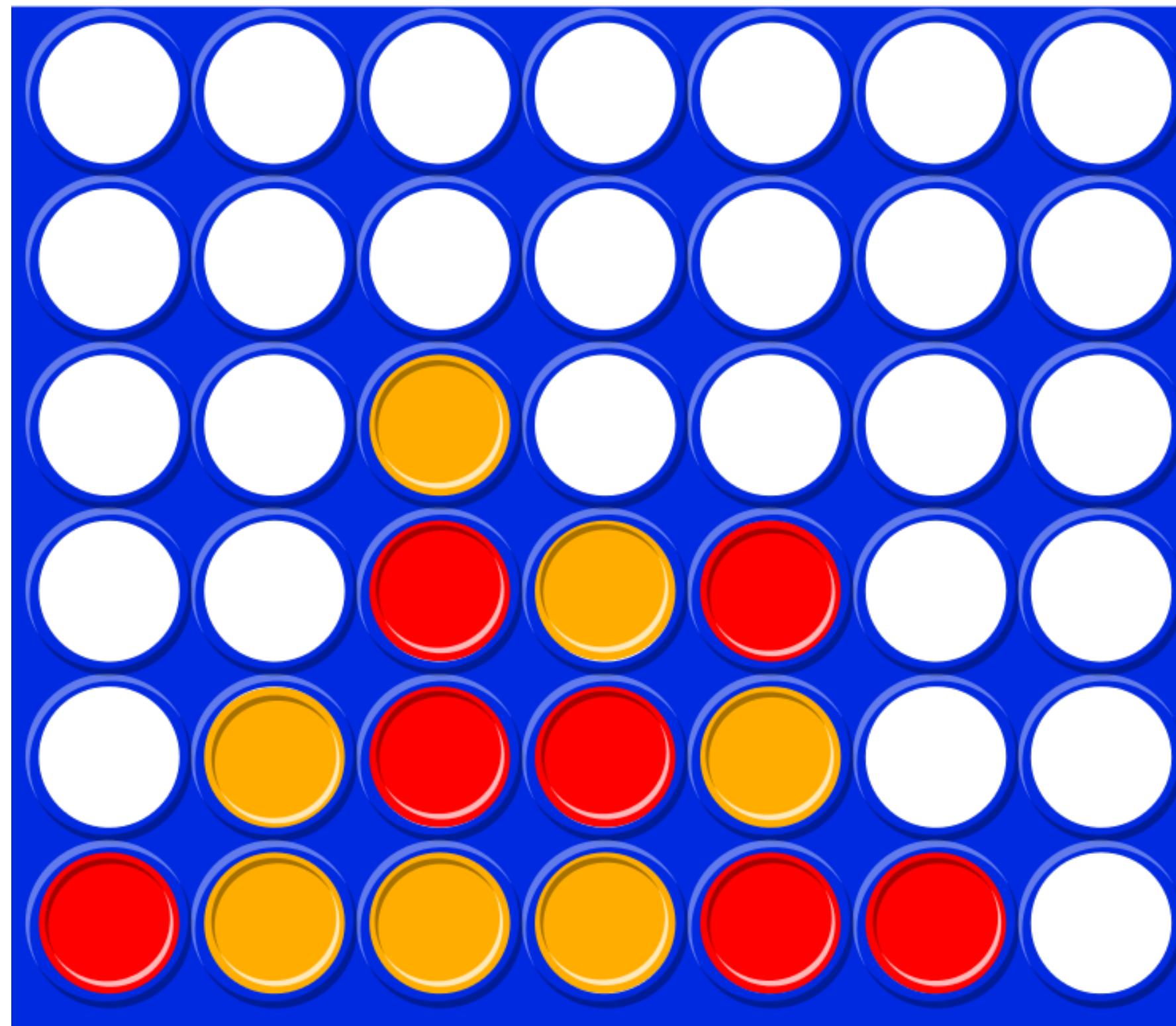
# Tic Tac Toe



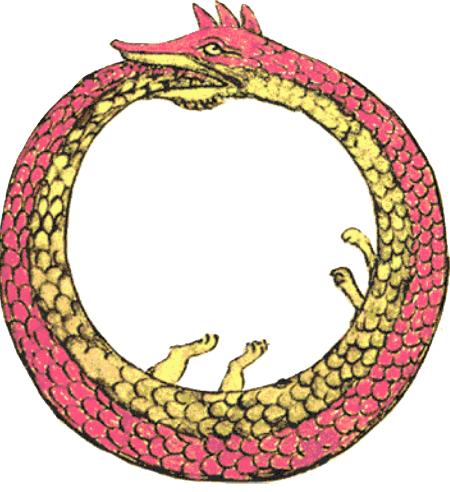
- $9! = 362880$  manières différentes de remplir la grille.
- Seulement 255168 parties possibles.
  - 131184 victoires de X
  - 77904 victoires de O
  - 46080 égalités
- Toujours égalité si les deux joueurs jouent parfaitement



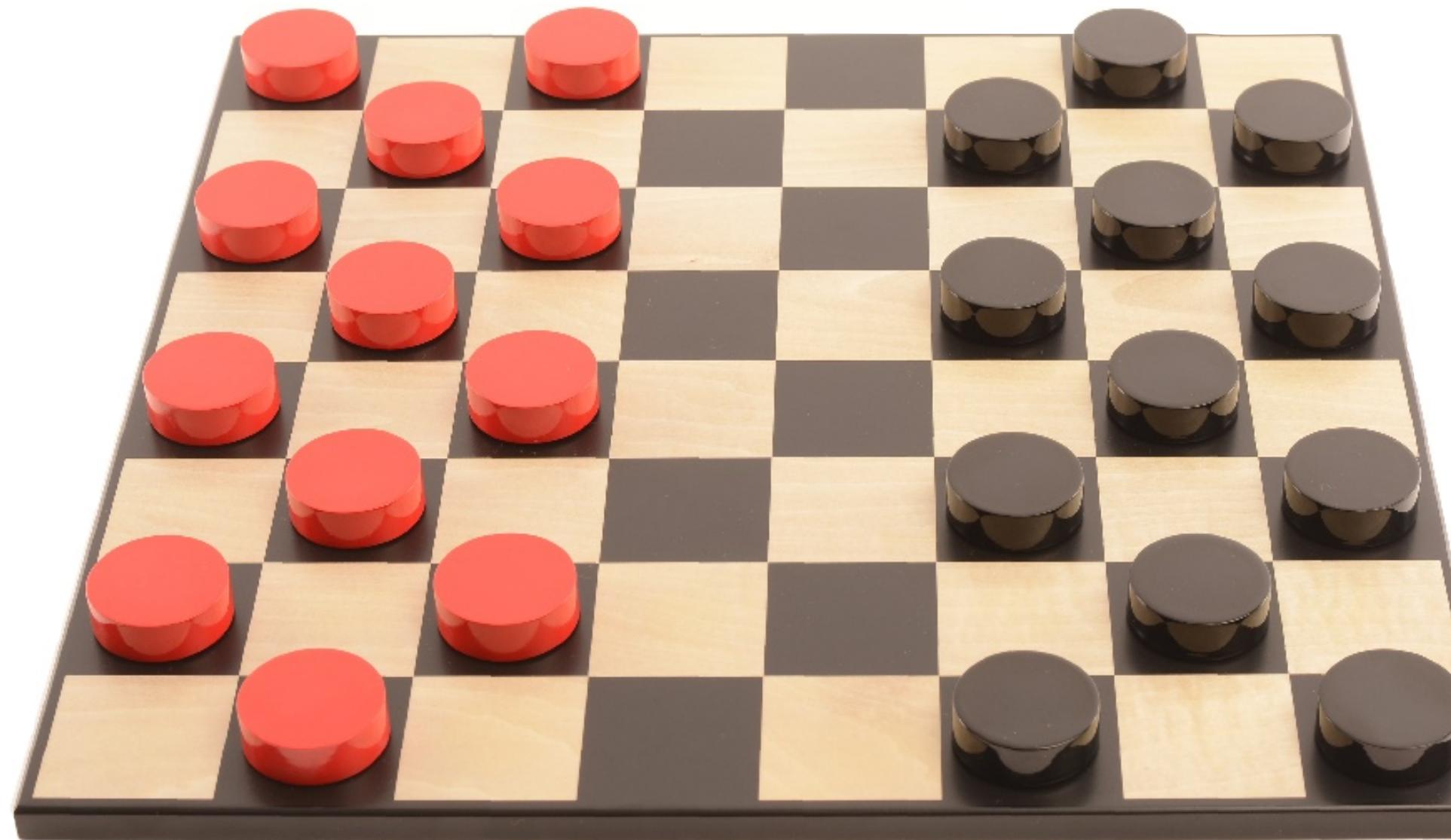
# Puissance 4



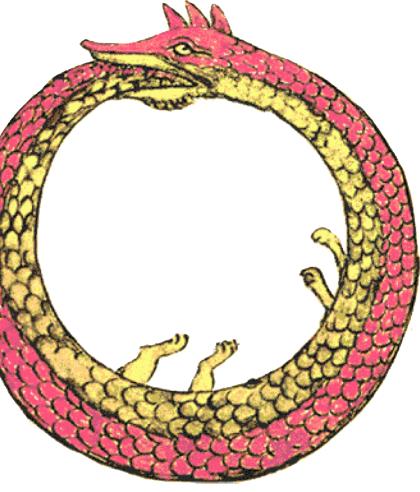
- Au début, 7 colonnes, donc  $O(7^n)$  pour n coups.
- Résolu par James D. Allen en 1988.
- 4'531'985'219'092 parties possibles.
- Joueur 1 gagne au 41ème coup s'il joue parfaitement en commençant dans la colonne centrale.
- Egalité s'il entame dans une colonne adjacente au centre.
- Joueur 2 gagne sinon



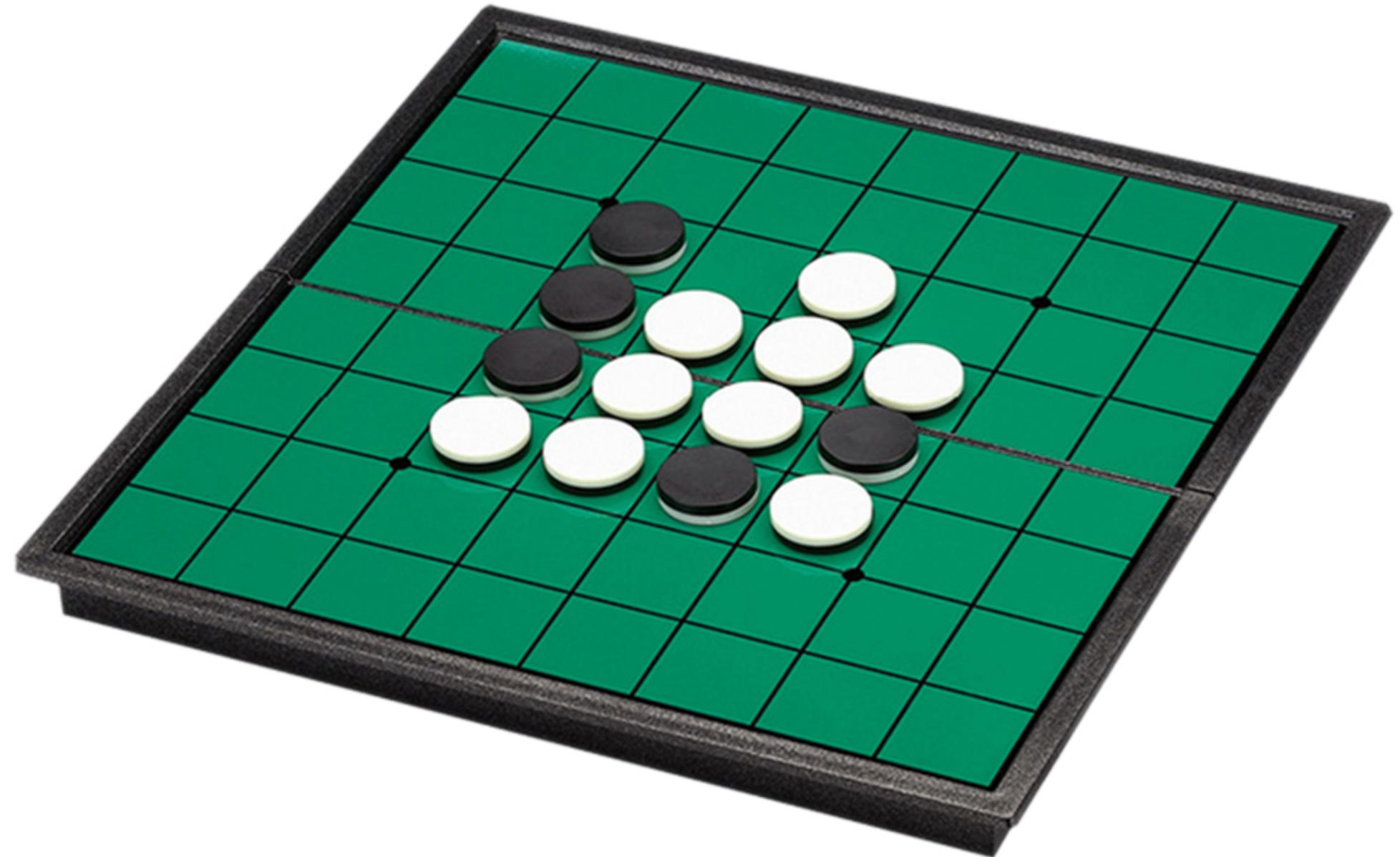
# Jeu de dames



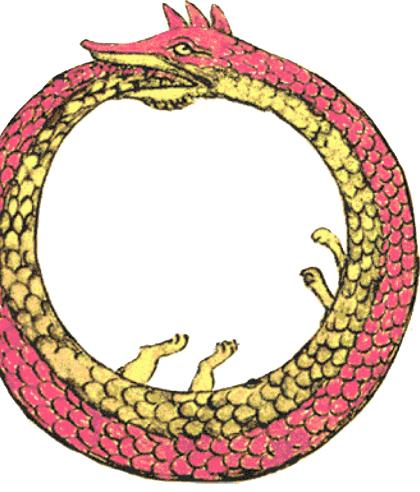
- $5 \times 10^{20}$  positions possibles sur grille 8x8
- Le programme Chinook gagne le championnat du monde en 1994 face aux humains.
- Résolu par Schaeffer et al. en 2007. Environ 18 ans de calcul non-stop
- Egalité si les deux joueurs jouent parfaitement



# Reversi



- Pas de solution complète ni de stratégie parfaite
- On dispose d'heuristiques suffisamment bonnes pour jouer aussi bien ou mieux que des humains
- En 1997, le logiciel Logistello gagne 6 victoires à 0 face au champion du monde humain Takeshi Murakami



# Jeu d'échecs



- En 1997, Deep Blue - qui tourne sur un super ordinateur avec du hardware dédié aux échecs - bat Gary Kasparov, champion du monde humain
- En 2003, Deep Junior et X3D Fritz - qui tournent sur des PC - font tous deux match nul face à Kasparov

# Jeu de Go



- Longtemps considéré comme hors d'atteinte des ordinateurs de par le nombre de combinaisons possibles:  $10^{170}$ .
- En 2016 le logiciel AlphaGo bat le champion du monde Lee Sedol



Humankind  
vs  
Computer

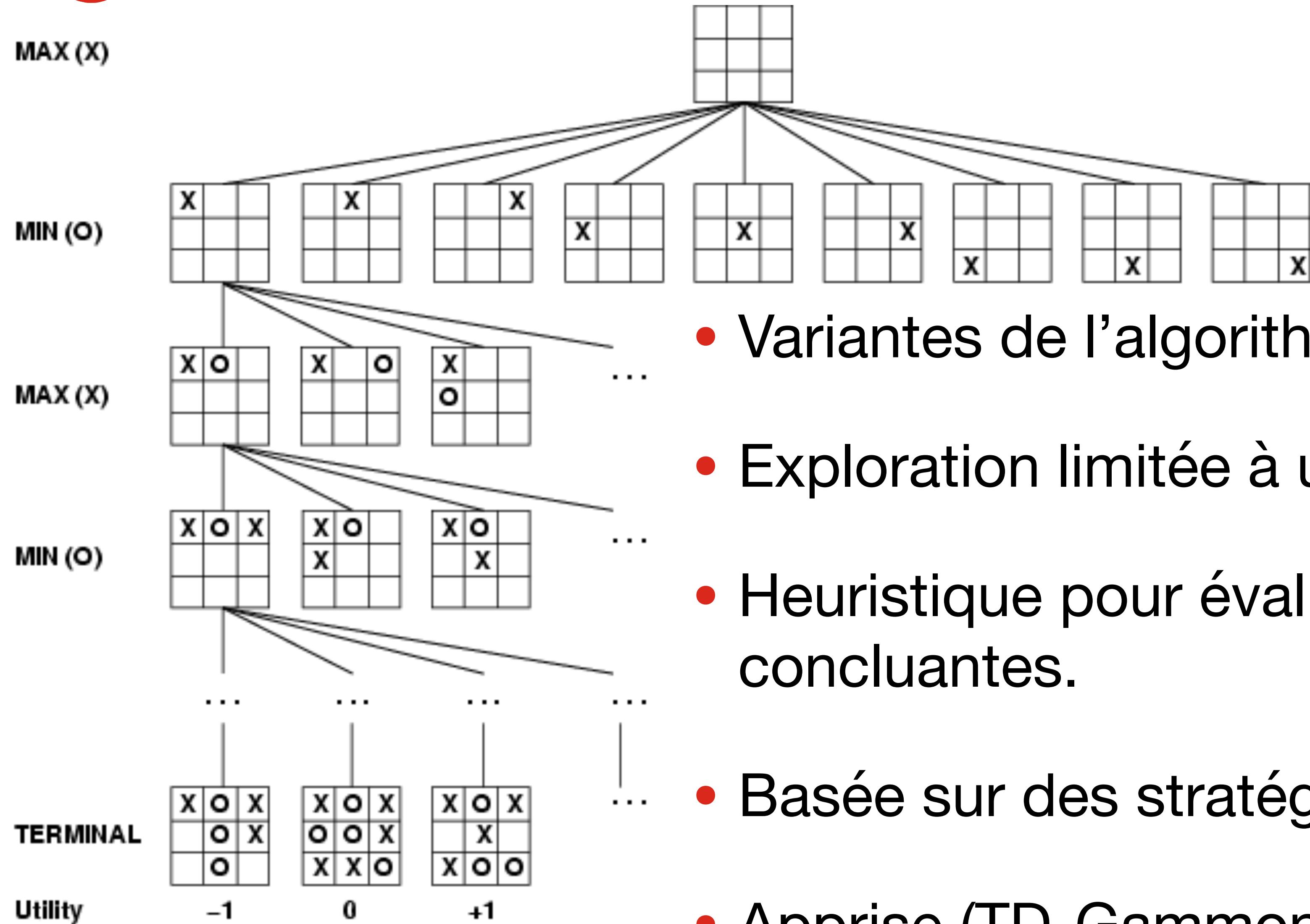
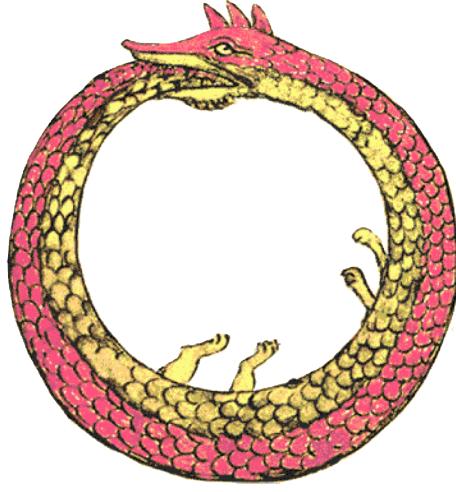


 AlphaGo vs Lee Sedol

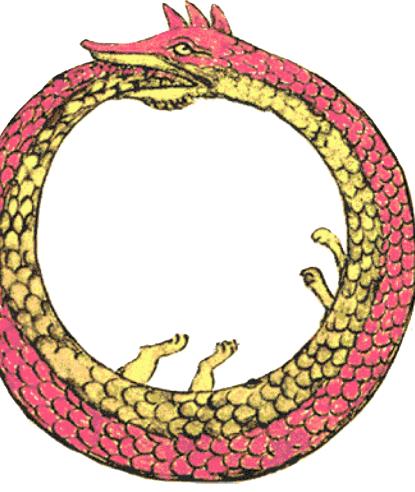
人類と人工知能の叡智をかけた  
史上最強の五番勝負が始まる。  
勝つのは、李世乭か？ Googleか？  
賞金100万ドル

対局日程：3月9、10、12、13、15日／対局場：韓国ソウル

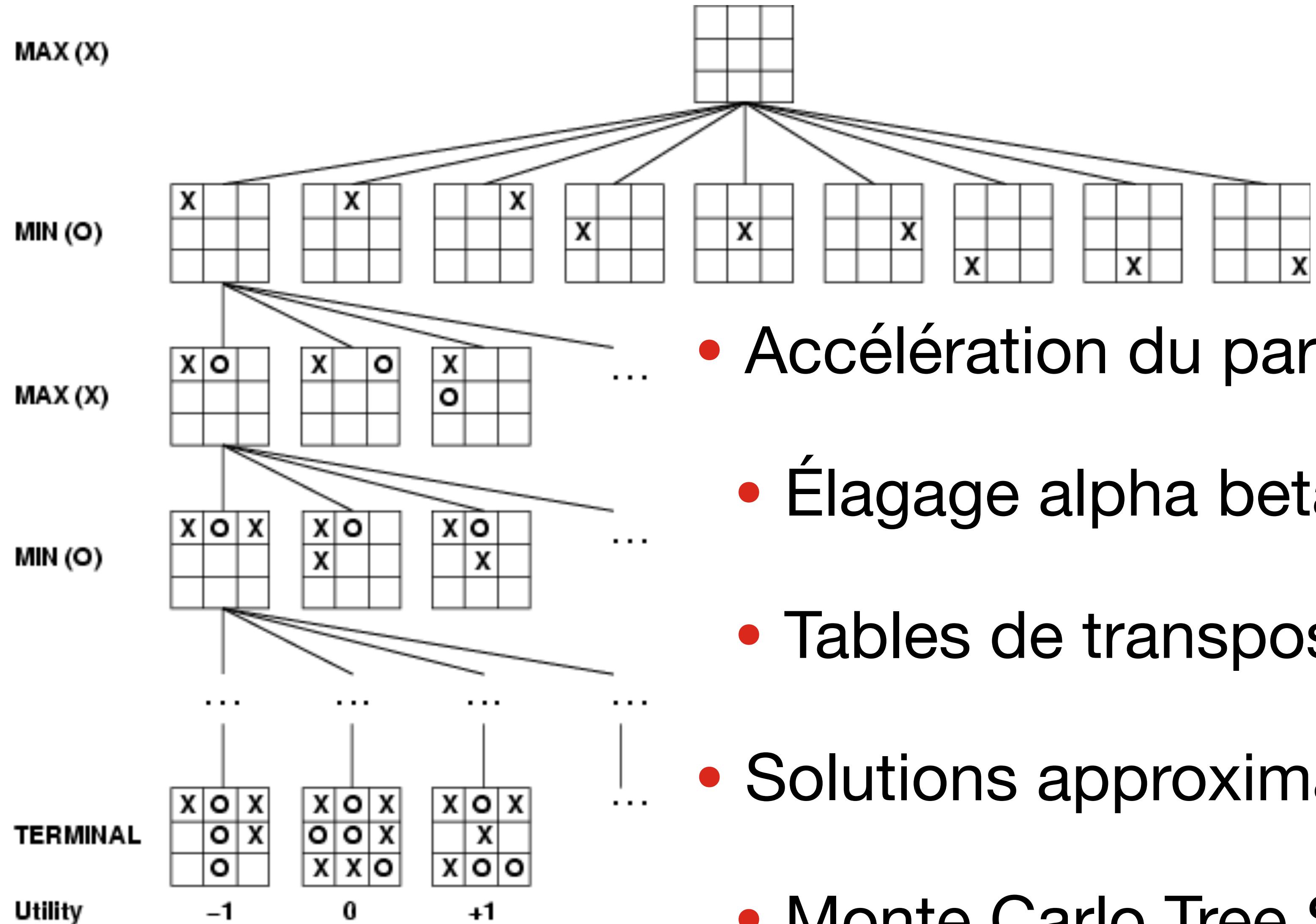
# Comment jouent-ils si bien ?



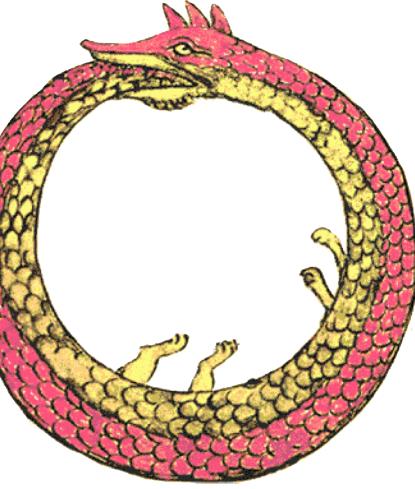
- Variantes de l'algorithme MiniMax
- Exploration limitée à une profondeur maximale
- Heuristique pour évaluer les positions non concluantes.
- Basée sur des stratégies connues
- Apprise (TD-Gammon)



# Comment jouent-ils si bien ? (2)

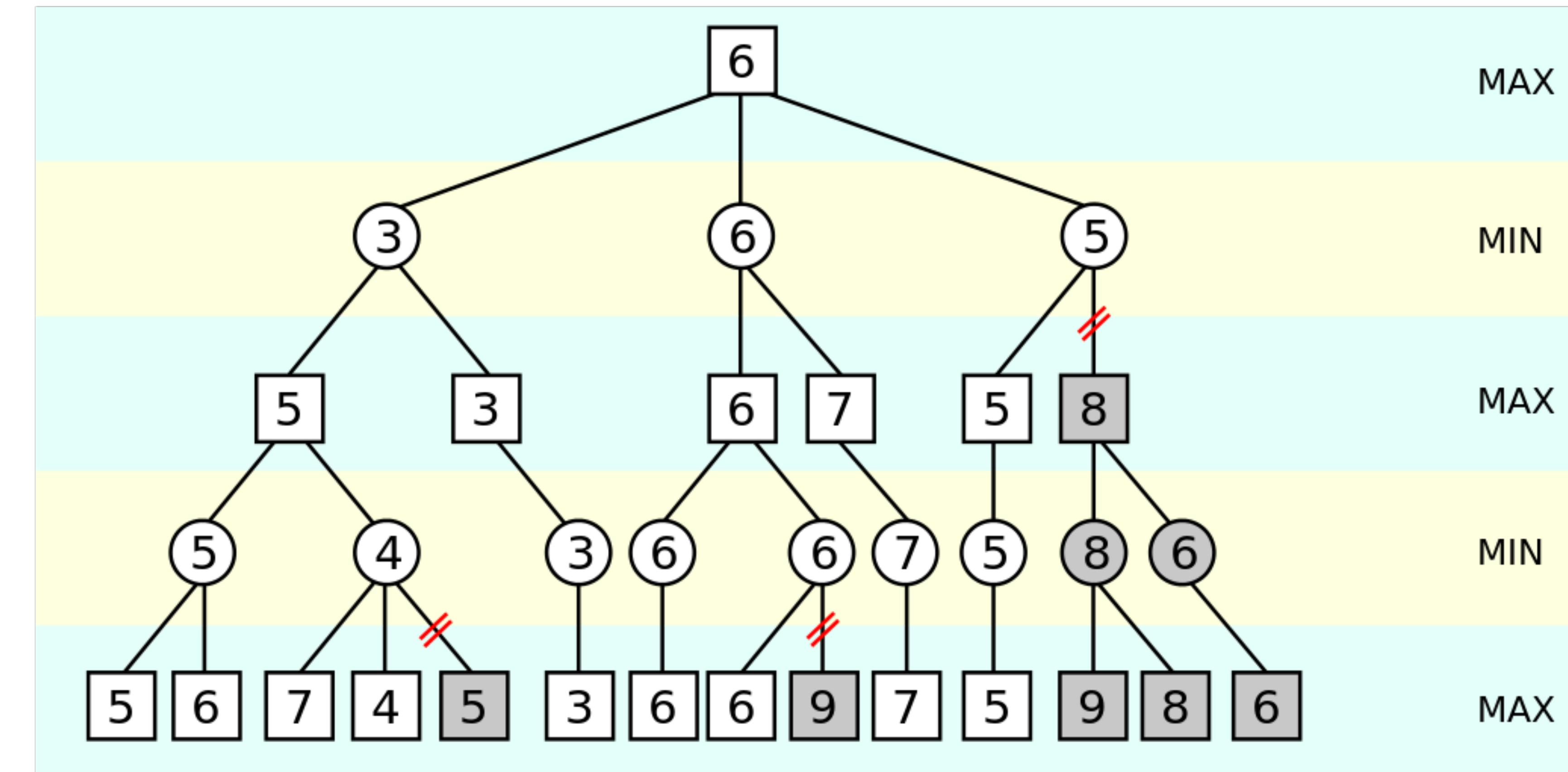


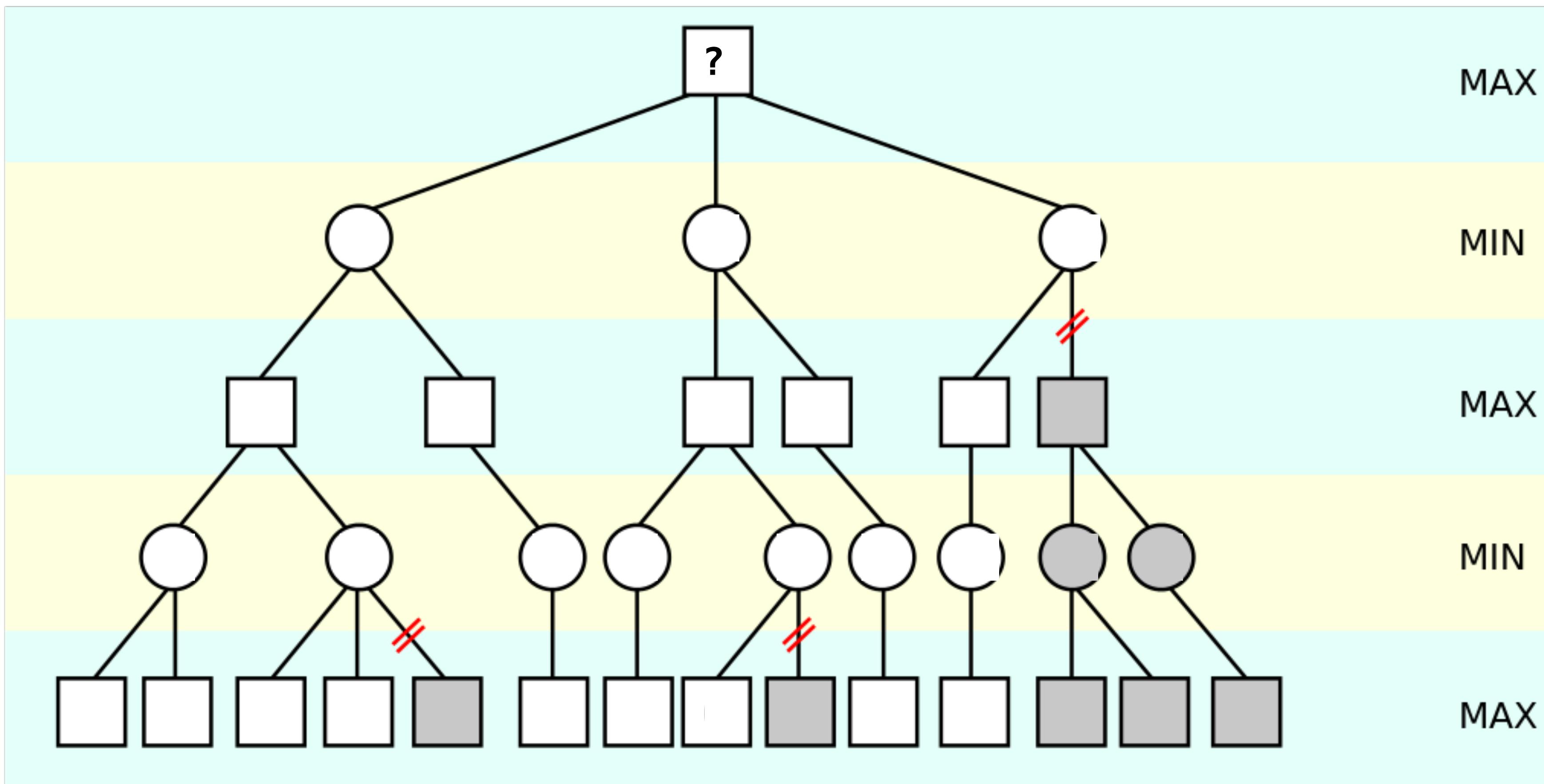
- Accélération du parcours exact de l'arbre
- Élagage alpha beta
- Tables de transposition
- Solutions approximatives
- Monte Carlo Tree Search (AlphaGo)

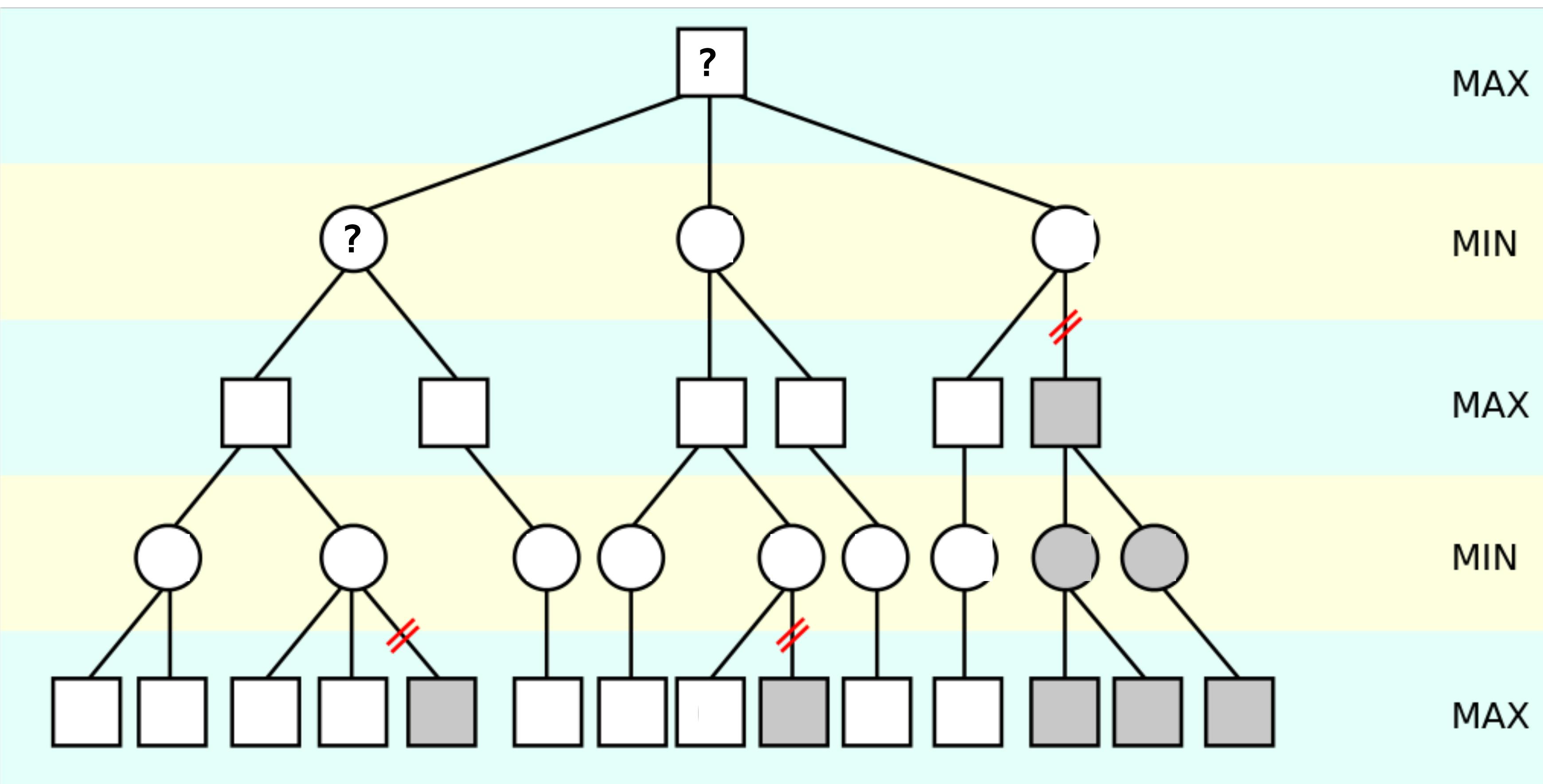


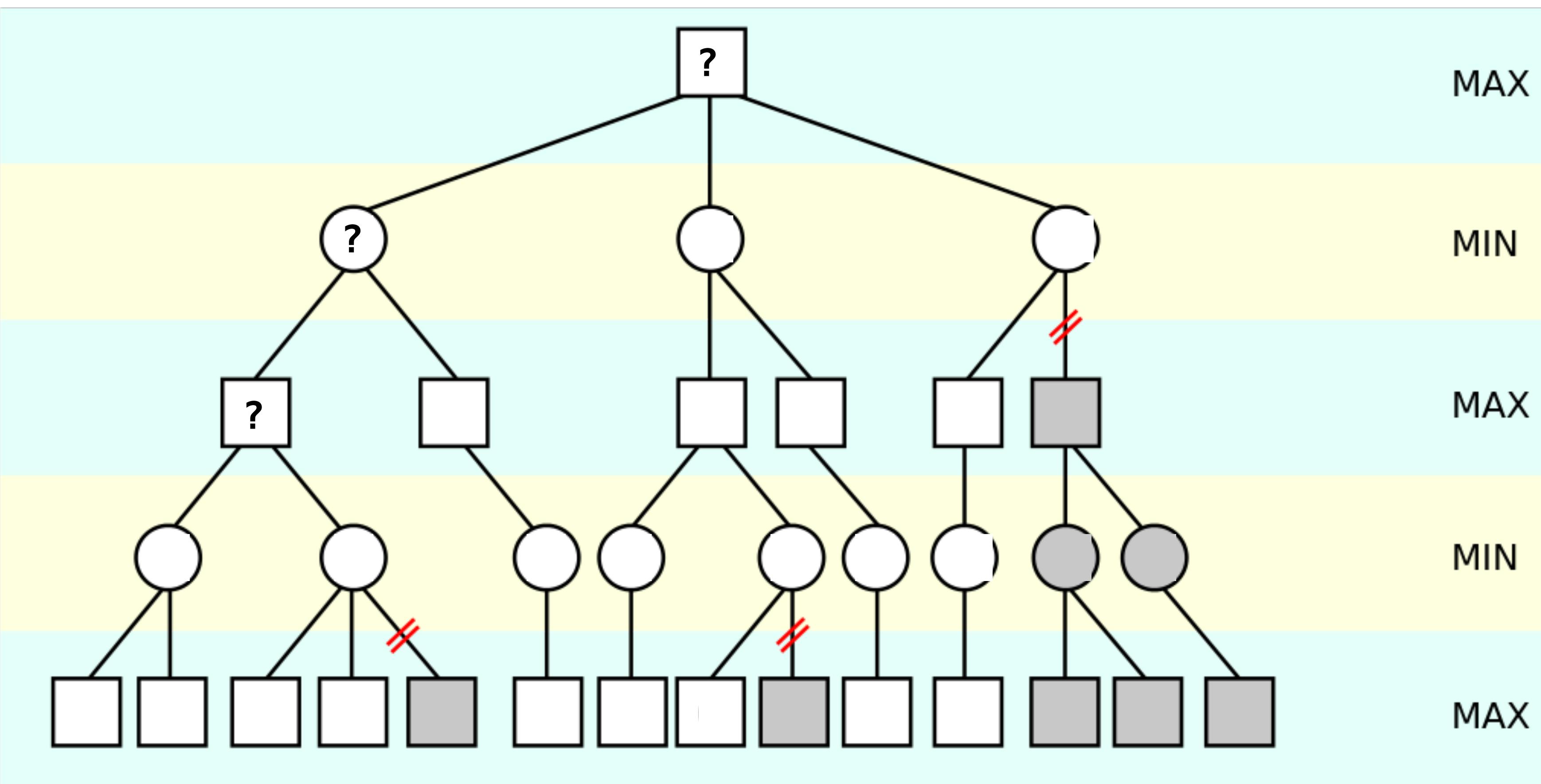
# Elagage alpha-beta

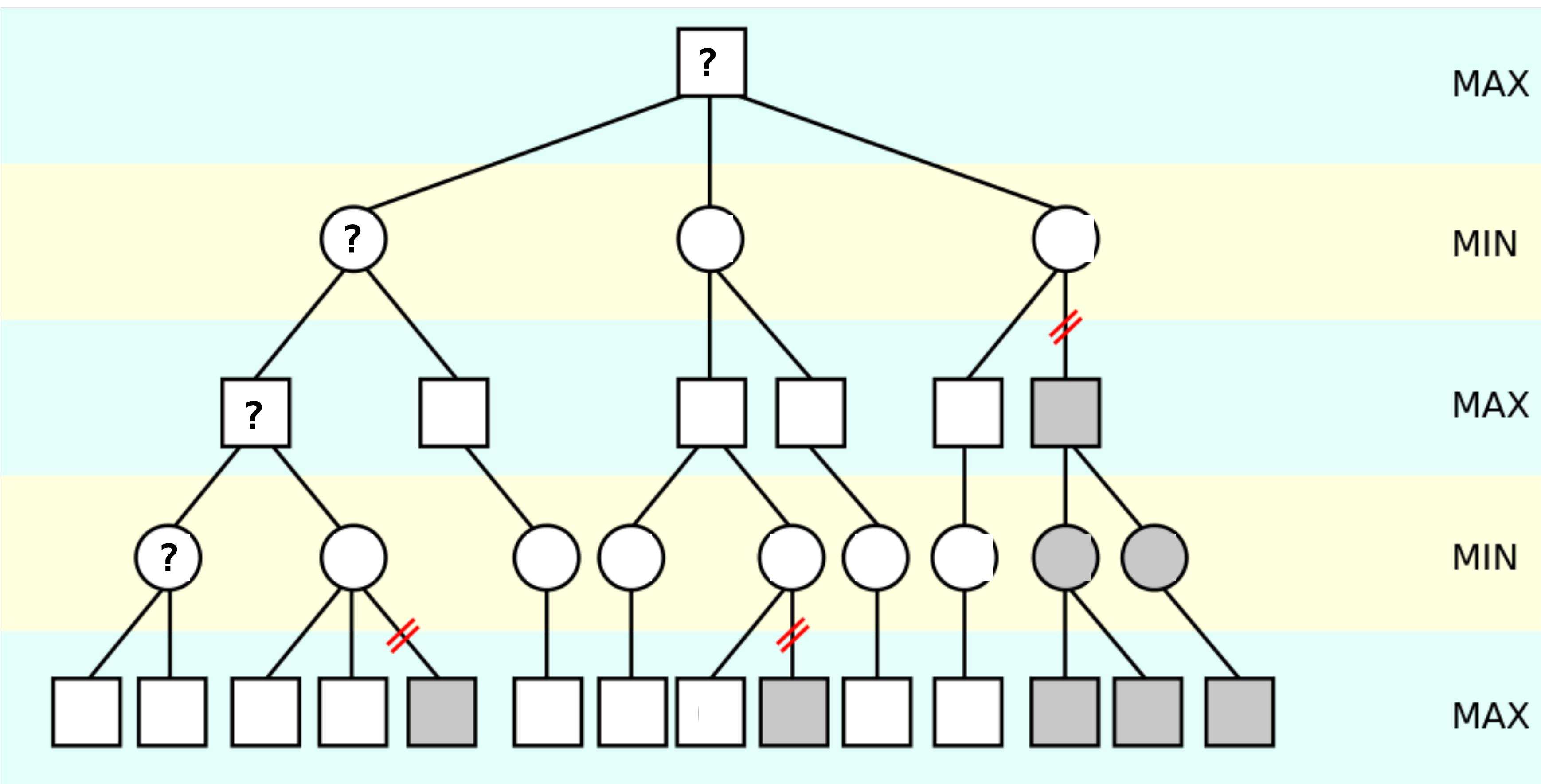
- Arrêter la recherche de minimum **beta** si il est déjà plus petit que **alpha**, le max au niveau au dessus
- Arrêter la recherche de maximum **alpha** si il est déjà plus grand que **beta**, le min au niveau au dessus

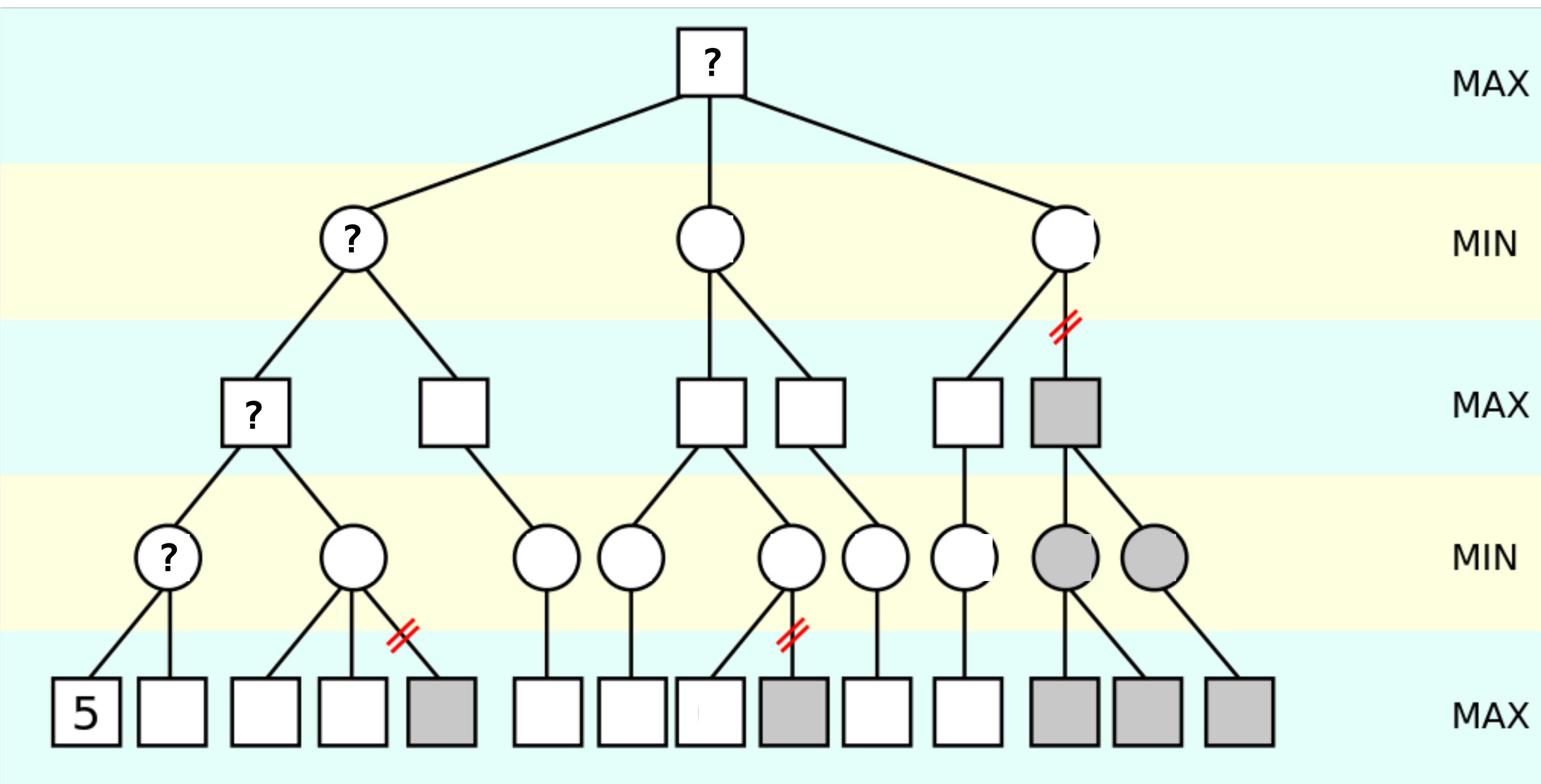


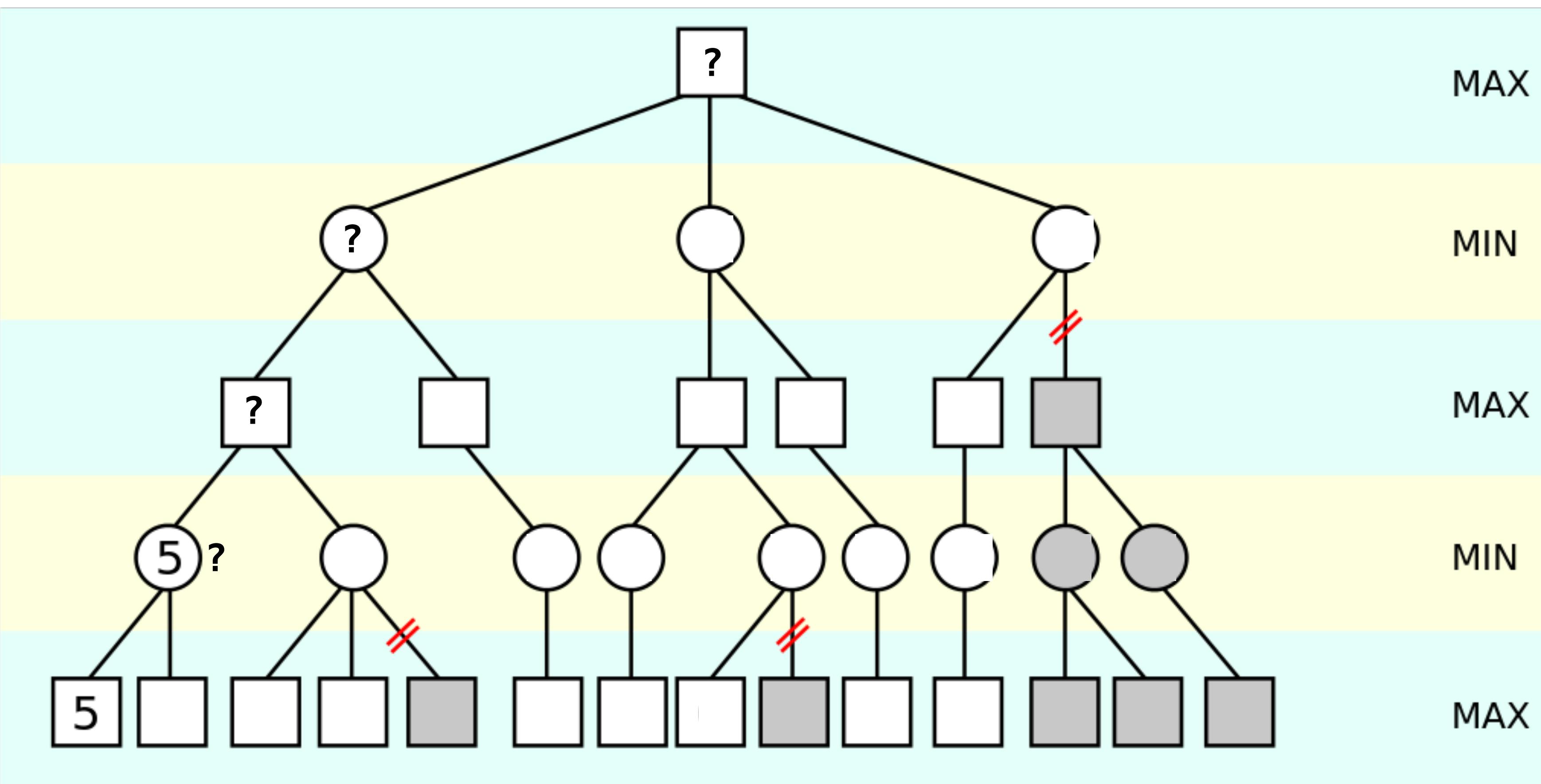


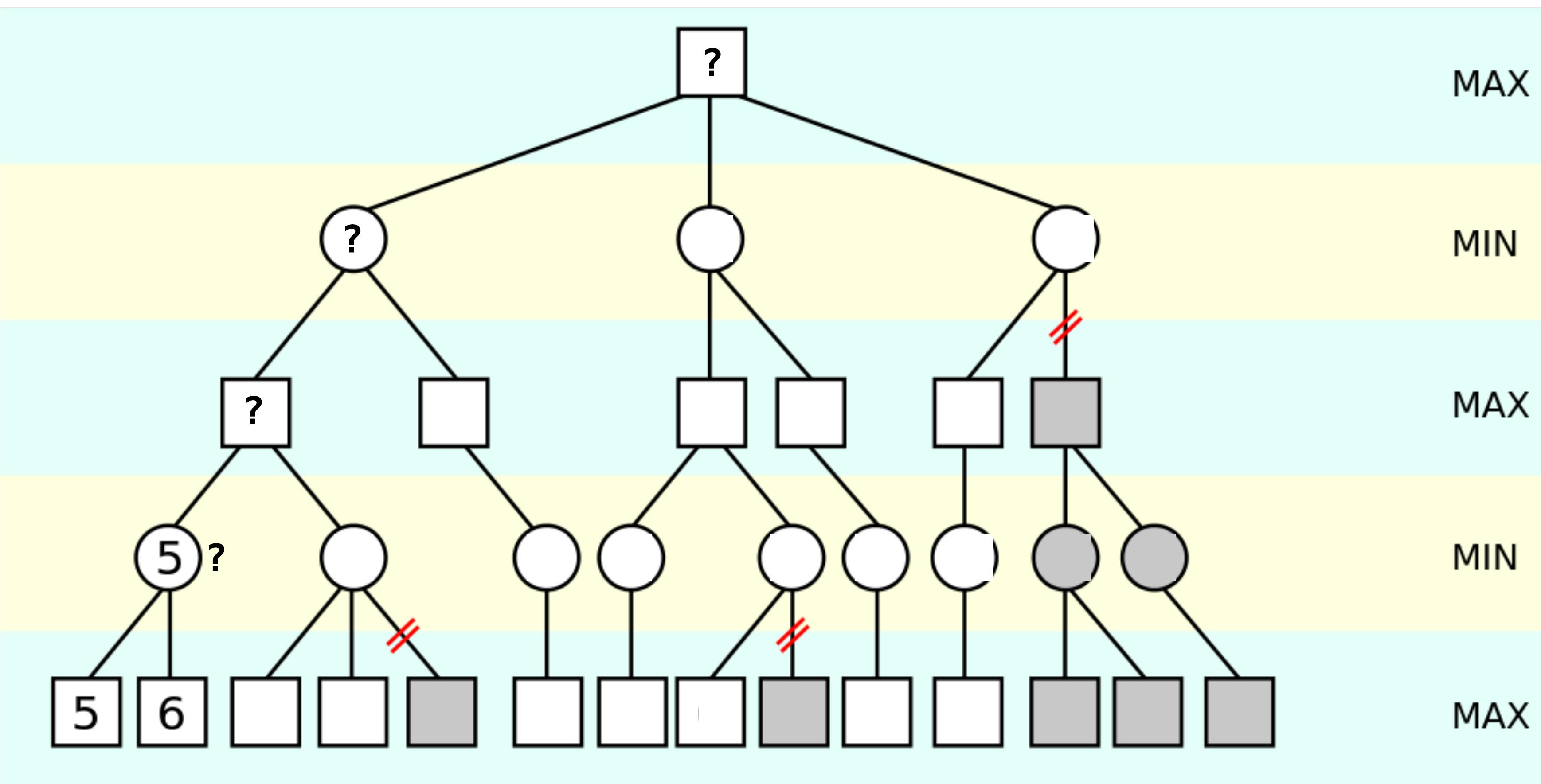


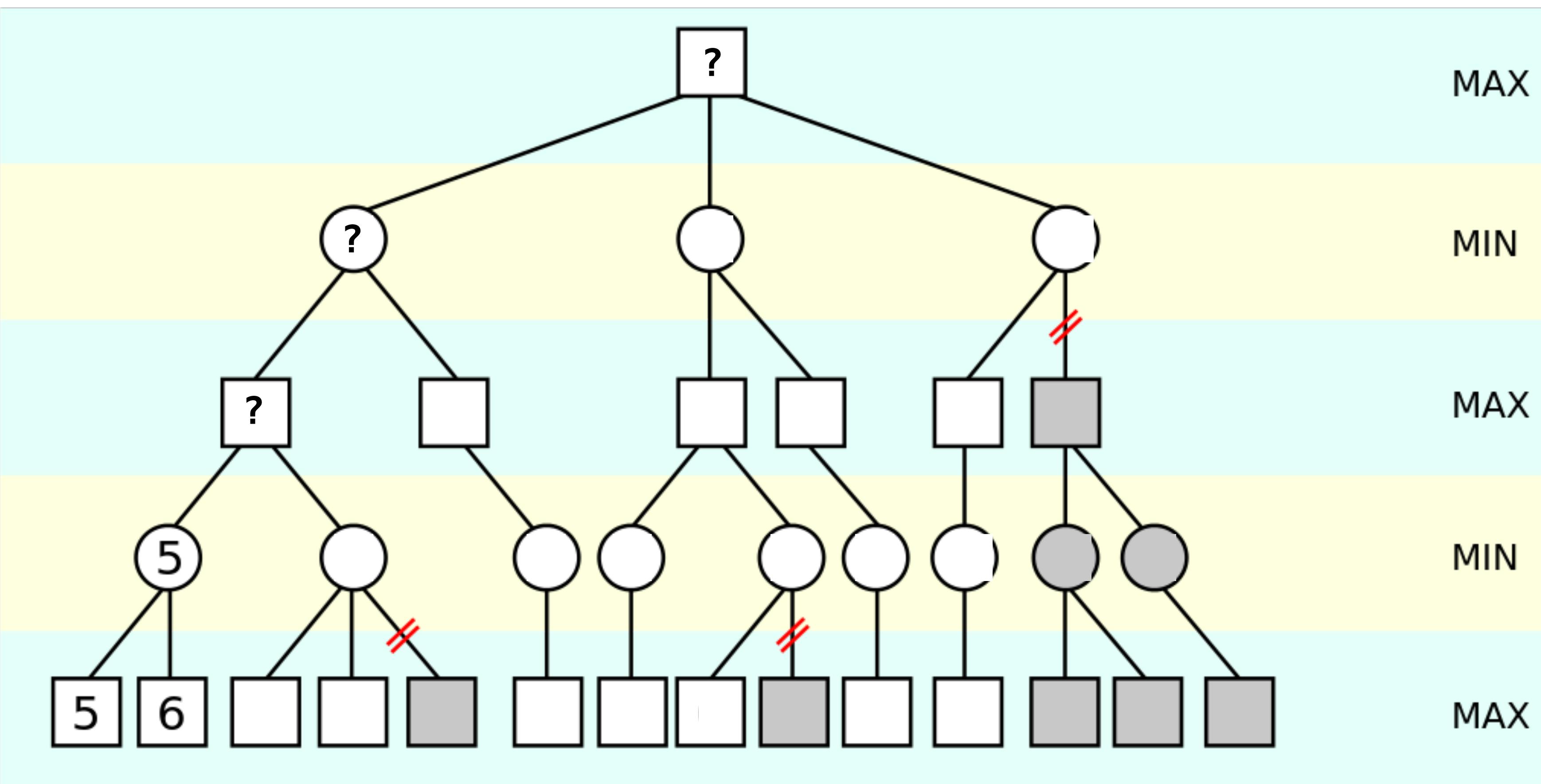


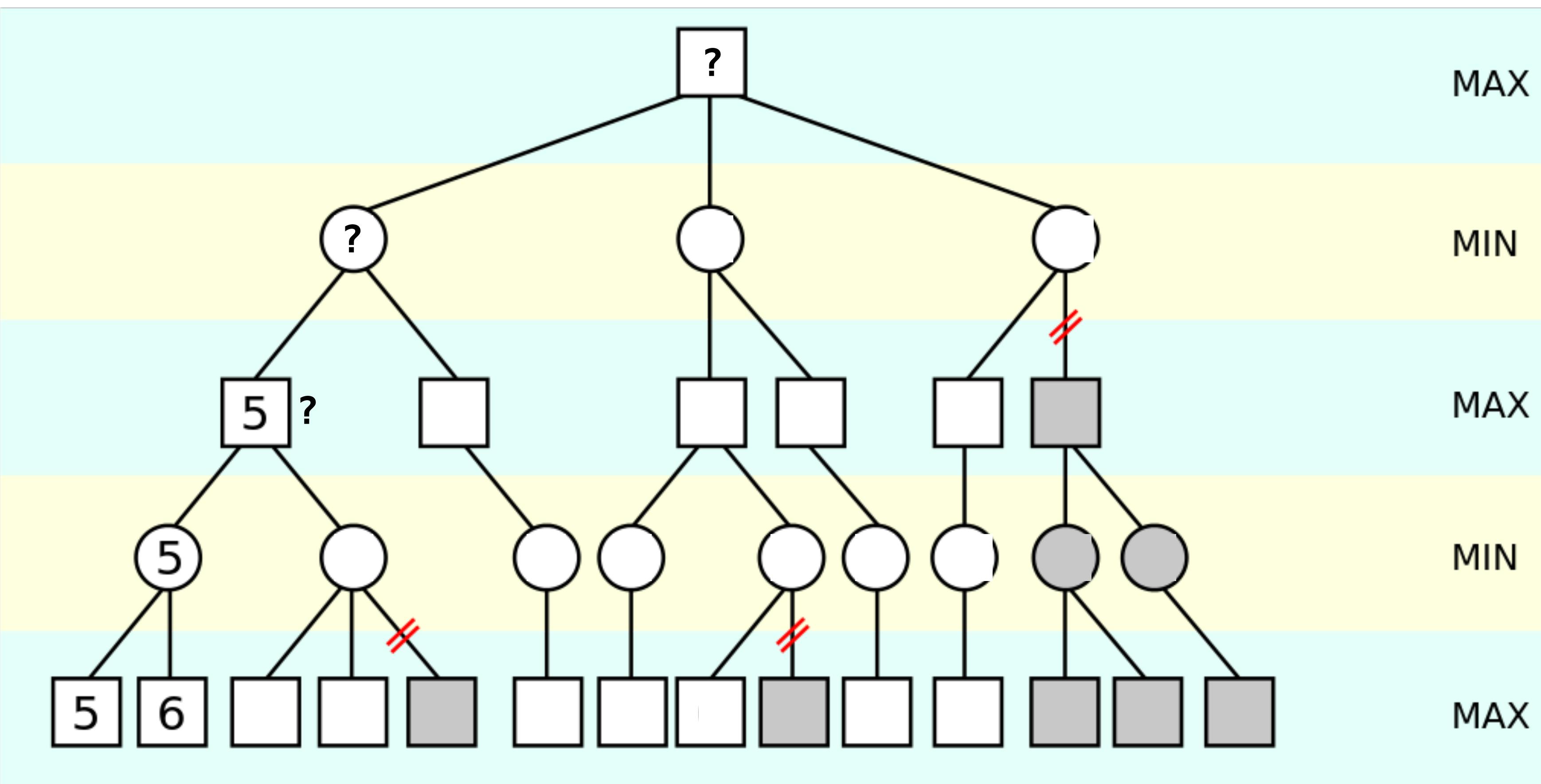


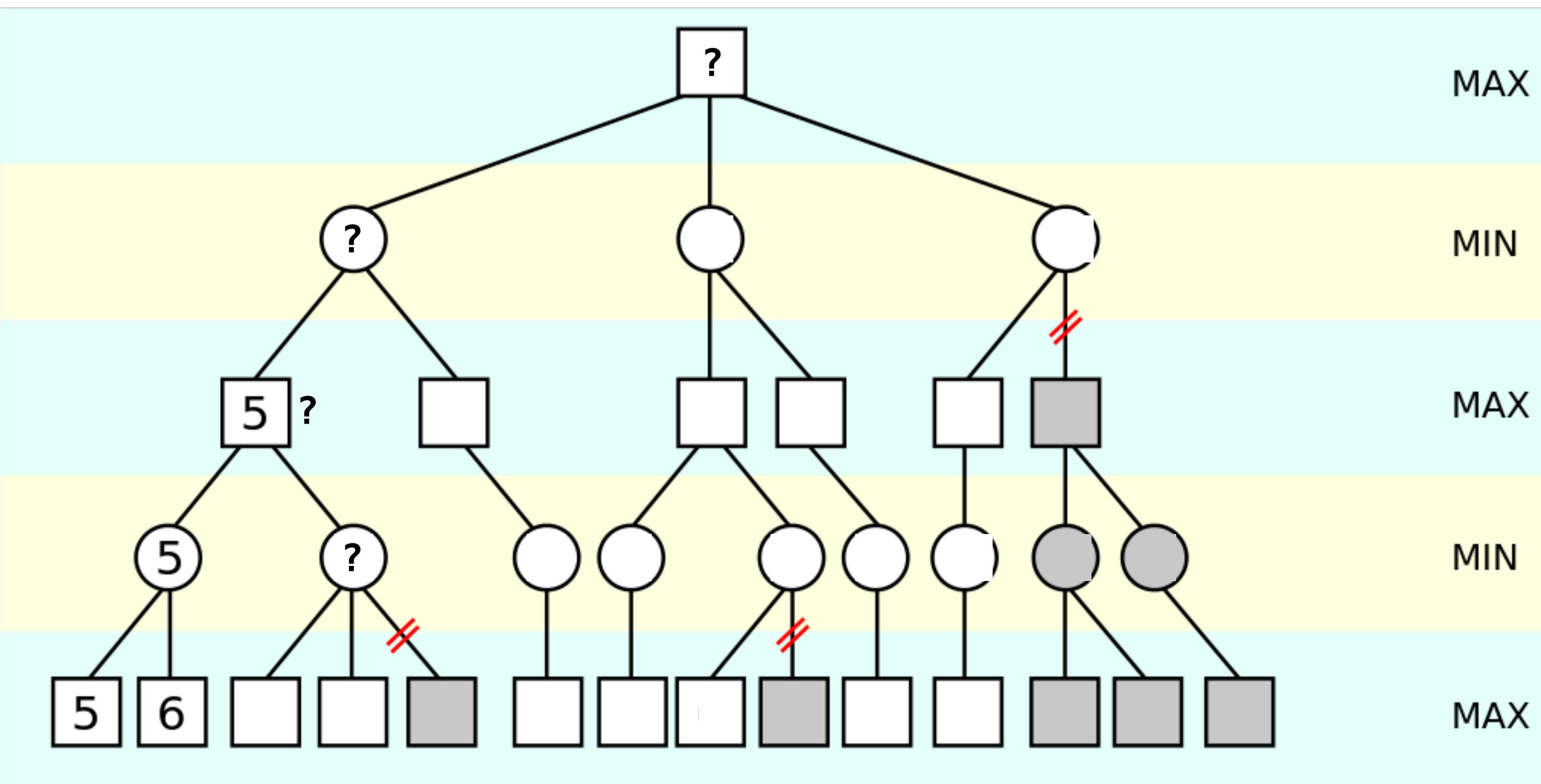


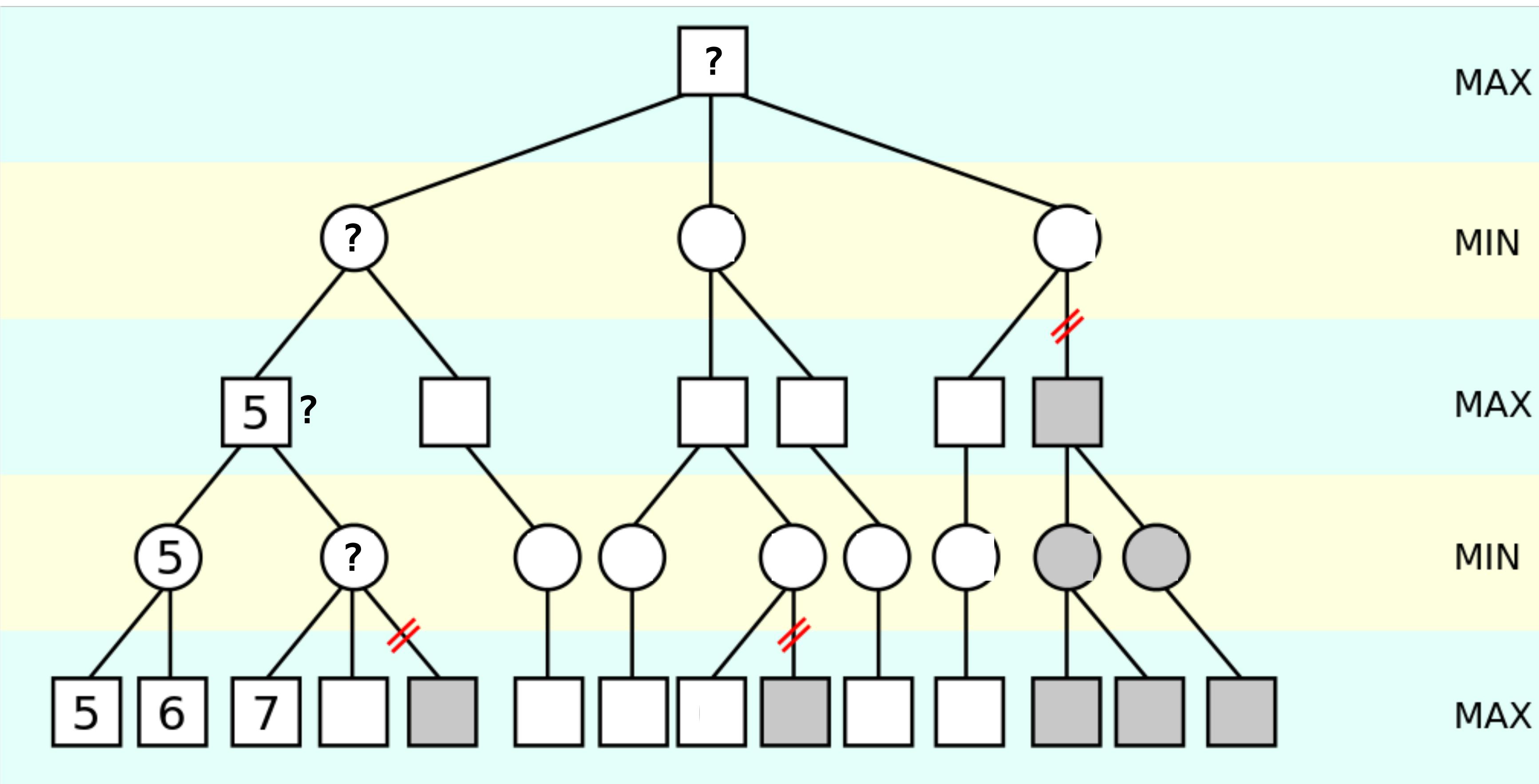


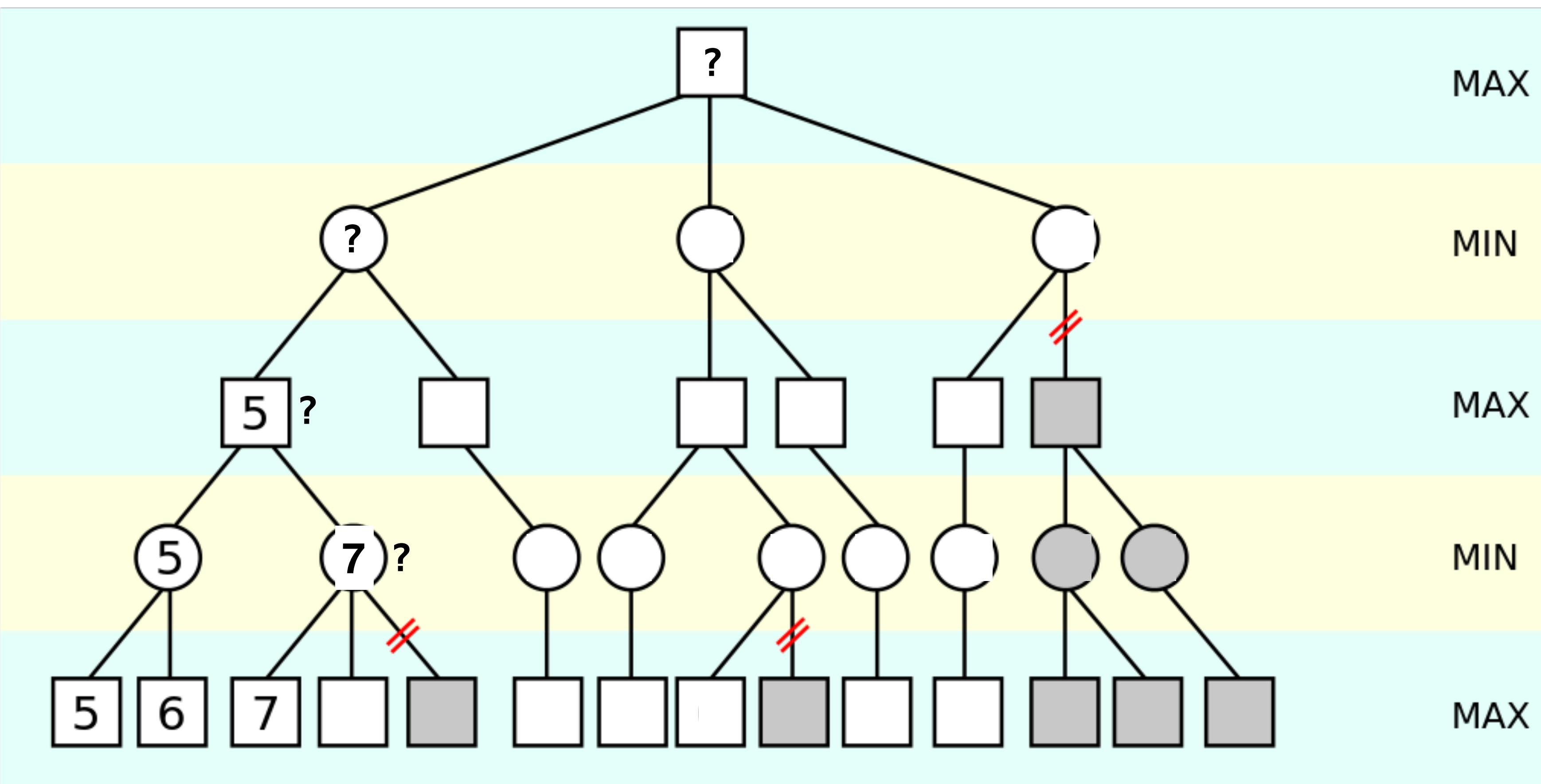


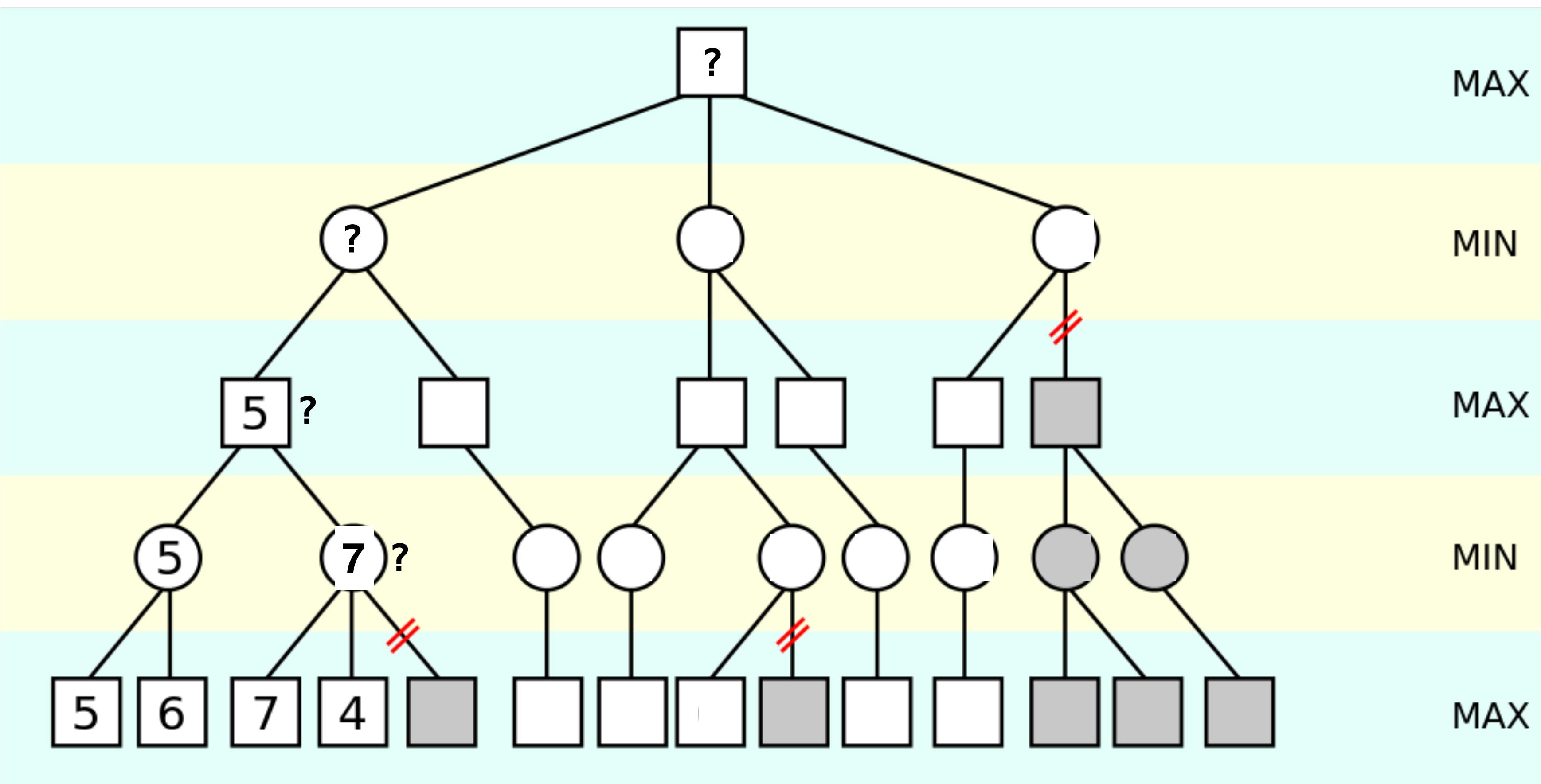


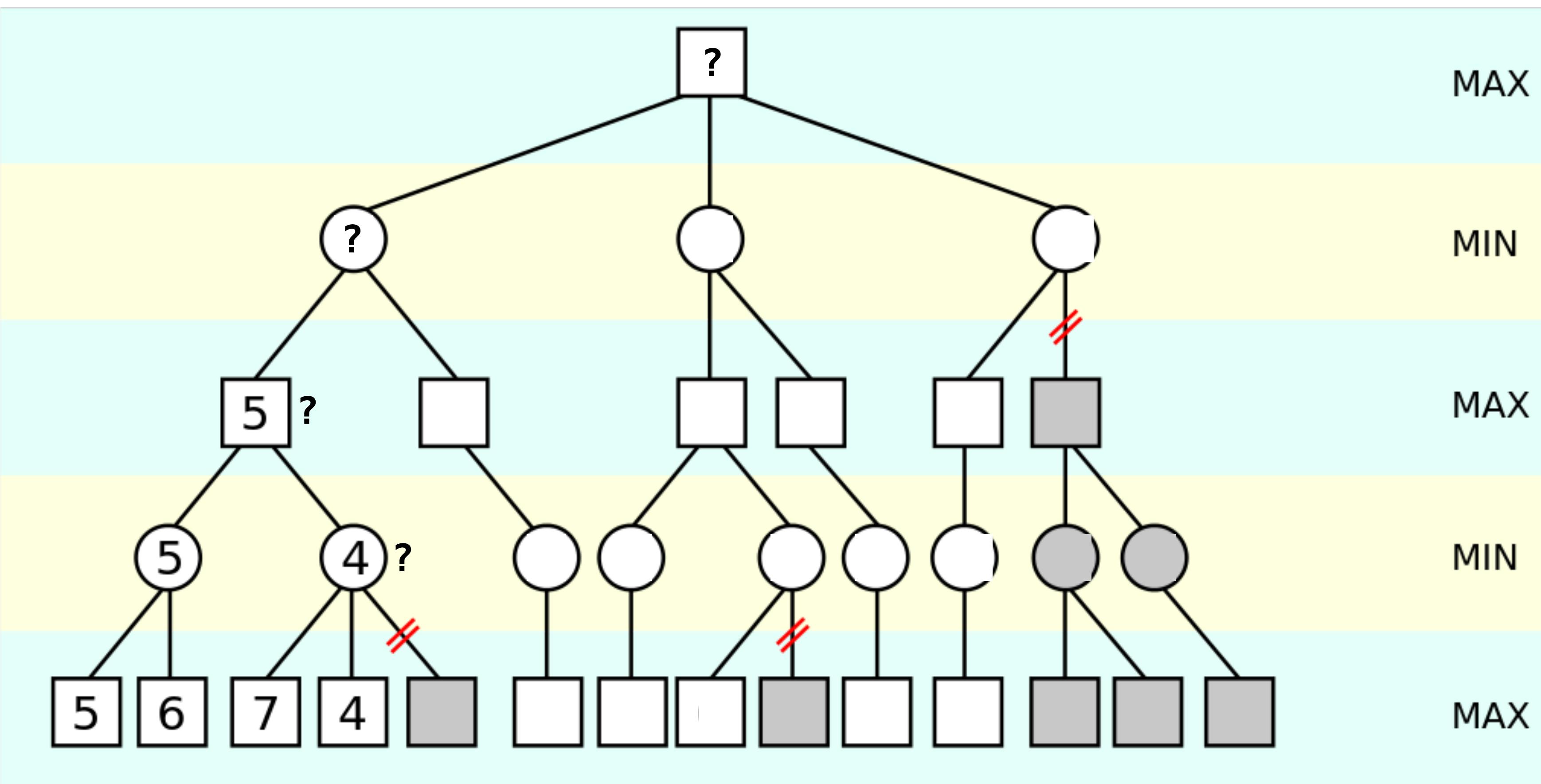




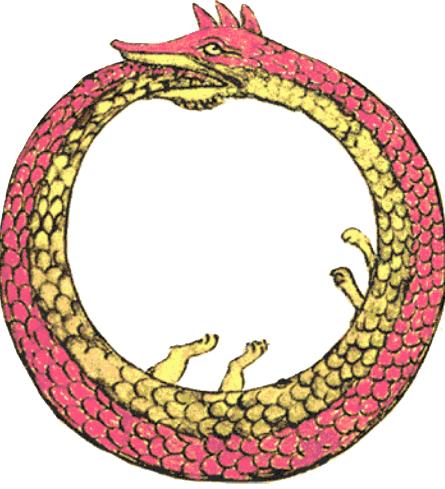




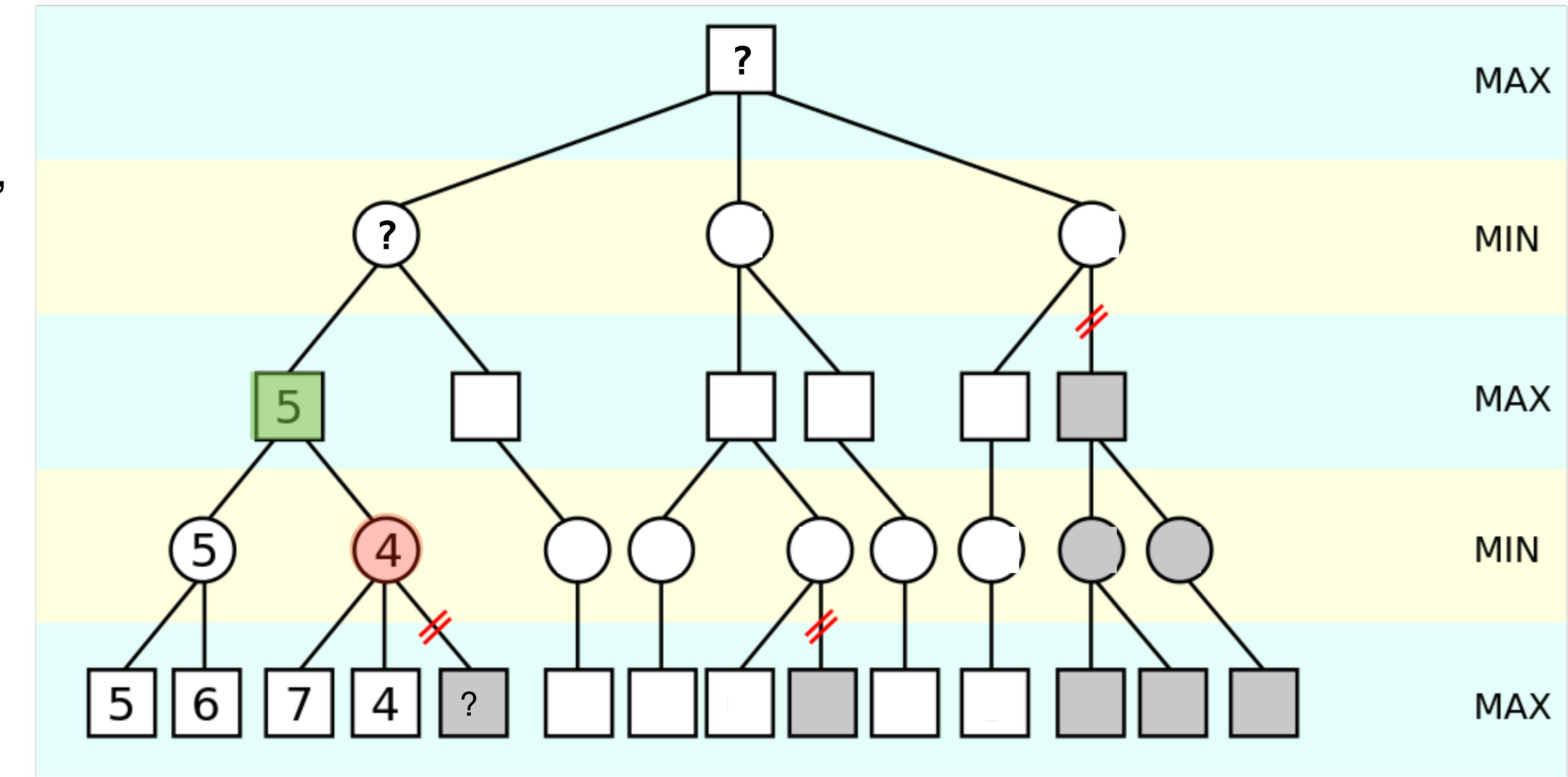




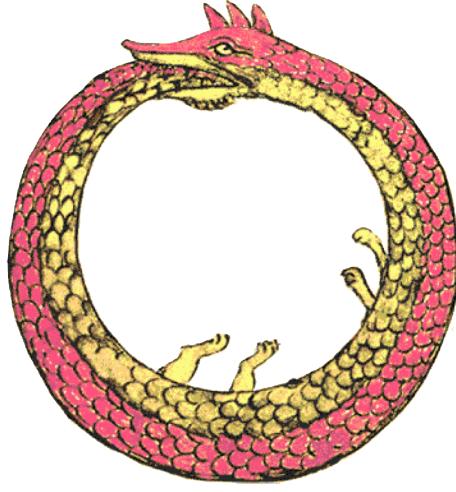
## Arrêtons-nous au calcul de 4



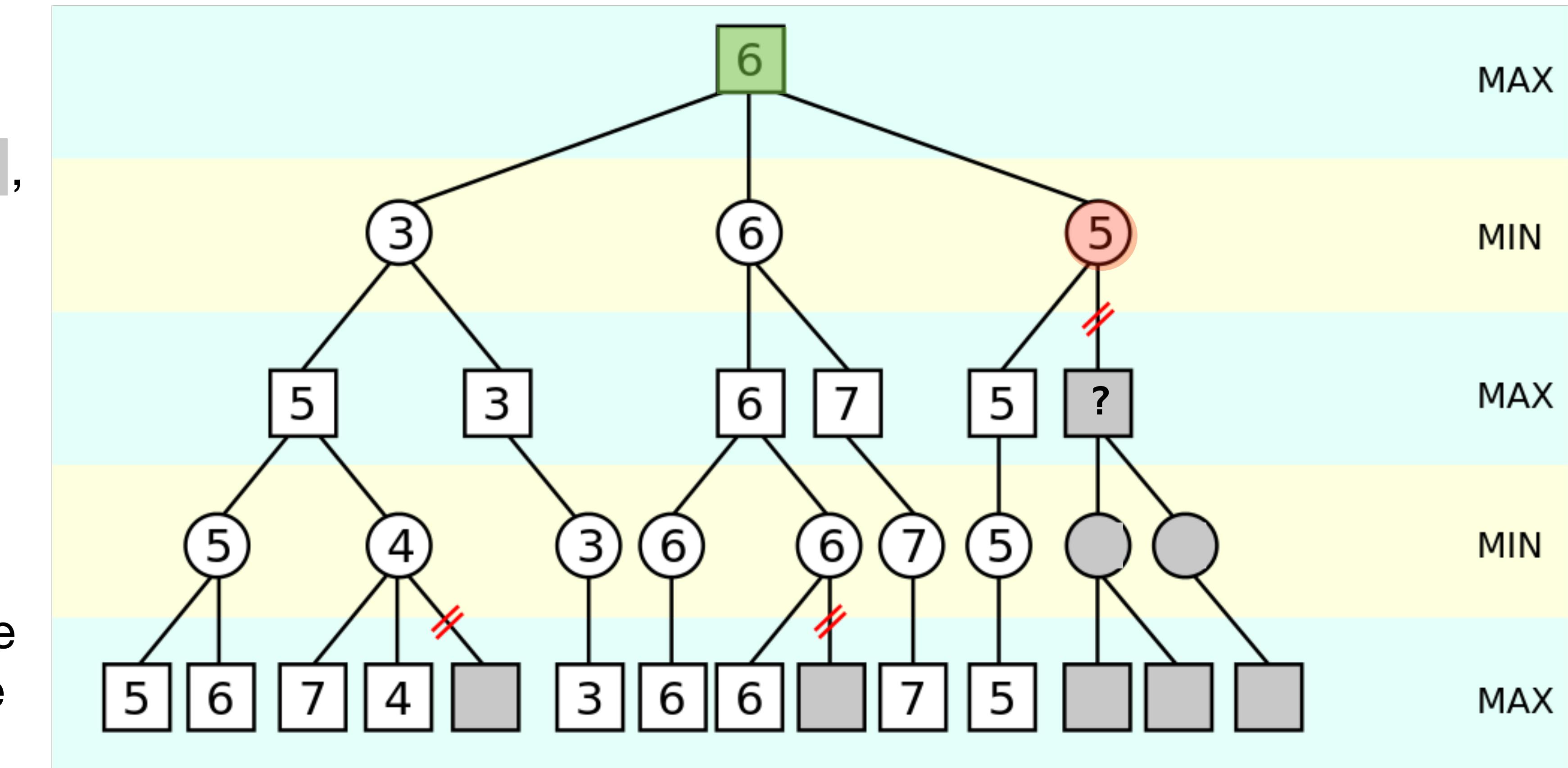
- C'est un calcul de minimum.
- Sa valeur ne peut que diminuer, avant d'évaluer ?, on sait qu'elle est  $\leq 4$
- La valeur de 4 ne sert qu'à mettre à jour le calcul du maximum au niveau supérieur
- Comme  $4 \leq 5$ , la valeur de 5 ne changera pas quelque soit la valeur de 4



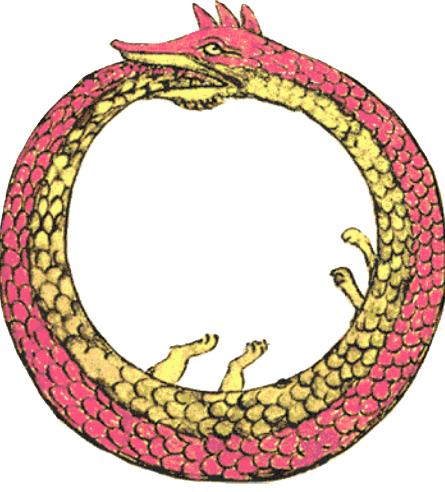
## Arrêtons-nous au calcul de 5



- C'est un calcul de minimum.
- Sa valeur ne peut que diminuer, avant d'évaluer ?, on sait qu'elle est  $\leq 5$
- La valeur de 5 ne sert qu'à mettre à jour le calcul du maximum au niveau supérieur
- Comme  $5 \leq 6$ , la valeur de 6 ne changera pas quelque soit la valeur de 5

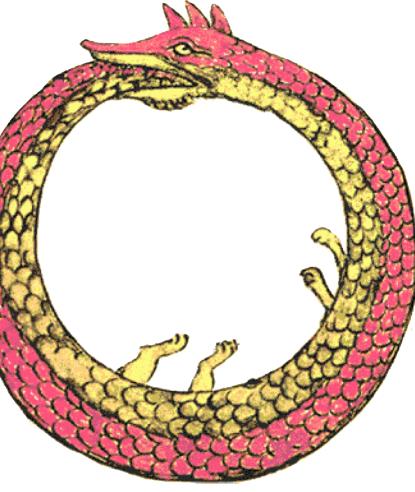


# Gain lié à l'élagage ?



- Cela dépend de l'ordre dans lequel les coups sont considérés.
- Dans le meilleur cas, il permet de doubler la profondeur d'exploration
- Dans le pire cas, il n'apporte aucune amélioration
- Trouver une bonne heuristique pour l'ordre permet d'accélérer considérablement l'algorithme.

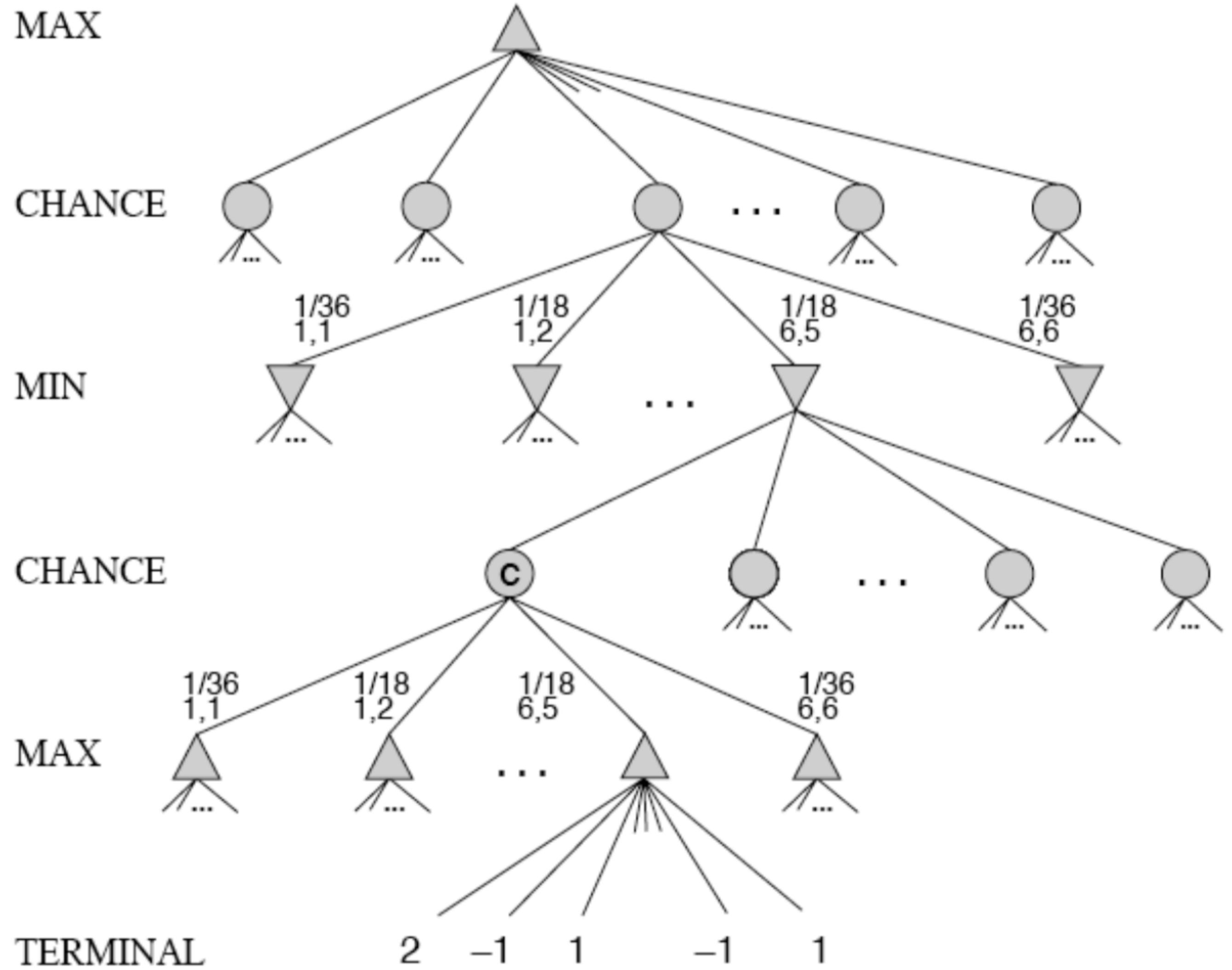
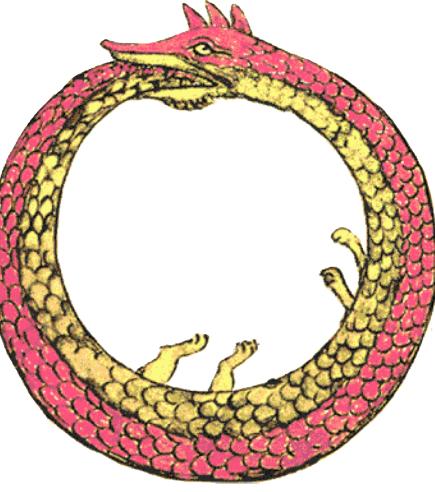
# Backgammon



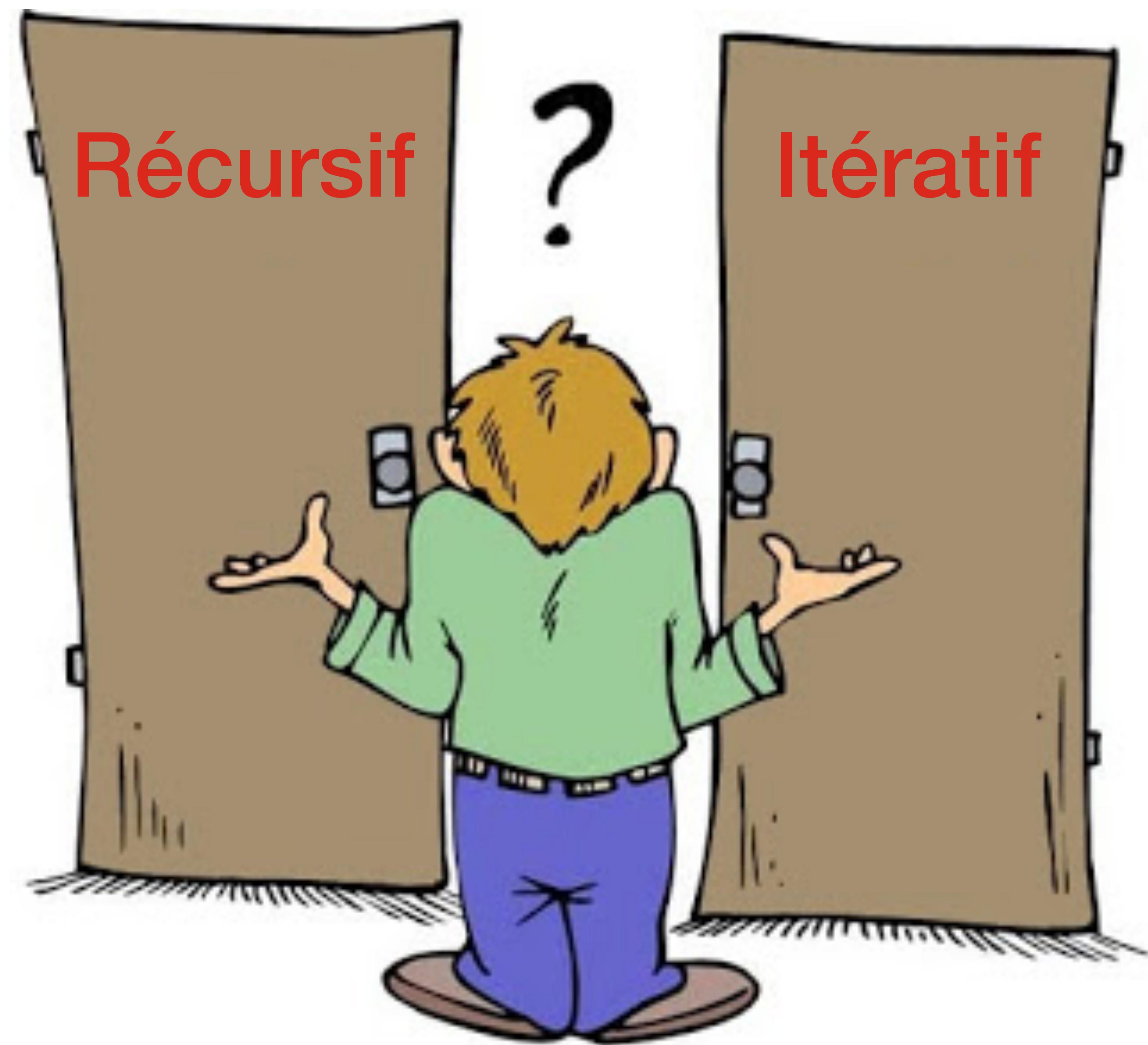
- Le lancer de dés introduit du hazard dans l'arbre des coups successifs possibles
- On intercale des moyennes pondérées par les probabilités  $1/18$  ou  $1/36$  entre les noeuds « Min » et « Max » de l'algorithme



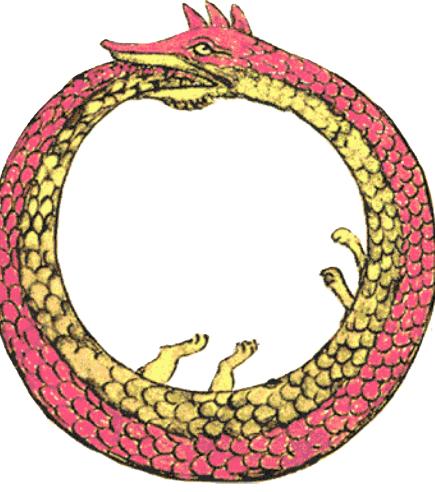
# Minimax pour backgammon



## 2.6. Récursif



# Factorielle

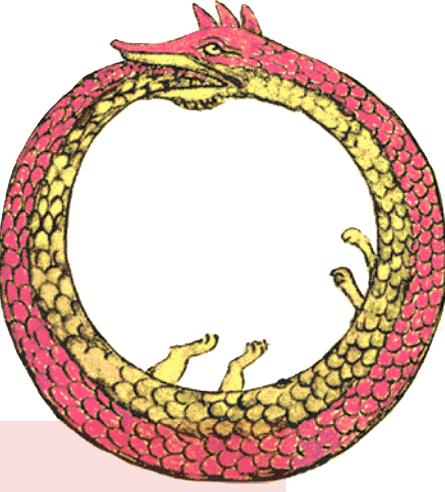


- Avec un seul appel récursif, la fonction récursive est équivalente à une simple boucle.
- Attention, les calculs sont effectués dans l'ordre inverse
  - Récursif : (((((1).2).3)....(n-1)).n)
  - Itératif : 1.n.(n-1)....3.2

```
fonction factorielle(n)
    si n vaut 0 alors
        retourner 1
    sinon
        retourner
            n x factorielle(n-1)
    fin si
```

```
fonction factorielle(n)
    r ← 1
    tant que n > 1
        r ← r x n
        décrémenter n
    fin tant que
    retourner r
```

# Fibonacci

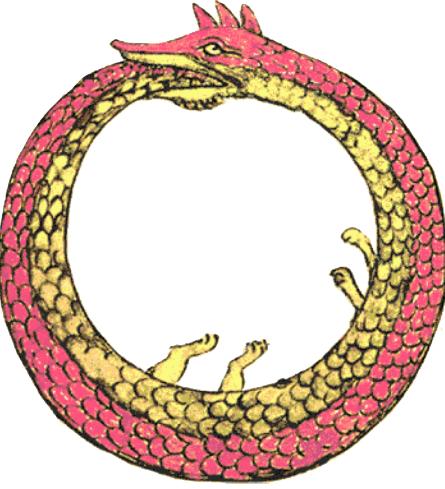


- Version itérative de Fibonacci plus complexe à écrire
- Complexité itérative bien meilleure
  - Récursif en  $O(\phi^n)$
  - Itératif en  $O(n)$

```
fonction Fibonacci(n)
  si n vaut 0 ou 1 alors
    retourner n
  sinon
    retourner Fibonacci(n-1)
      + Fibonacci(n-2)
  fin si
```

```
fonction Fibonacci(n)
  si n vaut 0 ou 1 alors
    retourner n
   $F_{n-1} \leftarrow 0$ 
   $F_n \leftarrow 1$ 
  pour i de 2 à n
     $F_{n-2} \leftarrow F_{n-1}$ 
     $F_{n-1} \leftarrow F_n$ 
     $F_n \leftarrow F_{n-1} + F_{n-2}$ 
  fin pour
  retourner  $F_n$ 
```

# Somme des n premiers entiers



- L'algorithme du chapitre 1 peut-être écrit itérativement, récursivement ou en utilisant la formule de Gauss
- L'approche récursive est-elle une bonne idée ?

```
long sumRec(int n) {  
    if(n<=0) return 0;  
    return n + sumRec(n-1);  
}
```

```
long sumIter(int n) {  
    long sum = 0;  
    for (int i=0; i<=n; ++i)  
        sum += i;  
    return sum;  
}
```

```
long sumGauss(int n) {  
    return (long)n*(n+1)/2;  
}
```

The screenshot shows a debugger interface with the following details:

- Project:** playground > main.cpp
- Code Editor:** The file main.cpp is open, showing a recursive sum function. The line `return n + sumRec(n-1);` is highlighted in blue and has a red lightning bolt icon to its left, indicating it is the current instruction being executed.
- Variables:** The variable `n` is shown with a value of `25267`.
- Exception:** An `EXC_BAD_ACCESS` exception is displayed, with the address `0x7ffee048dff8`.
- Frames:** The stack trace shows multiple frames for `sumRec(int)` at line 9 of `main.cpp`. The top frame is selected.
- Watches:** No watches are currently defined.

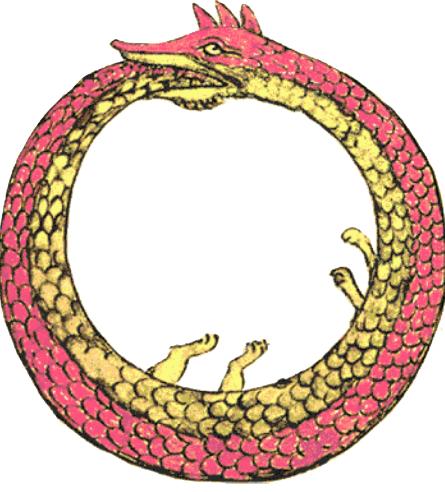
The interface includes standard debugger controls like Debugger, Console, and LLDB, along with navigation and search tools.

```
using namespace std;

long sumRec(int n) {    n: 25267
    if(n<=0) return 0;
    return n + sumRec(n-1);    n: 25267
}

int main() {
    cout << sumRec( n: 200000 );
}
```

# Pile d'appels (call stack)



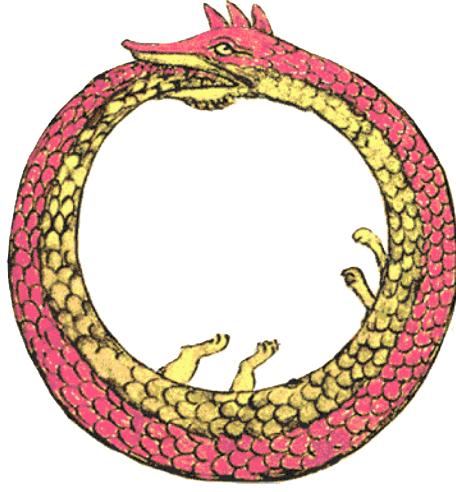
- A chaque appel de fonction le programme empile dans une structure de pile
  - Les variables locales de fonction
  - L'adresse de retour où sauter en sortant de la fonction
- A chaque sortie de fonction, le programme dépile les informations liées à cette fonction et saute à l'adresse indiquée
- Une récursion trop profonde peut dépasser la capacité de cette pile.
- Et pourtant ...

The screenshot shows a C++ development environment with the following details:

- Project Bar:** SommeRecursive | Release
- File Explorer:** playground > main.cpp
- Code Editor:** main.cpp (C++)

```
1 #include <iostream>
2 #include <vector>
3 #include <numeric>
4
5 using namespace std;
6
7 long sumRec(int n) {
8     if(n<=0) return 0;
9     return n + sumRec( n: n-1 );
10 }
11
12 int main() {
13     cout << sumRec( n: 200000 );
14 }
```
- Run Output:** Run: SommeRecursive  
/Users/oce/OneDrive/GitHub/ocuisenaire/playground/cmake-build-release/SommeRecursive  
20000100000  
Process finished with exit code 0
- Favorites:** A vertical sidebar with icons for Run, Debug, Problems, Terminal, CMake, Messages, and TODO.
- Bottom Navigation:** Run, Debug, Problems, Terminal, CMake, Messages, TODO, Event Log

# Récursion terminale (tail récursion)

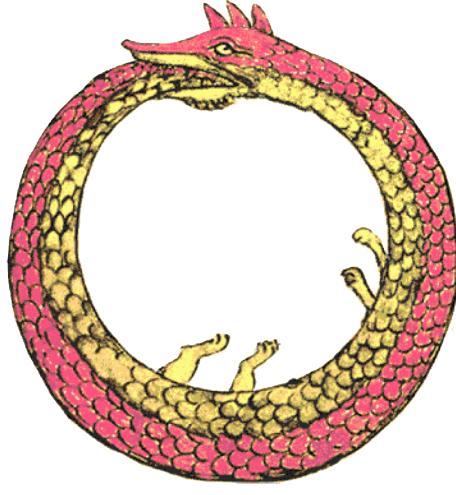


- Quand la toute dernière opération de la fonction récursive est l'appel récursif
- Pas besoin d'empiler de nouvelles infos
- On peut réutiliser celles de la fonction courante
- Mais ... pas en mode debug

```
long sumRec(int n) {  
    if(n<=0) return 0;  
    return n + sumRec(n-1);  
}
```

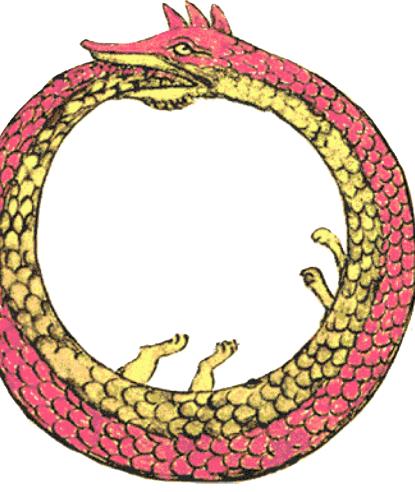
```
long sumTailRec(int n, long r) {  
    if(n==0) return r;  
    return sumTailRec(n-1,n+r);  
}
```

# Itération vs. récursion terminale



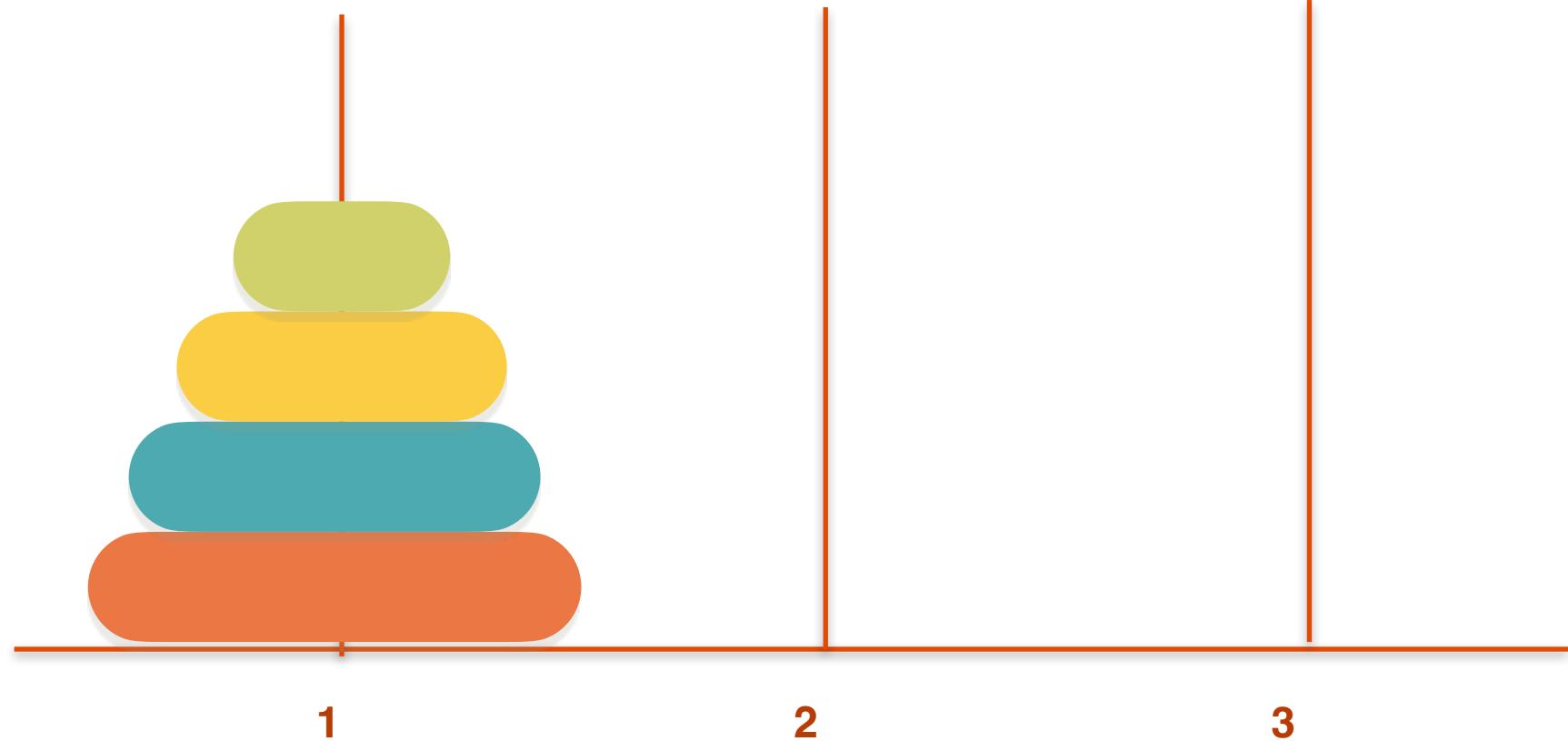
```
fonction recursion(n)
    faire A
    si B, alors
        faire C
        appeler recursion(..)
    fin si
```

```
fonction iteration(n)
    boucler
        faire A
        si non B, alors
            sortir boucle
        fin si
        faire C
    fin boucle
```



# Tour de Hanoï ?

- En général, les fonctions avec plusieurs appels récursifs sont difficiles / impossibles à rendre itératives
- Pour les tours de Hanoï, on dispose cependant d'un algorithme alternatif
  - Le + petit disque bouge une fois sur 2, toujours dans le même sens
  - L'autre déplacement est imposé ( le + petit disque va sur le + grand)



# Tour de Hanoï itératif

**fonction Hanoi(n)**

bouger le plus petit disque

vers la droite (n pair) // 1 -> 2

ou la gauche (n impair) // 1 -> 3

**boucler**  $2^{n-1}-1$  fois

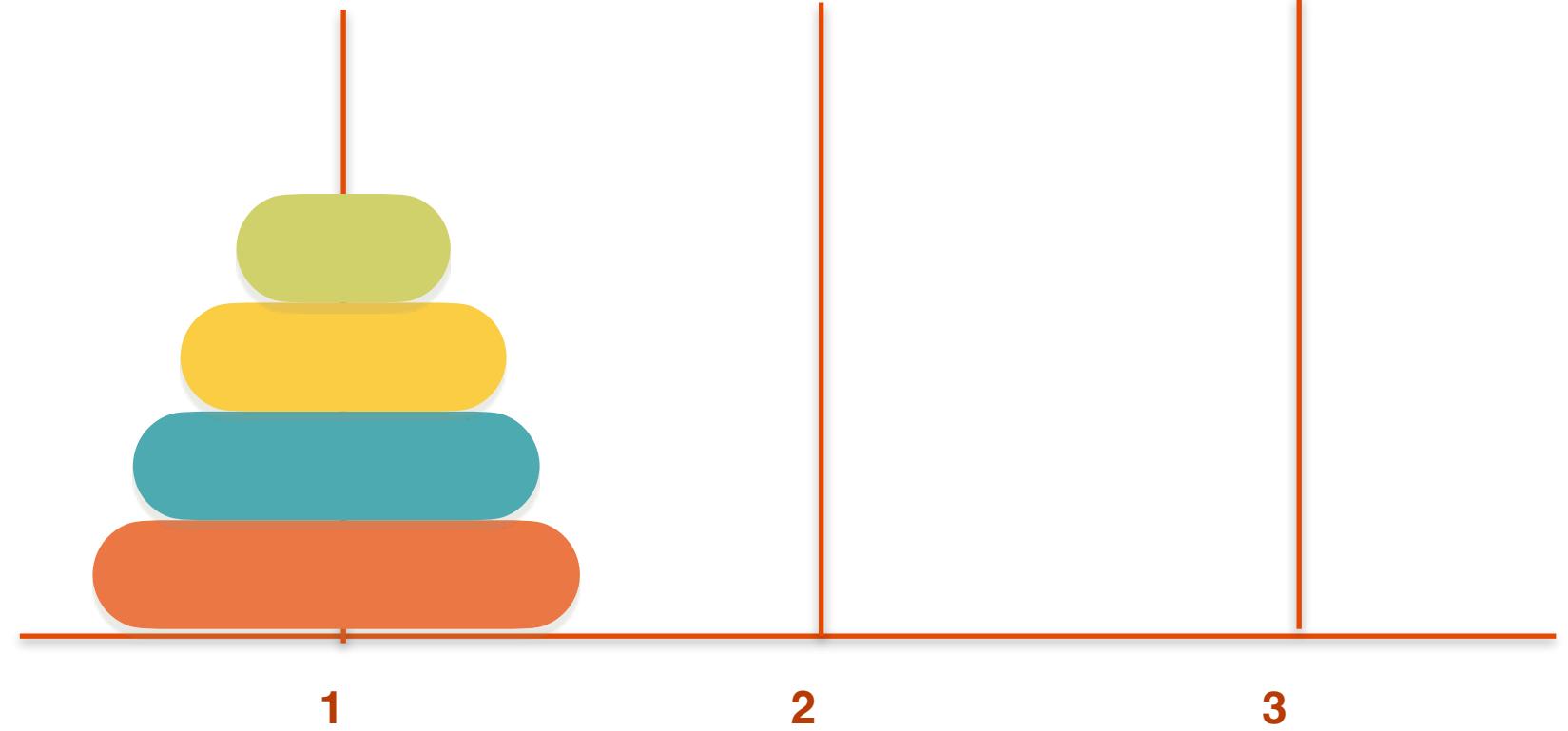
bouger le disque qui n'est pas le plus petit

bouger le plus petit disque

vers la droite (n pair)

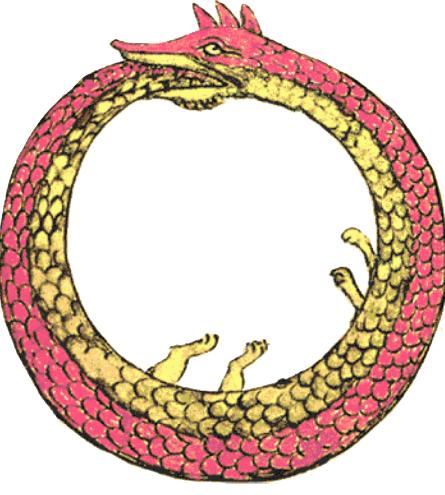
ou la gauche (n impair)

**fin boucler**



# Algorithmes

# Complexités



Factorielle récursif	$O(n)$
Factorielle itératif	$O(n)$
Fibonacci récursif	$O(1.618^n)$
Fibonacci itératif	$O(n)$
PGCD (Euclide)	$O(\log(n))$
Tours de Hanoï récursif	$O(2^n)$
Tours de Hanoï itératif	$O(2^n)$
Permutations	$O(n!)$
Tic Tac Toe	$9!$
Puissance 4, profondeur d'exploration de d tours	$O(7^d)$
Minimax (negamax), m mouvements possibles par tour, profondeur de d tours	$O(m^d)$