LOG-320
Structures de données et algorithmes

Algorithmes MiniMax et Alpha Beta

Plan du cours

Retour sur la récursivité

- Principes de base
- Algorithmes de retour en arrière

Recherche graphe de jeux

- Algorithme MinMax
- Algorithme Alpha-Beta
- Heuristiques

Au jeu!

Humain vs Ordinateur: qui est le meilleur?

Backgammon

- 1992 -1995 : TD-Gammon (IBM) était considéré tout près ou au même niveau que les meilleurs joueurs au monde

Dames

- 1994 : Chinook est devenu le champion du monde après un forfait de Marion Tinsley (décédé une semaine plus tard)
- 1995 : Chinook a conservé son titre en battant Don Lafferty

Othello

- 1997 : Logistello (NEC Research) a battu le champion du monde

Échec

- 1997 : Deep Blue (IBM) a battu Gary Kasparov

Go

- Octobre 2015 : AlphaGo a battu le triple champion européen Fen Hui.
- Mars 2016 : AlphaGo a battu Lee Sedol (considéré comme le meilleur joueur des 10 dernières années) 4-1
- Octobre 2017 : AlphaGo Zero bat AlphaGo 100-0.

Jeux de plateau

Caractéristiques d'un jeu

- Présence d'un adversaire qui introduit un élément d'incertitude.
- Les jeux intéressants sont très complexes
- Le but premier est de déterminer le prochain (meilleur) coup à jouer.
- Les coups doivent être évalués selon leur valeur.

Graphes et jeux

Un programme de jeu comprend :

- Un générateur de mouvements,
- Une fonction de test d'arrêt,
- Une fonction d'évaluation
- Une stratégie de contrôle

Pour formaliser le processus de recherche du meilleur coup, on utilise un arbre de décision appelé ***Arbre de Jeu*** qui est construit pour chaque position *p* comme suit:

- la position *p* constitue la racine de l'arbre.
- si les coups possibles, à partir de la position *p*, donnent les positions $p_1, \ldots, p_n$, alors *p* a comme fils $p_1, \ldots, p_n$.
- les nœuds correspondants aux positions finales sont les feuilles de l'arbre de jeu

Recherche dans les graphes de jeux

PRINCIPES DE LA RÉCURSIVITÉ

Fonctions récursives

Fonction récursive

Fonction dont le calcul nécessite d'invoquer la fonction elle-même

Exemple: Factorielle

Définition

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \times (n - 1)! & \text{si } n > 0 \end{cases}$$

Algorithme

Factoriel(n)

1. **si** n == 0
2. **retourner** 1
3. **retourner** Factoriel(n-1) * n

Fonctions récursives

Règle générale:

Tout processus récursif comprend deux parties:

1) Une formule simple (cas de base, règle de sortie) exécutable sans récursivité donnant généralement une borne inférieure où le processus récursif s'arrête (ex: 0! = 1).
2) Une méthode générale qui réduit un cas particulier en un ou plusieurs cas plus petit, ramenant progessivement par réduction vers le cas de base (ex: $n * (n - 1)!$).

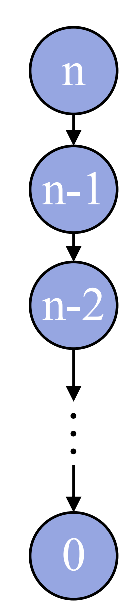
	Factoriel(n)
1.	**si** n == 0
2.	**retourner** 1
3.	**retourner** Factoriel(n-1) * n

Arbre de récursivité

Exemple :

	Factoriel(n)
1.	**si** n == 0
2.	**retourner** 1
3.	**retourner** Factoriel(n-1) * n

Arbre de récursivité



Chaque nœud représente un appel de la fonction récursive, la feuille étant la condition d'arrêt.

Autre exemple classique

Nombre de Fibonacci:

$$f(n) = \begin{cases} n & si \quad n \leq 1 \\ f(n-1) + f(n-2) & sinon \end{cases}$$

Exemple f(6):

Autre exemple: la tour de Hanoi

Problème tel que défini par Édouard Lucas (1892)

N. Claus de Siam a vu, dans ses voyages pour la publication des écrits de l'illustre Fer-Fer-Tam-Tam, dans le grand temple de Bénarès, au-dessous du dôme qui marque le centre du monde, trois aiguilles de diamant, plantées dans une dalle d'airain, hautes d'une coudée et grosses comme le corps d'une abeille. Sur une de ces aiguilles, Dieu enfila au commencement des siècles, 64 disques d'or pur, le plus large reposant sur l'airain, et les autres, de plus en plus étroits, superposés jusqu'au sommet. C'est la tour sacrée du Bralumâ. Nuit et jour, les prêtres se succèdent sur les marches de l'autel, occupés à transporter la tour de la première aiguille sur la troisième, sans s'écarter des règles fixes que nous venons d'indiquer, et qui ont été imposées par Brahma. Quand tout sera fini, la tour et les brahmes tomberont, et ce sera la fin des mondes !

Résumé du problème

Transporter les anneaux empilés en ordre croissant de la tour d'origine (1) vers la tour de destination (3) en utilisant la tour intermédiaire (2)

Une seule règle

En aucun cas, un anneau ne peut être empilé sur un anneau de dimension inférieure.

Source: https://fr.wikipedia.org/wiki/Tours_de_Hanoï

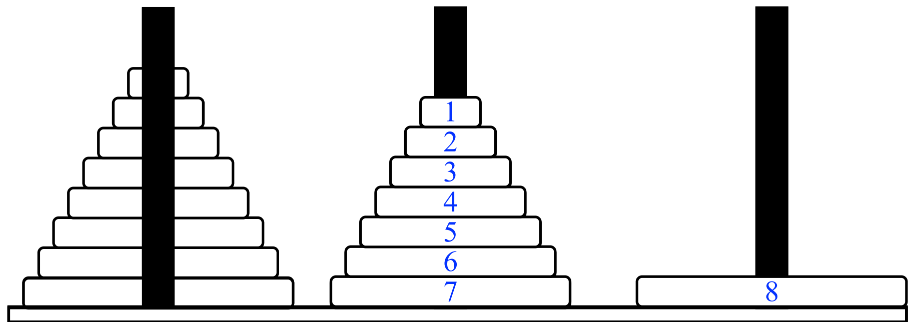
Tour de Hanoi

Problème:

Déplacer 64 anneaux en ordre croissant de (1) vers (3) en utilisant (2).

La solution:

- S'attaquer au plus difficile en premier soit déplacer l'anneau 64 (le + grand)
- L'anneau 64 doit nécessairement passer directement de la tour 1 à la tour 3.



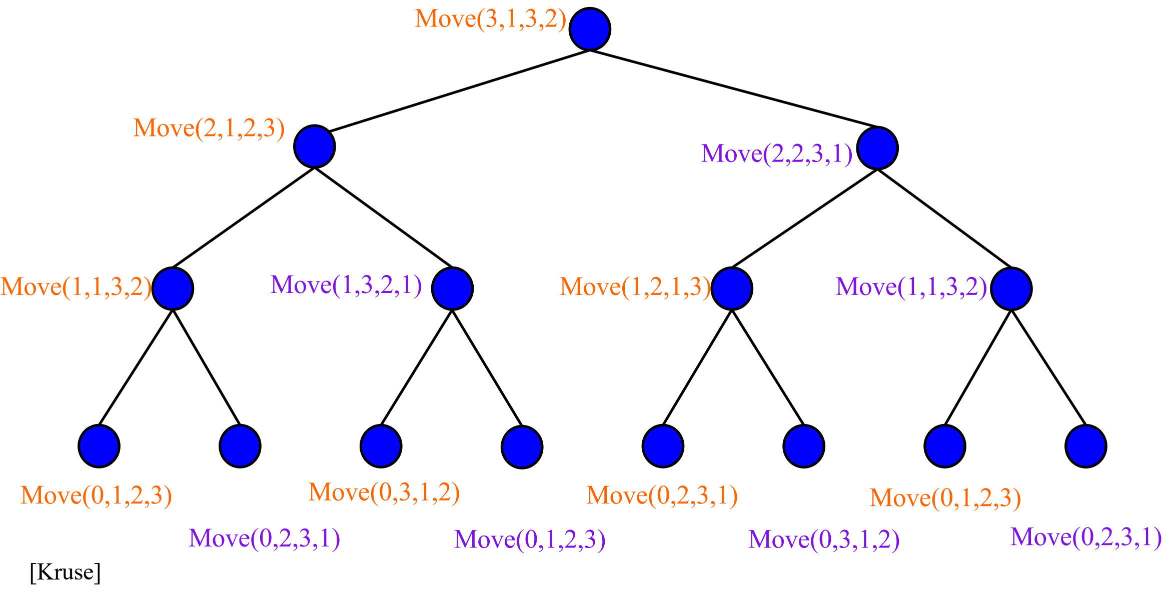
- Faire le même processus pour les 63 autres anneaux récursivement
- Prévoir une règle d'arrêt des appels récursifs.

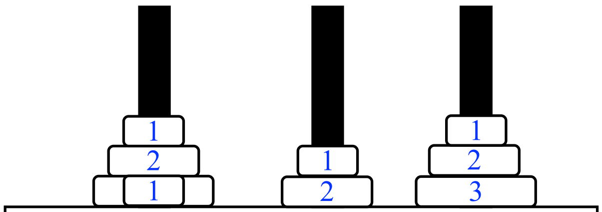
Tour de Hanoi

Algorithme:

	DéplacerAnneau(anneau,tourDépart,tourArrivée,tourTemp)
1.	**si** anneau > 0
2.	DéplacerAnneau(anneau-1,tourDépart,tourTemp,tourArrivée)
3.	Effectuer le déplacement tourDépart vers tourArrivée
4.	DéplacerAnneau(anneau-1,tourTemp,tourArrivée,tourDépart)

L'arbre de récursivité (pour 3 disques):





Recherche dans les graphes de jeux

ALGORITHMES DE RETOUR EN ARRIÈRE

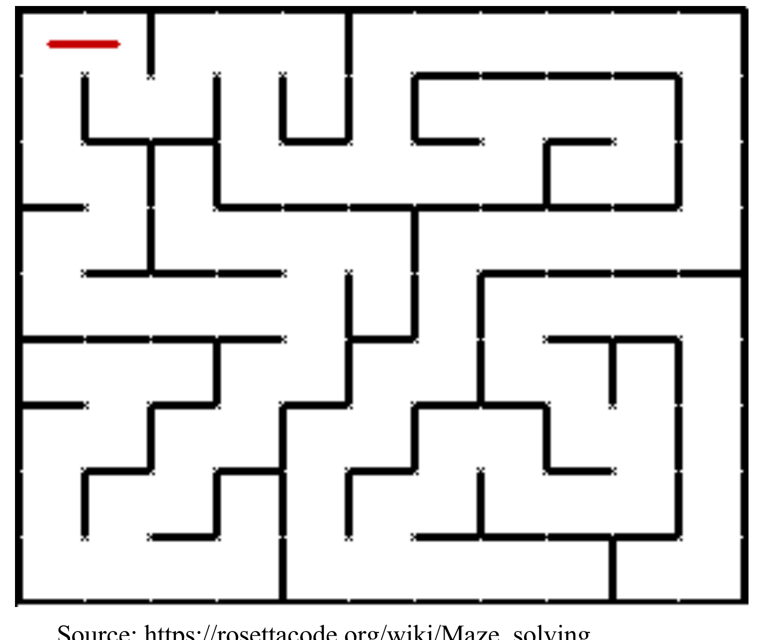
Algorithmes de type « retour en arrière »

Aussi appeler « backtracking », l'algorithme explore toutes les possibilités

- Lorsque l'algorithme se retrouve dans un cul-de-sac, il revient en arrière et explore une nouvelle solution

Algorithme type:

	RetourEnArrière()
1.	**si** solution == Vrai
2.	**retourner** Vrai
3.	**pour chaque** choix possible et valide
4.	Appliquer le choix
5.	**si** RetourEnArrière() == Vrai
6.	**retourner** Vrai
7.	Annuler l'application du choix
8.	**retourner** Faux

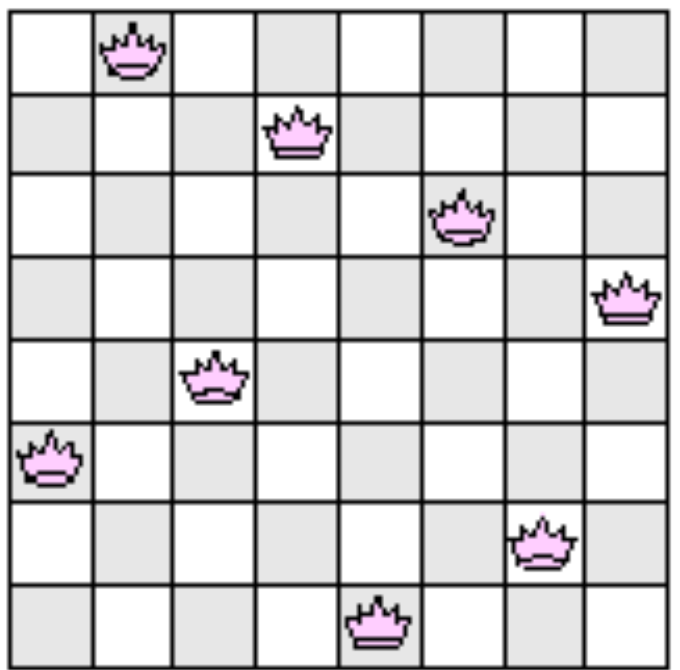


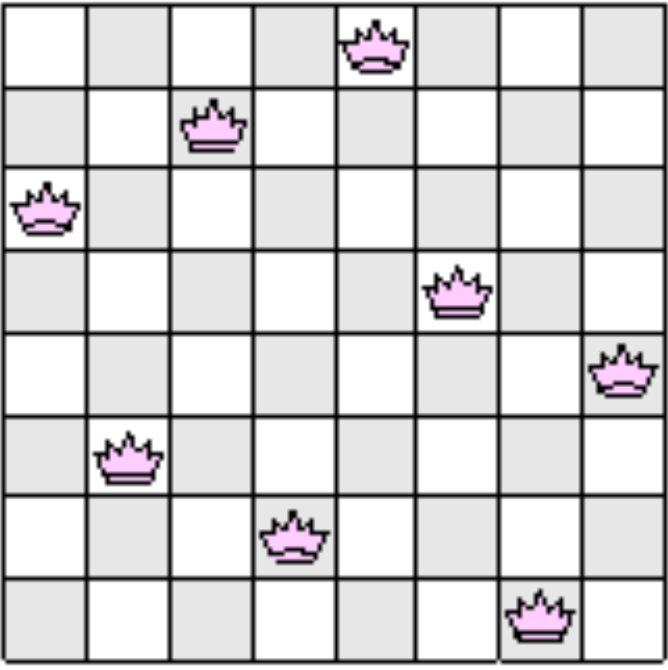
Source: https://rosettacode.org/wiki/Maze_solving

Exemple: le problème des 8 reines

Problème des 8 reines

Placer 8 reines sur un échiquier de 8x8 de telle sorte qu'elles ne puissent s'attaquer entres-elles, selon les règles du jeu d'échecs.



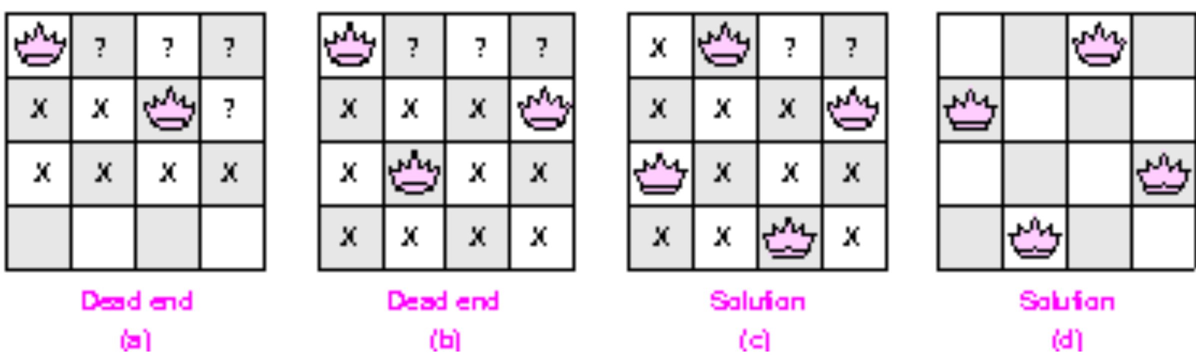


© Avec la permission de Robert Kruse, C. L. Tondo, B. Leung

Exemple: le problème des 8 reines

Exemple à 4 reines:

- Placer une reine sur la 1ière rangée (coin supérieur gauche),
  - ⇒ Les points d'interrogation représentent les cases pas encore essayées.
- Essayer une reine sur la 2e rangée,
  - ⇒ On trouve sur la 3e colonne.
- Conclusion : Cul de sac. On doit reculer. (Figure (a))



© Avec la permission de Robert Kruse, C. L. Tondo, B. Leung

Exemple: le problème des 8 reines

Algorithme:

```
    AjouterReine()
1.    pour chaque position non gardé p sur le plateau
2.      placer la reine à la position p
3.      nombreReines = nombreReines + 1
4.      si nombreReines == 8
5.          Imprimer le plateau
6.      sinon
7.          AjouterReine()
8.      Enlever la reine à la position p
9.      nombreReine = nombreReine – 1
```

Exemple: Jeu des pierres

Source: https://leetcode.com/problems/stone-game-iii/

Considérez les pierres suivantes:

1€

2€

3€

6€

Règles:

- Deux joueurs
- Chaque joueur peut prendre 1,2 ou 3 des premières pierres
- Lorsqu'il n'y a plus de pierre, le gagnant est celui qui à amasser la plus grande valeur

Jeu de pierres : retour en arrière

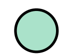
1€

2€

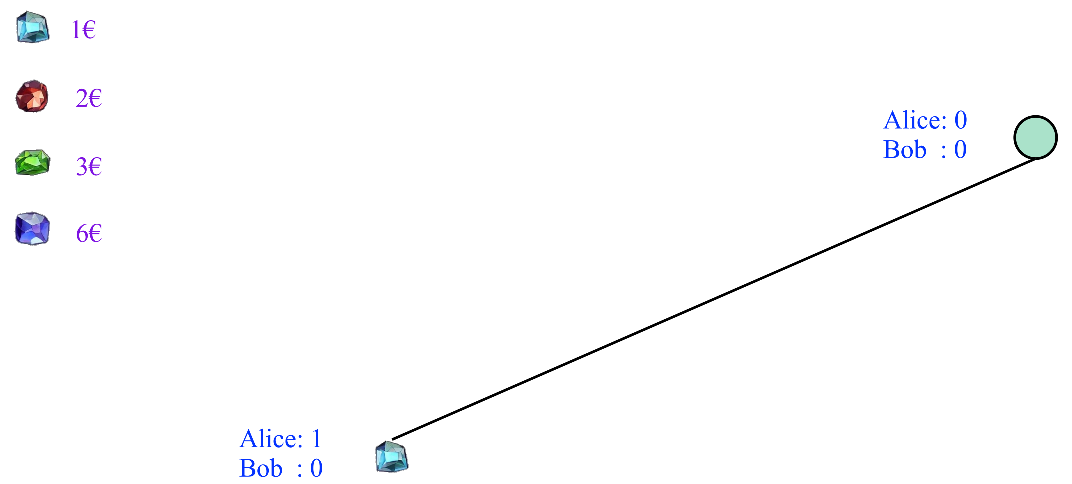
3€

6€

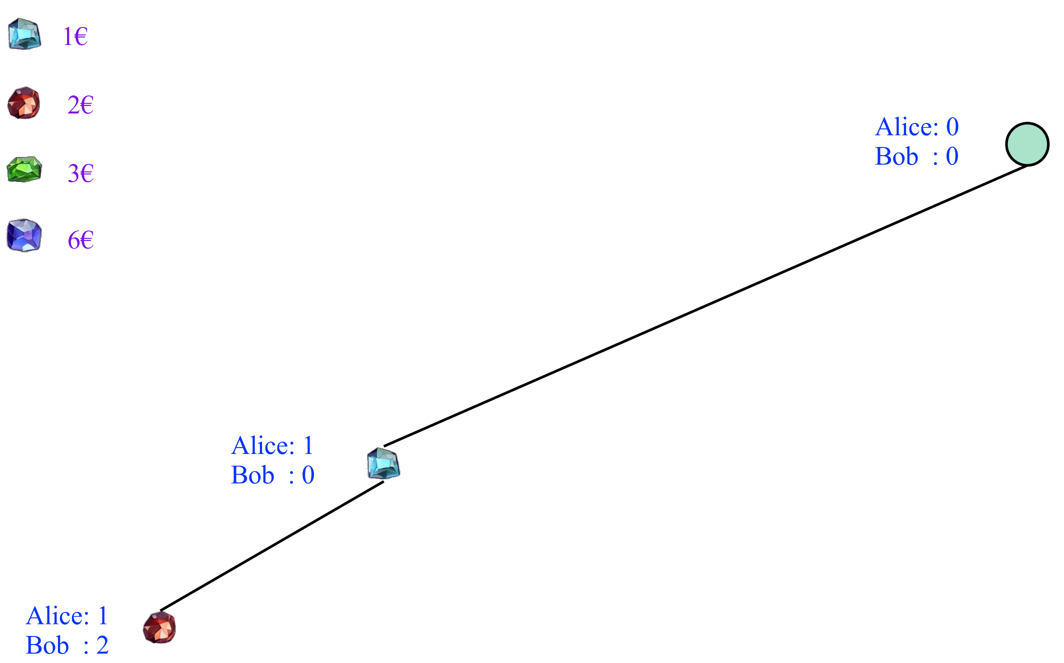
Alice: 0
Bob : 0



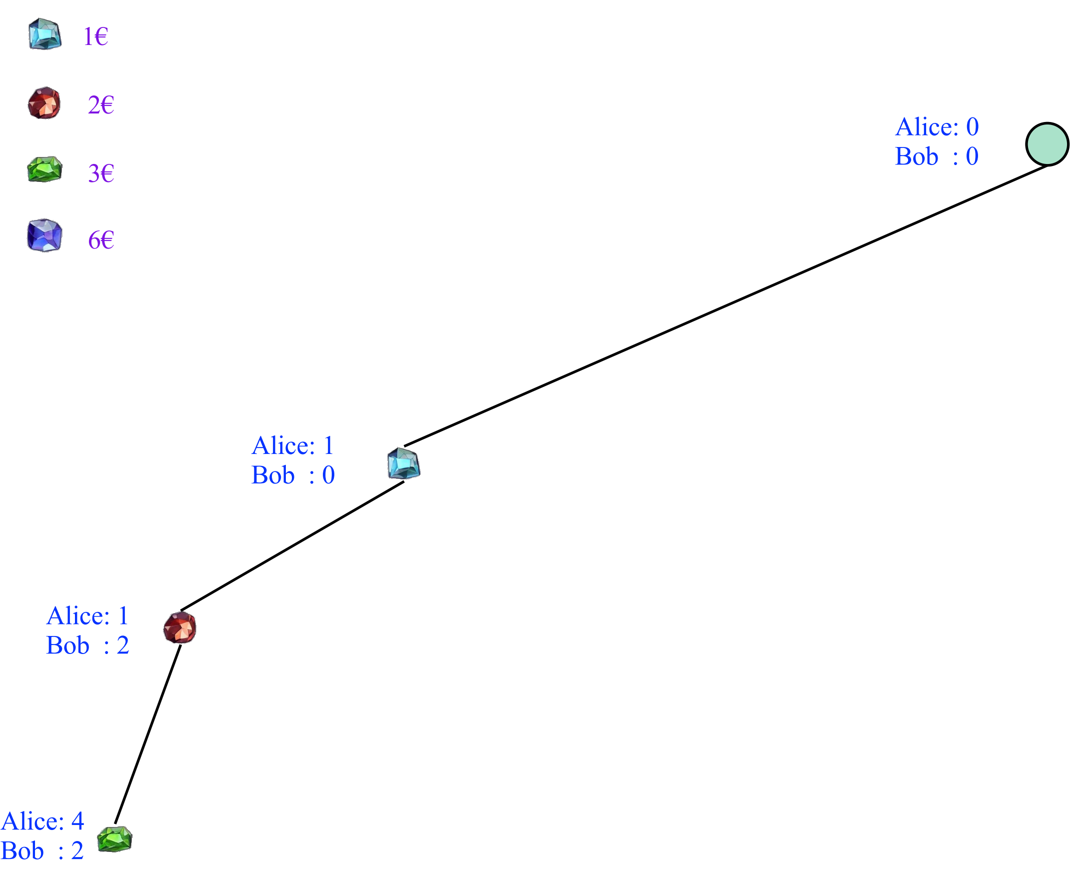
Jeu de pierres : retour en arrière



Jeu de pierres : retour en arrière



Jeu de pierres : retour en arrière



Jeu de pierres : retour en arrière

1€

2€

3€

6€

Alice: 0
Bob : 0

Alice: 1
Bob : 0

Alice: 1
Bob : 2

Alice: 4
Bob : 2

Alice: 4
Bob : 8

Jeu de pierres : retour en arrière

1€

2€

3€

6€

Alice: 0
Bob : 0

Alice: 1
Bob : 0

Alice: 1
Bob : 2

Alice: 4
Bob : 2

Alice: 10
Bob : 2

Alice: 4
Bob : 8

Jeu de pierres : retour en arrière

1€

2€

3€

6€

Alice: 0
Bob : 0

Alice: 1
Bob : 0

Alice: 1
Bob : 2

Alice: 1
Bob : 5

Alice: 4
Bob : 2

Alice: 10
Bob : 2

Alice: 4
Bob : 8

Jeu de pierres : retour en arrière

1€

2€

3€

6€

Alice: 0
Bob : 0

Alice: 1
Bob : 0

Alice: 1
Bob : 2

Alice: 1
Bob : 5

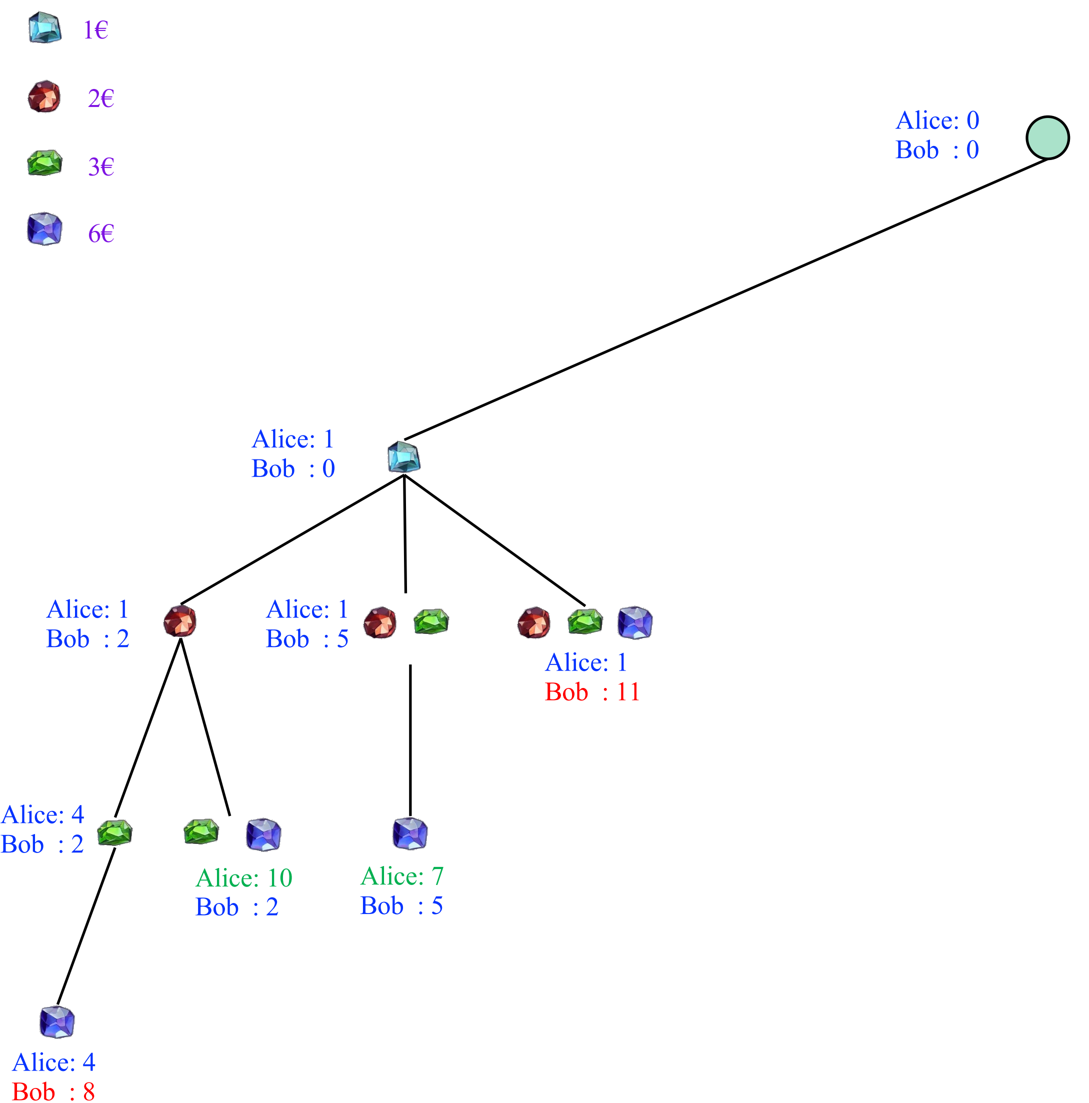
Alice: 4
Bob : 2

Alice: 10
Bob : 2

Alice: 7
Bob : 5

Alice: 4
Bob : 8

Jeu de pierres : retour en arrière



Jeu de pierres : retour en arrière

1€
2€
3€
6€

Alice: 0
Bob : 0

Alice: 1
Bob : 0

Alice: 3
Bob : 0

Alice: 1
Bob : 2

Alice: 1
Bob : 5

Alice: 1
Bob : 11

Alice: 4
Bob : 2

Alice: 10
Bob : 2

Alice: 7
Bob : 5

Alice: 4
Bob : 8

Jeu de pierres : retour en arrière

1€

2€

3€

6€

Alice: 0
Bob : 0

Alice: 1
Bob : 0

Alice: 3
Bob : 0

Alice: 1
Bob : 2

Alice: 1
Bob : 5

Alice: 1
Bob : 11

Alice: 3
Bob : 3

Alice: 4
Bob : 2

Alice: 10
Bob : 2

Alice: 7
Bob : 5

Alice: 4
Bob : 8

Jeu de pierres : retour en arrière

1€

2€

3€

6€

Alice: 0
Bob : 0

Alice: 1
Bob : 0

Alice: 3
Bob : 0

Alice: 1
Bob : 2

Alice: 1
Bob : 5

Alice: 1
Bob : 11

Alice: 3
Bob : 3

Alice: 4
Bob : 2

Alice: 10
Bob : 2

Alice: 7
Bob : 5

Alice: 9
Bob : 3

Alice: 4
Bob : 8

Jeu de pierres : retour en arrière

1€

2€

3€

6€

Alice: 0
Bob : 0

Alice: 1
Bob : 0

Alice: 3
Bob : 0

Alice: 1
Bob : 2

Alice: 1
Bob : 5

Alice: 1
Bob : 11

Alice: 3
Bob : 3

Alice: 3
Bob : 9

Alice: 4
Bob : 2

Alice: 10
Bob : 2

Alice: 7
Bob : 5

Alice: 9
Bob : 3

Alice: 4
Bob : 8

Jeu de pierres : retour en arrière

1€

2€

3€

6€

Alice: 0
Bob : 0

Alice: 1
Bob : 0

Alice: 3
Bob : 0

Alice: 6
Bob : 0

Alice: 1
Bob : 2

Alice: 1
Bob : 5

Alice: 1
Bob : 11

Alice: 3
Bob : 3

Alice: 3
Bob : 9

Alice: 4
Bob : 2

Alice: 10
Bob : 2

Alice: 7
Bob : 5

Alice: 9
Bob : 3

Alice: 4
Bob : 8

Jeu de pierres : retour en arrière

1€

2€

3€

6€

Alice: 0
Bob : 0

Alice: 1
Bob : 0

Alice: 3
Bob : 0

Alice: 6
Bob : 0

Alice: 1
Bob : 2

Alice: 1
Bob : 5

Alice: 1
Bob : 11

Alice: 3
Bob : 3

Alice: 3
Bob : 9

Alice: 6
Bob : 6

Alice: 4
Bob : 2

Alice: 10
Bob : 2

Alice: 7
Bob : 5

Alice: 9
Bob : 3

Alice: 4
Bob : 8

Jeu de pierres : code du retour en arrière

```
bool Solve(int* stones, int size, int indexFirst)
{

    if(indexFirst >= size)
        return true;

    for(int nPick=1; nPick<=3; nPick++){
        int index = indexFirst+nPick-1;

        if(index >= size)
            break;

        solution.push_back(nPick);
        if(Solve(stones,size,index+1)){
            PrintSolution(stones,size);
        }
        solution.pop_back();

    }
    return false;

}
```

Recherche dans les graphes de jeux

ALGORITHME MINIMAX

Graphes et jeux

On suppose que les deux joueurs exécutent alternativement un seul coup.

Les niveaux des nœuds pour les joueurs adversaires s'alternent dans *l'arbre de jeu*.

1 coup

demi-coup

100 0 -100 0 100 -100 -100 100 0 -100 100 100

Minimax

Deux joueurs:

- MAX : Le joueur à qui on veut faire gagner la partie. Il va toujours tenter de maximiser son avantage.
- MIN : Le joueur adverse. Il tente de minimiser l'avantage du joueur MAX.

Les deux joueurs veulent gagner.

On suppose que le joueur MIN a les mêmes connaissances sur l'état du jeu que le joueur MAX.

Le joueur MIN joue toujours le pire coup pour MAX.

Minimax

L'algorithme MINIMAX, dû à Von Neumann, a comme but l'élaboration d'une **stratégie optimale** pour le joueur MAX.

À chaque tour, le joueur MAX va choisir le coup qui va maximiser son score, tout en minimisant les **bénéfices** de l'adversaire.

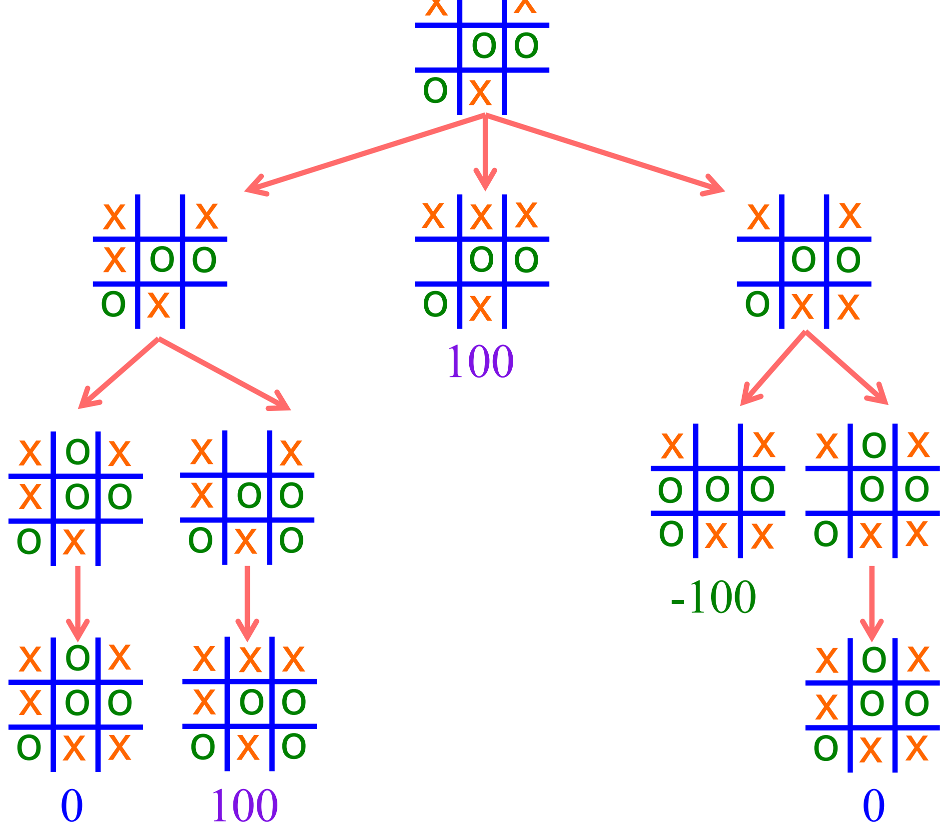
Ces bénéfices sont évalués en termes de la ***fonction d'évaluation statique*** utilisée pour apprécier les positions pendant le jeu.

Exemple: Tic Tac Toe

On considère que pendant une partie de *Tic Tac Toe* on est arrivé dans la position présentée dans la racine de l`arbre. X = Max

La valeur de la fonction d'évaluation statique :

- +100 pour la victoire de MAX,
- 0 pour une partie nulle, et
- -100 pour la victoire de MIN



Minimax

Exemple. On peut voir ce processus d`association des valeurs aux nœuds intermédiaires, valeurs marquées en *italiques*.

Minimax

Un algorithme qui implémente le joueur MAX comportera les phases suivantes :

1. Construire l'arbre de jeu pour *P* niveaux
2. Visiter l'arbre de jeu *par niveaux* en remontant des nœuds terminaux jusqu'à la racine:
   - si le nœud courant *p* représente une position terminale, on lui associe la valeur de la fonction d'évaluation statique, f(*p*)
   - s'il s'agit d'une position intermédiaire et d'un nœud MIN on lui associe la plus petite des valeurs associées à ses fils
   - s'il s'agit d'une position intermédiaire et d'un nœud MAX on lui associe la valeur maximale associée à ses fils

Minimax

3. Quand on a réussi à associer une valeur à la racine (appelée valeur *MiniMax*), on fait le choix de coup qui mène vers le fils qui a cette valeur maximale.
4. S'il y a plusieurs fils qui ont la même valeur, alors on en choisit un au hasard ou
   1. celui qui conduit à la victoire ou
   2. celui qui conduit à une situation finale dans le plus petit nombre de coups ou
   3. un autre critère selon les spécificités du jeu

Jeu de pierres : minimax

1€

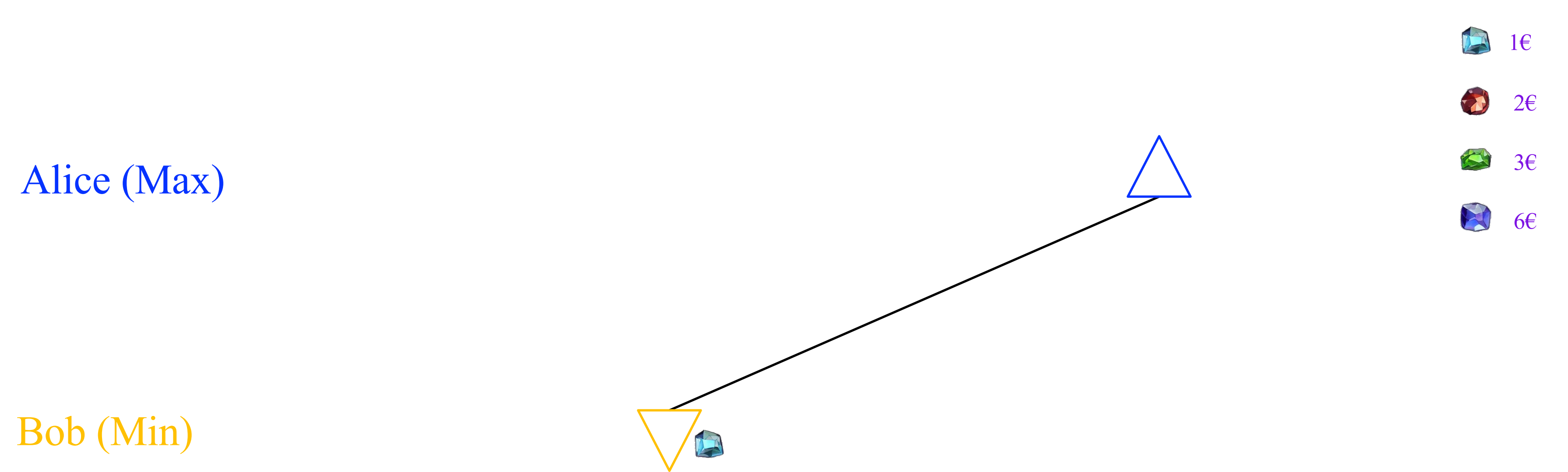
2€

3€

6€

Alice (Max)

Jeu de pierres : minimax



Jeu de pierres : minimax

1€

2€

3€

6€

Alice (Max)

Bob (Min)

Alice (Max)

Jeu de pierres : minimax

1€

2€

3€

6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€

2€

3€

6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€

2€

3€

6€

Alice (Max)

Bob (Min)

Alice (Max)

-100

Bob (Min)

-100

Alice (Max)

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

-100

Bob (Min)

Alice: 10
Bob : 2
Eval: 100

-100

Alice (Max)

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

-100
100

Bob (Min)

Alice: 10
Bob : 2
Eval: 100

-100

Alice (Max)

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€

2€

3€

6€

Alice (Max)

Bob (Min)

100

Alice (Max)

-100

100

Bob (Min)

Alice: 10
Bob : 2
Eval: 100

-100

Alice (Max)

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

100

-100

100

Alice: 10
Bob : 2
Eval: 100

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

100

-100

100

Bob (Min)

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

-100

Alice (Max)

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

100

-100

100

100

-100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

100
100
-100
100
100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

100 100

Alice: 1
Bob : 11
Eval: -100

-100 100 100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

100
100
-100

Alice: 1
Bob : 11
Eval: -100

-100
100
100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

-100

Alice (Max)

100

100

-100

Alice: 1
Bob : 11
Eval: -100

Bob (Min)

-100

100

100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

Alice (Max)

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€

2€

3€

6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

-100

100

100

-100

Alice: 1
Bob : 11
Eval: -100

-100

100

100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

-100

100

100

-100

Alice: 1
Bob : 11
Eval: -100

-100

100

100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

-100

100

100

-100

Alice: 1
Bob : 11
Eval: -100

-100

100

100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

Alice: 9
Bob : 3
Eval: 100

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

-100
100
100
-100
-100
100
100
100
-100

Alice: 1
Bob : 11
Eval: -100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

Alice: 9
Bob : 3
Eval: 100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

-100

100

100

-100

100

Alice: 1
Bob : 11
Eval: -100

-100

100

100

100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

Alice: 9
Bob : 3
Eval: 100

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€

2€

3€

6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

-100

100

100

-100

100

Alice: 1
Bob : 11
Eval: -100

Alice: 3
Bob : 9
Eval: -100

-100

100

100

100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

Alice: 9
Bob : 3
Eval: 100

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

-100

100

100

-100

100

-100

Alice: 1
Bob : 11
Eval: -100

Alice: 3
Bob : 9
Eval: -100

-100

100

100

100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

Alice: 9
Bob : 3
Eval: 100

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

-100 -100

100 100 -100 100 -100

Alice: 1
Bob : 11
Eval: -100

Alice: 3
Bob : 9
Eval: -100

-100 100 100 100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

Alice: 9
Bob : 3
Eval: 100

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

-100 -100

100 100 -100 100 -100

Alice: 1
Bob : 11
Eval: -100

Alice: 3
Bob : 9
Eval: -100

Alice: 6
Bob : 6
Eval: 0

-100 100 100 100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

Alice: 9
Bob : 3
Eval: 100

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

-100 -100

100 100 -100 100 -100 0

Alice: 1
Bob : 11
Eval: -100

Alice: 3
Bob : 9
Eval: -100

Alice: 6
Bob : 6
Eval: 0

-100 100 100 100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

Alice: 9
Bob : 3
Eval: 100

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€

2€

3€

6€

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

-100 -100 0

100 100 -100 100 -100 0

Alice: 1
Bob : 11
Eval: -100

Alice: 3
Bob : 9
Eval: -100

Alice: 6
Bob : 6
Eval: 0

-100 100 100 100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

Alice: 9
Bob : 3
Eval: 100

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

Alice (Max)

Bob (Min)

Alice (Max)

Bob (Min)

Alice (Max)

1€
2€
3€
6€

0

-100
-100
0

100
100
-100
100
-100
0

Alice: 1
Bob : 11
Eval: -100

Alice: 3
Bob : 9
Eval: -100

Alice: 6
Bob : 6
Eval: 0

-100
100
100
100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

Alice: 9
Bob : 3
Eval: 100

-100

Alice: 3
Bob : 8
Eval: -100

Jeu de pierres : minimax

1€
2€
3€
6€

Alice (Max)
0

Bob (Min)
-100
-100
0

Coup sélectionné

Alice (Max)
100
100
-100
100
-100
0

Alice: 1
Bob : 11
Eval: -100

Alice: 3
Bob : 9
Eval: -100

Alice: 6
Bob : 6
Eval: 0

Bob (Min)
-100
100
100
100

Alice: 10
Bob : 2
Eval: 100

Alice: 7
Bob : 5
Eval: 100

Alice: 9
Bob : 3
Eval: 100

Alice (Max)
-100

Alice: 3
Bob : 8
Eval: -100

Autre exemple : jeu de Nim

Exemple : Jeu de Nim

Règle du jeu:

- Deux joueurs
- Un certain nombre d’allumettes sont placées sur une table.
- À chaque tour, un joueur doit diviser une pile d’allumettes en deux piles de grandeurs différentes.
- Le premier joueur qui n’a aucune possibilité de coups perd la partie.

Autre exemple : jeu de Nim

MAX
-100
7

MIN
-100 6,1
-100 5,2
-100 4,3

MAX
100 5,1,1
-100 4,2,1
-100 4,2,1
100 3,2,2
-100 3,3,1
-100 4,2,1

MIN
100 4,1,1,1
-100 3,2,1,1
-100 3,2,1,1
-100 3,2,1,1
100 2,2,1,1
100
-100 3,2,1,1
-100 3,2,1,1
-100 3,2,1,1

MAX
100 3,1,1,1,1
-100 2,2,1,1,1
-100
-100 2,2,1,1,1
-100
-100 2,2,1,1,1
-100
-100 2,2,1,1,1
-100
-100 2,2,1,1,1
-100
-100 2,2,1,1,1
-100

MIN
100 2,1,1,1,1,1
100

MiniMax

	Minimax(positionActuelle,joueur)
1.	**si** positionActuelle est finale
2.	retourner f(p)
3.	**si** joueur == Max
4.	maxScore = $-\infty$
5.	**pour** tous les successeurs p_i de PositionActuelle
6.	score = Minimax(p_i,Min)
7.	maxScore = MAX(maxScore,score)
8.	**retourner** maxScore
9.	**si** joueur == Min
10.	minScore = ∞
11.	**pour** tous les successeurs p_i de PositionActuelle
12.	score = Minimax(p_i,Max)
13.	minScore = MIN(minScore,score)
14.	**retourner** minScore

Exemple de comportement étrange

Max

Min

Max

Min

Max

Exercice

Quel coup sera sélectionné par l'algorithme Minimax et quel sera son coût?

A B C

10 7 6 8 12 4 7 6 12 9 14 0 9 5 4 3 12 2 8 2 9 12 5 11 1 5 8 3 14 8 6 2 12 7 8 13

Minimax

Pour les jeux complexes, la solution exacte ne peut pas être trouvée dans un temps raisonnable:

$$\textit{Complexité en temps} = \boldsymbol{O}(b^P)$$

Par exemple, le jeu d'échec a un facteur de branchement b=35 et pour des jeux communs *P*=100, on obtient donc un nombre de traitements de l'ordre 35^{100}. Ce nombre est trop grand (pour faire une comparaison, le nombre des atomes de l'univers est de l'ordre de 10^{100}).

Il est nécessaire de trouver des stratégies plus pratiques.

Minimax à profondeur limitée

Pour plusieurs jeux, il est impossible d'explorer jusqu'aux feuilles de l'arbre.

Exemples:

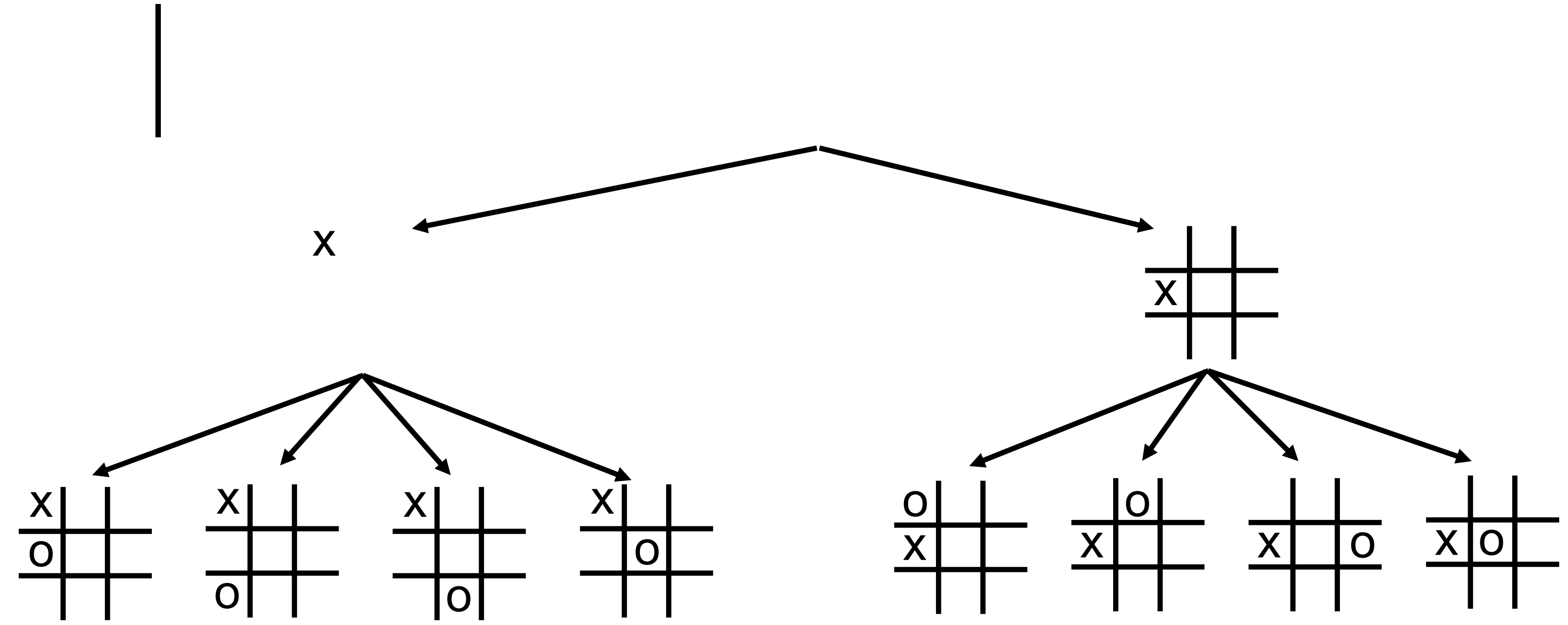
- Jeu d'échec
- Jeu de « breakthrough »

Solution: Arrêter la recherche à un niveau pré-déterminé.

- Demande une fonction d'évaluation statique plus évoluée.

Minimax à profondeur limitée

On veut limiter la recherche à deux demi-coups.



Quel coup choisir?

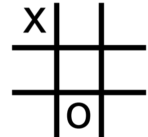
Minimax à profondeur limitée

Il faut définir une heuristique

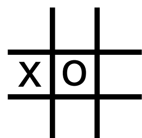
– F(plateau) = X(plateau) – O(plateau)

où X(plateau) est le nombre de lignes gagnantes possibles pour le joueur (MAX).
O(plateau) est le nombre de lignes gagnantes possibles pour l'adversaire (MIN).

Exemples:



X ont 6 possibilités de gagner;
O ont 5 possibilités de gagner.
F(plateau) = 6 – 5 = 1



X ont 4 possibilités de gagner;
O ont 6 possibilités de gagner.
F(plateau) = 4 – 6 = -2

Minimax à profondeur limitée

MAX

MIN

MAX

-1

-1

-2

6-5 = 1

5-5 = 0

6-5 = 1

4-5 = -1

5-6 = -1

6-6 = 0

6-6 = 0

4-6 = -2

Minimax à profondeur limitée

Complexité:

- $O(b^h)$ où h est la hauteur de l'arbre et b est le facteur de branchement

Jeu d'échec

- 4 niveaux : joueur débutant
- 8 niveaux : bon programme, comparable à un joueur de niveau «maître».
- 12 niveaux : Deep-Blue et Kasparov

Recherche dans les graphes de jeux

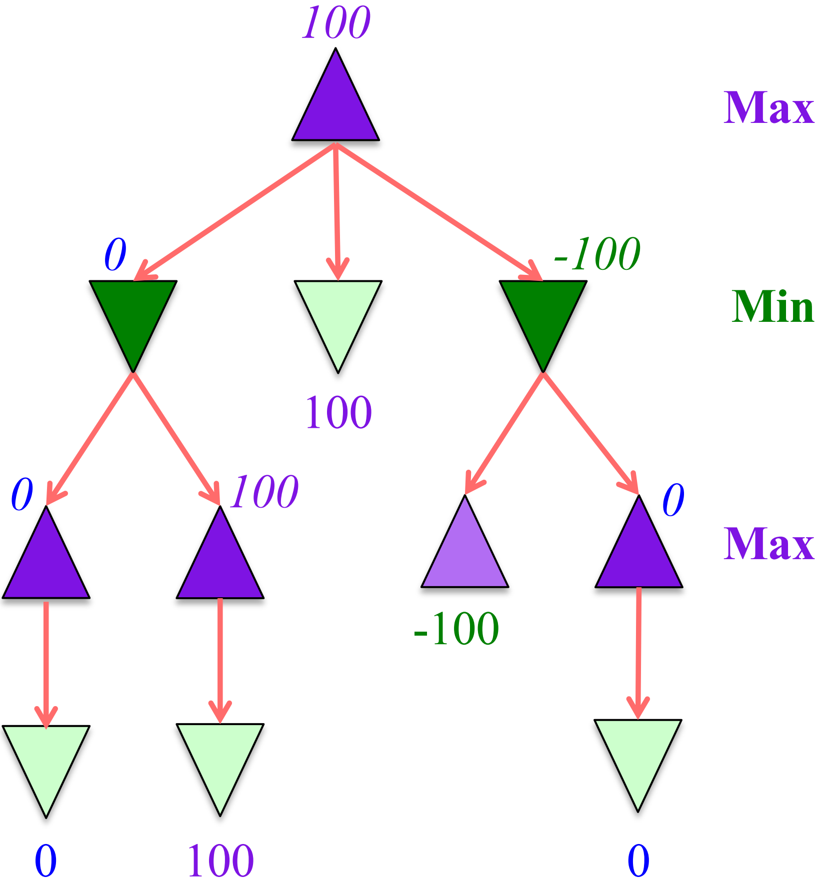
ALGORITHME ALPHA-BETA

Minimax: Principe d'élagage

La racine (MAX) a un fils avec la valeur maximale possible (+100)

Il n'est donc pas nécessaire de regarder les valeurs des autres fils car elles ne peuvent pas changer la valeur MiniMax

On peut donc arrêter l'exploration d'autres fils et accélérer la recherche.



Minimax – Élagage Alpha-Beta

Principe:

- Ne pas générer un nœud successeur lorsqu'il est évident que ce nœud ne sera pas choisi.
- Une valeur α est associée à chaque nœud MAX. Cette valeur correspond au coût du meilleur successeur du nœud.
- Une valeur β est associée à chaque nœud MIN. Cette valeur correspond au coût du pire successeur du nœud.
- Valeurs initiales :
  - $\alpha = -\infty$
  - $\beta = \infty$

Minimax – Élagage Alpha-Beta

Il y a deux règles à suivre:

- on interrompt la recherche d'un nœud *n* de type MAX si ***α(n) est plus grand que β(Père(n))***, parce que c'est une valeur qui ne peut pas changer la valeur **β** du nœud père de *n* (qui est un nœud de type MIN);
- on interrompt la recherche d'un nœud *m* de type MIN si ***β(m) est plus petit que α(Père(m))*** parce que cette valeur ne peut pas influencer le processus de détermination du maximum qui établit la valeur ***α*** de son père.

Minimax – Élagage Alpha-Beta: exemple

α=4

β=4

β=3

β=2

α=4

α=5

α=3

α=2

2 4 5 6 3 0 4 5 2 1 4 3

Minimax – Élagage Alpha-Beta: exemple 2

Quel coup sera sélectionné par l’algorithme Alpha-Beta et quel sera son coût?

A B C

10 7 6 8 12 4 7 6 12 9 14 0 9 5 4 3 12 2 8 2 9 12 5 11 1 5 8 3 14 8 6 2 12 7 8 13

Minimax – Élagage Alpha-Beta: exemple 3

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\alpha = 3$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\alpha = 6$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\beta = 6$

$\alpha = 6$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\beta = 6$

$\alpha = 6$ $\alpha = 8$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\beta = 6$

$\alpha = 6$ $\alpha = 8$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\beta = 6$

$\alpha = 6$ $\alpha = 8$ $\alpha = 3$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\beta = 6$

$\alpha = 6$ $\alpha = 8$ $\alpha = 12$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\alpha = 6$

$\beta = 6$

$\alpha = 6$ $\alpha = 8$ $\alpha = 12$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\alpha = 6$

$\beta = 6$

$\alpha = 6$ $\alpha = 8$ $\alpha = 12$ $\alpha = 3$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\alpha = 6$

$\beta = 6$

$\alpha = 6$ $\alpha = 8$ $\alpha = 12$ $\alpha = 4$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\alpha = 6$

$\beta = 6$ $\beta = 4$

$\alpha = 6$ $\alpha = 8$ $\alpha = 12$ $\alpha = 4$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\alpha = 6$

$\beta = 6$ $\beta = 4$

$\alpha = 6$ $\alpha = 8$ $\alpha = 12$ $\alpha = 4$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\alpha = 6$

$\beta = 6$ $\beta = 4$

$\alpha = 6$ $\alpha = 8$ $\alpha = 12$ $\alpha = 4$ $\alpha = 2$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\alpha = 6$

$\beta = 6$ $\beta = 4$

$\alpha = 6$ $\alpha = 8$ $\alpha = 12$ $\alpha = 4$ $\alpha = 8$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\alpha = 6$

$\beta = 6$ $\beta = 4$ $\beta = 8$

$\alpha = 6$ $\alpha = 8$ $\alpha = 12$ $\alpha = 4$ $\alpha = 8$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\alpha = 6$

$\beta = 6$ | $\beta = 4$ | $\beta = 8$

$\alpha = 6$ | $\alpha = 8$ | $\alpha = 12$ | $\alpha = 4$ | $\alpha = 8$ | $\alpha = 5$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\alpha = 6$

$\beta = 6$ $\beta = 4$ $\beta = 5$

$\alpha = 6$ $\alpha = 8$ $\alpha = 12$ $\alpha = 4$ $\alpha = 8$ $\alpha = 5$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice

$\alpha = 6$

$\beta = 6$ $\beta = 4$ $\beta = 5$

$\alpha = 6$ $\alpha = 8$ $\alpha = 12$ $\alpha = 4$ $\alpha = 8$ $\alpha = 5$

3 6 8 3 3 12 3 4 6 8 10 12 2 8 5 4 7 9

Exercice 2

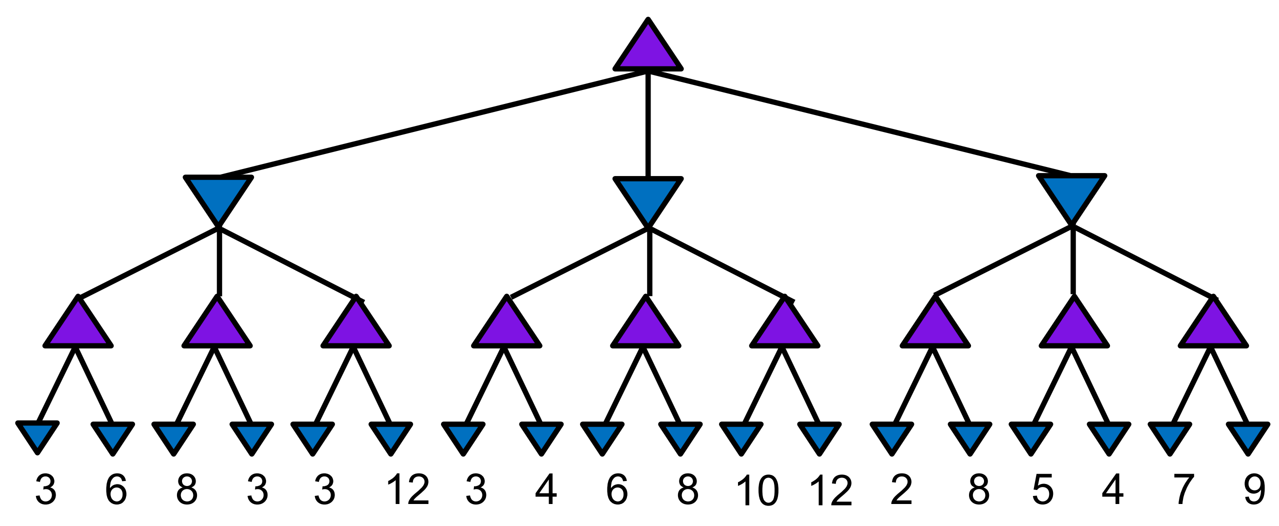
MAX

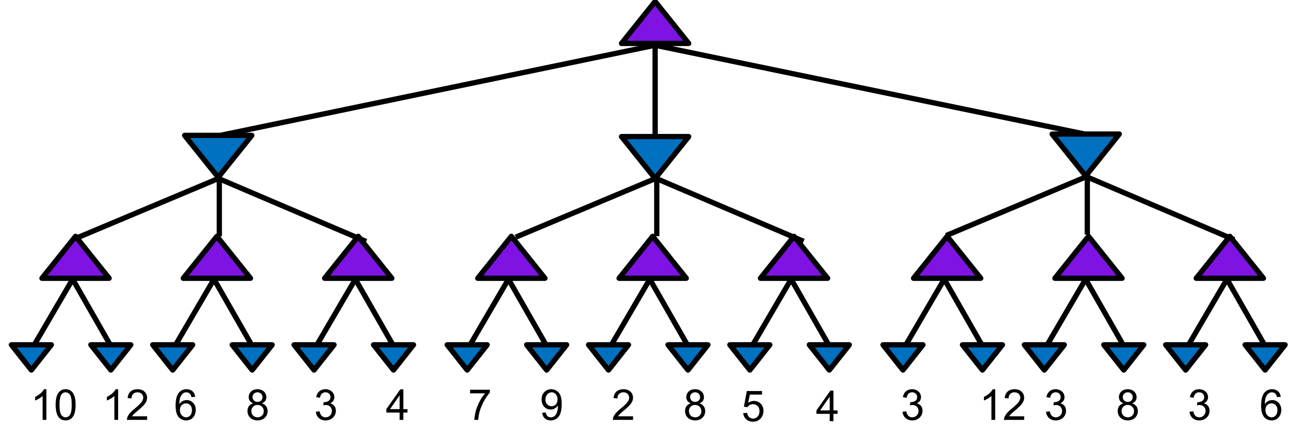
MIN

MAX

10 12 6 8 3 4 7 9 2 8 5 4 3 12 3 8 3 6

Comparaison des nombres des 2 exercices





Minimax – Élagage Alpha-Beta

MinimaxAlphaBeta(positionActuelle,joueur,α,β)

1. **si** positionActuelle est finale
2. **retourner** f(p)
3. **si** joueur == Max
4. $\alpha_\tau = -\infty$
5. **pour** tous les successeurs p_i de PositionActuelle
6. score = MinimaxAlphaBeta(p_i,Min,MAX(α, α_τ),β)
7. α_τ= MAX(α_τ,score)
8. **si** $\alpha_\tau >= \beta$
9. **retourner** α_τ
10. **retourner** α_τ
11. **si** joueur == Min
12. *Même principe*

Minimax – Élagage Alpha-Beta

Propriétés de l'élagage alpha-beta :

- L'élagage ne modifie pas le résultat final de l'algorithme.
- L'efficacité de l'élagage dépend fortement de l'ordre d'exploration des nœuds.
- Amélioration de la complexité:
  - Si l'exploration est effectuée dans un ordre optimal, ça permet de doubler la profondeur de recherche : $O(b^{h/2})$
  - Dans le pire cas, la complexité est la même que pour minimax $O(b^h)$.

Table de transposition

TABLE DE HACHAGE

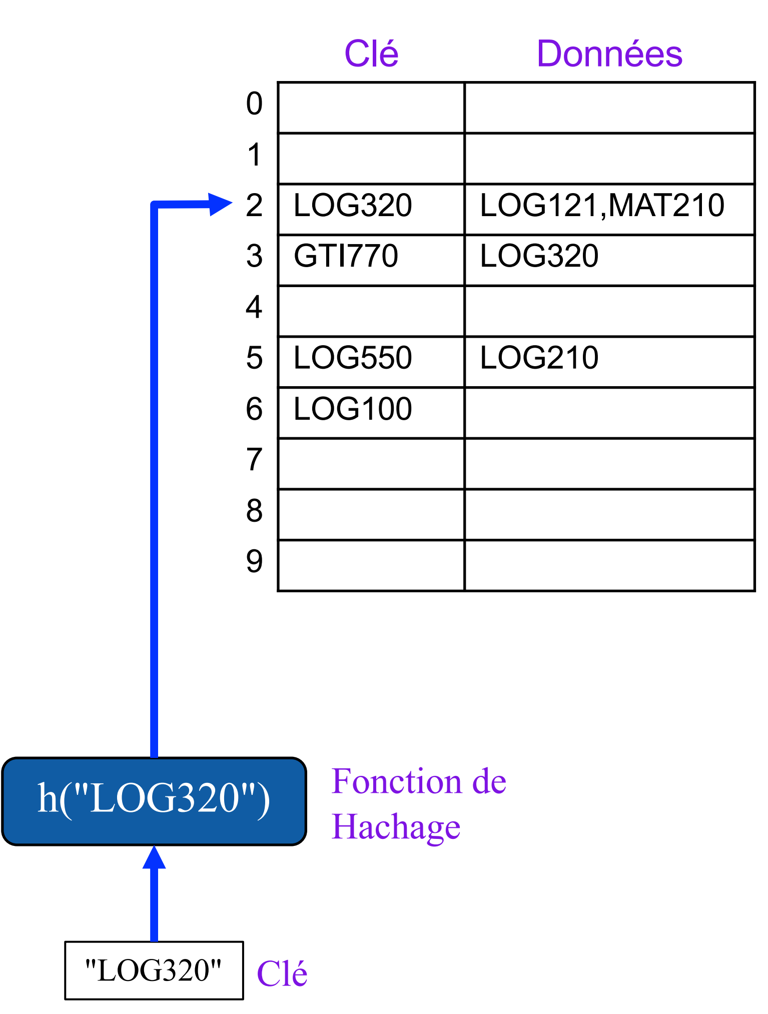
Table de hachage (HashTable)

Utilité de cette structure de données

- Permet d'insérer, supprimer et chercher des éléments en O(1)
- Contraintes:
  - Ordre des éléments n'est pas important
  - Performance dépend de la fonction de hachage

Ce qu'il faut décider lors de la conception:

- Déterminer la fonction de hachage
- Déterminer la taille de la table
- Déterminer stratégie de résolution des collisions



Opérations

Insertion

T[h("LOG320")] = < "LOG320", "LOG121,MAT210">

Fonction de hachage — Clé — Données

Suppression

T[h("LOG320")] = NULL

Recherche

T[h("LOG320")] retourne les données associées à "LOG320"

	Clé	Données
0		
1		
2	LOG320	LOG121,MAT210
3	GTI770	LOG320
4		
5	LOG550	LOG210
6	LOG100	
7		
8		
9		

Une bonne conception permet un temps moyen de O(1) pour l’insertion et la recherche.

Dimension de la table

La dimension de la table est le nombre d'emplacements (« *buckets* ») disponibles

La dimension dépend de la fonction de hachage et de la stratégie de résolution des conflits

Une bonne règle du pouce pourrait être:

- Pour une bonne performance, il ne faut pas que le nombre d'éléments dépasse 75% de la capacité totale de la table
- Donc, la taille du tableau devrait être d'environ 1.3 fois le nombre maximum de clés qu'elle devra contenir
- Dimension du tableau devrait être un nombre premier

Par exemple, pour N éléments, il faut trouver le plus petit nombre premier plus grand que N*1.3

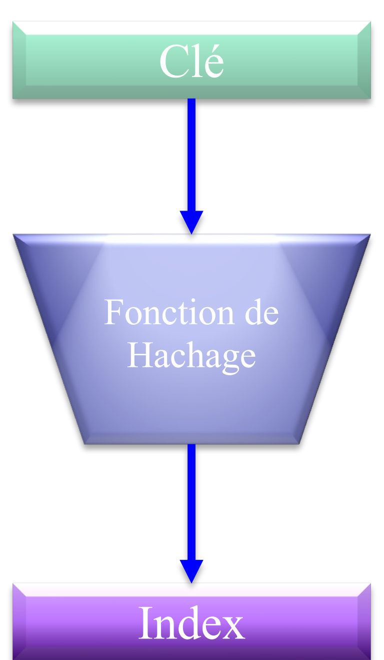
Que se passe-t-il si l'estimation est mauvaise:

- Table trop petite, il faudra éventuellement en créer une plus grande et « rehasher » tous les éléments
- Table trop grande, on gaspille potentiellement de l'espace

Fonction de hachage

La fonction de hachage fait l'association entre la clé et un index dans la table

- h(clé) = index dans la table de hachage



Choix d'une fonction de hachage:

- Doit-être facile et rapide à calculer
- Devrait distribuer les clés de façon la plus uniforme possible
  - Réduit les risques de collision
- Ne doit pas être aléatoire

Fonction de hachage
Exemple: chaîne de caractères

Troisième approche:

Multiplie chaque caractère par un nombre premier plus grand que le nombre de caractères

```
   Hash(str,taille)
1.   h = 0
2.   pour chaque c dans str
3.      h = 31*h + c;
4.   retourner h % taille
```

$$h(S) = \left[\sum_{i=0}^{L-1} S[L-i-1] \cdot 31^i\right] \% taille$$

Il s'agit ici de la fonction hashCode() de la classe String en Java.

Problèmes:

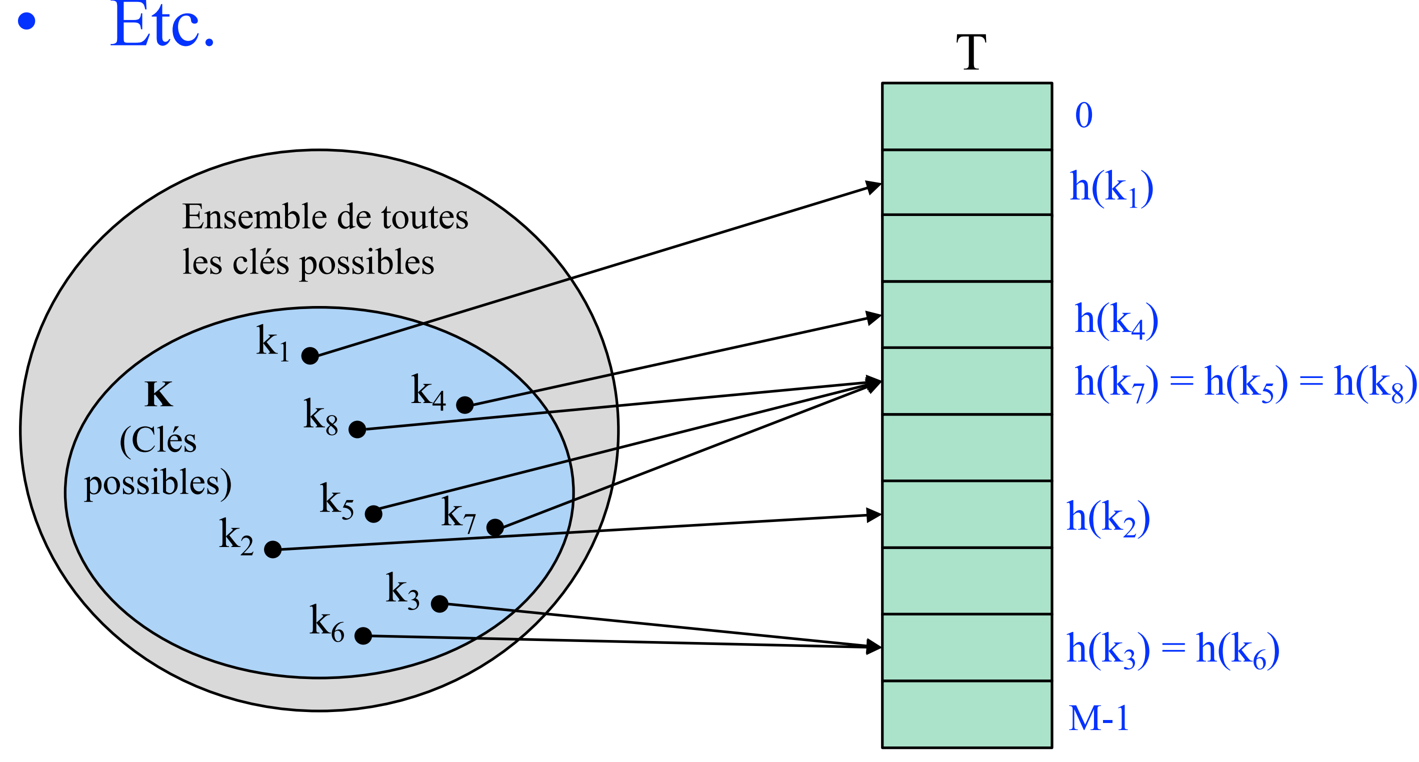
- N'est pas la meilleure en terme de bonne distribution des clés
  - Très simple
  - Raisonnablement rapide pour les clés de longueur modérée
- Peut devenir extrêmement lente pour les grandes clés
  - Il est possible de réduire le temps de calcul en limitant le nombre de caractères utilisés
- Forte possibilité que h devienne trop grand pour une variable
  - Il faut tenir compte de cette possibilité dans l'implémentation

[weiss]

Gestion des collisions

Techniques pour la gestion des collisions

- Par listes chaînées
- Adressage ouvert
- Double hachage
- Etc.



[CLRS]

Résolution par liste chaînée

Chaque élément du tableau est la tête d’une liste chaînée

- Lors d’une collision, les éléments sont ajoutés à la liste

T

Ensemble de toutes les clés possibles

K (Clés possibles)

k_1 k_4 k_8 k_5 k_7 k_2 k_3 k_6

k_1

k_4

k_5 k_7 k_8

k_2

k_3 k_6

[CLRS]

Résolution par liste chaînée

Avantages

- Traitement simple des collisions
- Traitement simple des suppressions
    - Il suffit d’éliminer l’élément de la liste
- Le nombre de données peut dépasser la grandeur de la table.

Inconvénients:

- La liste peut devenir très longue
    - De trop longues listes réduisent l’efficacité de la table de hachage
- Plus de mémoire à cause des pointeurs
    - Particulièrement important lorsque la taille des données est petite
- Le pire cas absolu:
    - Tous les éléments dans une seule liste
    - Causé par une mauvaise fonction de hachage

Résolution par adressage ouvert

Principe:

- Tous les éléments sont stockés directement dans la table
- Lorsqu'il y a une collision, il faut chercher une autre case disponible

Fonction de hachage:

- Fonction de hachage devient:

$$h_i(x) = [h(x) + f(i)] mod\ M$$

La distance parcourue dans la table après la $i^{ième}$ collision, le point départ étant l'index h(x) (i=0).

HashMap en Java 11

Dimension de la table

- Puissance de 2
- Valeur par défaut: 16
- Facteur de charge maximal : 0.75

Résolution des conflits

- Liste chainée
  - Liste chainée simple implémentée dans le nœud
  - N'utilise pas la classe LinkedList<>
- Devient un arbre rouge-noir si trop d'éléments dans le « bucket »
  - Seuil par défaut : 8
  - Seuil pour revenir à une liste chainée: 6

HashMap en Java 11(suite)

Calcul de l'index

- Considérant une table de dimension 2^i
  - Utilise les i bits les moins significatifs du hash
  - index = (n - 1) & hash
- Cette méthode peut causer beaucoup de collisions
  - Redistribue les bits significatifs sur les bits moins significatifs
  - ```
    static final int hash(Object key) {
        int h;
        return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);
    }
    ```

Redimensionnement de la table

- Seuil par défaut : 0.75
- Double la dimension de la table
    ```

Répétitions dans l'arbre de jeu

Il y a plusieurs combinaisons de coups qui vont mener au même plateau.

Hachage de Zobrist

Fonctionnement:

- Génère un nombre aléatoire pour chacune des cases du plateau et pour pour chacune des pièces
  - Pour le tic tac toe, il y a 9 cases et deux pièces possibles, il faut donc 18 nombres aléatoires
- Calcule le hash du plateau

Utilise la fonction ou-exclusif

- Cette fonction à la propriété d'être réversible

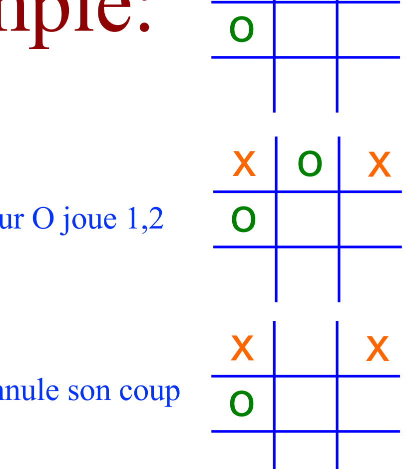
$$(A \otimes B) \otimes B = A$$

Hachage de Zobrist

Génération des clés

case	Clé pour X	Clé pour O
1,1	44532	72217
1,2	90195	10291
1,3	81410	65932
2,1	36721	91854
…	…	…

Exemple:



Case 1,1 Case 1,3 Case 2,1

$$H(plateau) = 44532 \otimes 81410 \otimes 91854$$
$$= 33408$$

Joueur O joue 1,2

$$H(plateau) = 33408 \otimes 10291$$
$$= 43699$$

Joueur O annule son coup

$$H(plateau) = 43699 \otimes 10291$$
$$= 33408$$

Minimax et alpha-beta (conclusion)

Références:

- http://turing.cs.pub.ro/auf2/html/chapters/chapter3/chapter_3_4_2.html
- G. F. Luger et W.A. Stubblefield, Artificial Intelligence and the Design of Expert Systems,The Benjamin/Cummings Publishing Compagny