Chapitre 4

# Théorie des jeux

Dans ce chapitre nous allons nous intéresser à l'étude théorique et algorithmique de jeux à deux joueurs antagonistes jouant alternativement. Les jeux qui nous intéressent sont à information totale : à tout instant d'une partie chacun des joueurs a une vision complète de l'état du jeu. Ceci exclus la plus-part des jeux de cartes (on ne connaît pas le jeu de l'adversaire) mais inclus des jeux tels les échecs, les dames, le go, etc.
Dans un premier temps, nous allons nous intéresser à des jeux « simples » pour lesquels il est possible de déterminer (au moins pour de petites configurations) une stratégie gagnante, puis nous verrons, pour les jeux les plus complexes, comment bâtir une stratégie à l'aide d'une heuristique

## I. Jeu d'accessibilité sur un graphe

**Exemple fil rouge : un jeu de Nim** :

- C'est un jeu à 2 joueurs qui commence avec 20 bâtonnets posés sur la table ;
- à tour de rôle, chaque joueur peut enlever 1, 2, ou 3 bâtonnets ;
- le perdant est celui qui enlève le dernier bâtonnet. Un exemple de partie est donné ci-dessus, avec les mouvements d'Alice en rouge, et ceux de Bob en bleu.

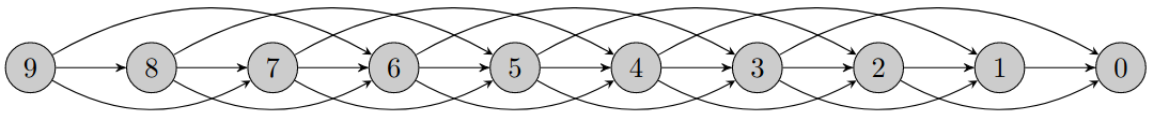
$$20 \longrightarrow 17 \longrightarrow 16 \longrightarrow 15 \longrightarrow 12 \longrightarrow 10 \longrightarrow 8 \longrightarrow 7 \longrightarrow 4 \longrightarrow 3 \longrightarrow 1 \longrightarrow 0$$

### 1. Modélisation par graphe

À un tel jeu, on peut associer un graphe orienté $G = (V, E)$ :

- les sommets V de G sont les positions atteignables dans le jeu. Pour le jeu précédent, on pourrait utiliser 21 sommets numérotés de 0 à 20, indiquant le nombre de bâtonnets restants.
- les arcs E de G indiquent quel sommet est atteignable depuis un autre en jouant un unique coup.

Voici ci-dessous le graphe pour une partie à 9 batonnets.




**Définition 4.1.**

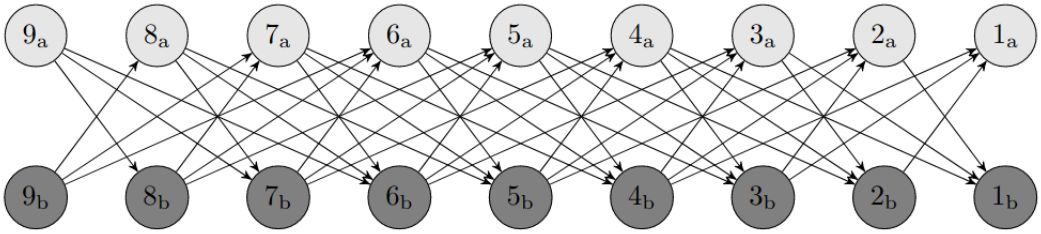
Une **partie** sur un tel graphe est un chemin où le premier joueur, depuis le sommet de départ, suit un arc, et ensuite chaque joueur suit à tour de rôle un arc, si c'est possible. Une partie est **finie** si ce chemin termine sur un sommet sans successeur, **infinie** sinon.

**Proposition 4.1.**

Si le graphe est sans circuit, toute partie est finie

## Graphe biparti

Une fois fixé le joueur qui commence (Alice par exemple), on peut, en doublant chacun des sommets qu'on indexera par le nom du joueur, créer un nouveau graphe dont les sommets seront partitionnés en deux sous-ensembles $S = S_a \cup S_b$ avec $S_a \cap S_b = \emptyset$ où $S_i$ est l'ensemble des positions à partir desquelles le joueur $i$ jouera. Un tel graphe est dit **biparti**. En voici une illustration pour 9 batonnets :




**Exercice 4.1**

Dessiner le graphe biparti pour le jeu à 5 batonnets

### 2. Construction du graphe

Nous allons mettre en place une implémentation du graphe sous la forme d'un dictionnaire dont les clés sont les sommets et la valeur associée à un sommet est la liste de successeurs de ce sommet. Ainsi, $G[s] = [a, b]$ signifie que $s$ est un sommet et qu'il y a un arc reliant $s$ à $a$ et un autre reliant $s$ à $b$ Procédons par récursivité : en partant d'un sommet de la forme $(n, j)$ on va aller visiter les trois successeurs de ce sommet : $(n-1, o)$,$(n-2, o)$,$(n-3, o)$ où $o$ est le nom de l'opposant du joueur, mais attention, il faut garder en tête que le nombre d'allumettes doit être supérieur ou égal à 1.

**Exercice 4.2**

Compléter la fonction ci-dessous permettant de construire le graphe `G` du jeu d'argument `N` le nombre de batonnets et `joueur` précisant le joueur commençant la partie.

```
Opp = {"Alice" : "Bob", "Bob" : "Alice"}
G = {}
def GrapheNim(N, joueur):
''' Renvoie le graphe du jeu de Nim partant de N allumettes avec joueur qui commence'''
...
```

### 3. Modélisation d'une partie

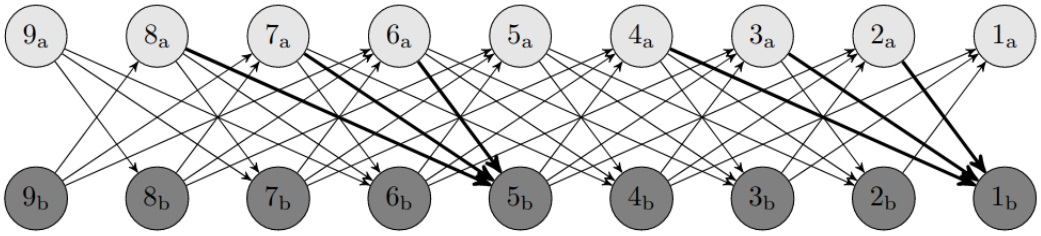
De manière générale, on peut imaginer un jeu se jouant sur un graphe orienté non valué appelé arène avec deux joueurs J1 et J2. Chaque sommet désigne une configuration du jeu. L'un de ces sommets est la position de départ.
Une arête qui relie $s_1$ à $s_2$ signifie que l'un des joueurs à la possibilité de passer de $s_1$ à $s_2$.
Le **graphe** possède une propriété d'être **biparti** : l'ensemble des sommets S vérifie $S = S_{J_1} \cup S_{J_2}$ avec $S_{J_1} \cap S_{J_2} = \emptyset$ où $S_{J_1}$ (resp $S_{J_2}$) désigne l'ensemble des sommets à partir duquel $J_1$ (resp$(J_2)$ joue.
Si $s \in S_{J_1}$ , on dit que c'est un sommet contrôlé par $J_1$. De plus, les arcs ne peuvent relier que l'un des sommets de $S_{j_1}$ vers un sommet de $S_{J_2}$ et inversement 4.
Un sommet sans successeur est appelé **sommet terminal** (le jeu s'arrête donc), un sommet sans prédécesseur est appelé **sommet initial**.
On suppose le graphe fini et acyclique (il n'y a pas de cycle), ainsi partant d'un sommet initial $s_0$, on

est obligé de finir sur un sommet n'ayant pas de successeur et donc le jeu s'arrête.
Notons T la liste des sommets terminaux, alors $\mathtt{T} = \mathtt{G1} \cup \mathtt{G2} \cup \mathtt{N}$.
Les sommets de G1 sont synonymes de victoires pour $J_1$, les sommets de G2 sont synonymes de victoires pour $J_2$ et les sommets de N de parties nulles.
Ainsi, une partie où $J_1$ commence est un chemin de la forme

$$s_0 \underset{J_1}{\longrightarrow} s_1 \underset{J_2}{\longrightarrow} s_2 \underset{J_1}{\longrightarrow} s_3 \longrightarrow \cdots \longrightarrow s_n$$

Où $s_i s_{i+1}$ est un arc du graphe. Les sommets $s_0, s_2, s_4$ etc. sont contrôlés par J1. Cela veut dire, qu'au début de la partie, c'est $J_1$ qui a choisi d'aller en $s_1$ (parmi tous les successeurs de $s_0$), d'aller en $s_3$ (parmi les successeurs de $s_2$). En revanche, les sommets $s_1$, $s_3$, $s_5$ etc. sont contrôlés par $J_2$. Cela veut dire que quand on était en $s_1$ c'est $J_2$ qui a choisi d'aller en $s_2$ (parmi tous les successeurs de $s_1$) etc.
Le sommet $s_n$ est un sommet terminal et donc la partie s'arrête, suivant que $s_n$ appartiennent à G1, à G2, ou à N, on a le résultat de la partie.
Le graphe est connu des joueurs, il n'y a pas d'informations cachées, pas de hasard. Les jeux dont le graphe est bipartie, et dont les parties s'arrêtent sur trois états (victoires de $J_1$, victoire de $J_2$ ou nul) s'appellent des **jeux d'accessibilité**.
Une **stratégie** du joueur $J_1$ est une fonction $\varphi : \begin{cases} S_{J_1} & \longrightarrow S_{J_2} \\ s_1 & \longmapsto s_2 \in G[s_1] \end{cases}$.
Cela veut dire que si le joueur $J_1$ doit jouer alors qu'il est en position $s_1$, alors il jouera la position $s_2$ (un des successeurs de $s_1$). Conformément au programme, c'est une stratégie sans mémoire : le joueur joue seulement en fonction de la position actuelle $s_1$ et non en fonction de l'historique de la partie.
Une **stratégie** $\varphi$ est dite **gagnante** pour $J_1$ depuis le sommet $s_0$ si toute partie jouée depuis $s_0$ en suivant $\varphi$ est gagnante.
Une **position** est dite **gagnante** pour $J_1$ s'il existe une stratégie gagnante partant de cette position.

**Exemple 4.3**
Dans le jeu de Nim à 9 bâtonnets, en gras, les arcs d'une stratégie gagnante pour le premier joueur.
Les positions $9_a$ et $5_a$ étant perdantes, la définition de la stratégie sur ces sommets n'a pas d'importance.




## 4. Calcul des attracteurs

Dans un jeu d'accessibilité à deux joueurs, résoudre le jeu consiste à calculer les positions gagnantes de chacun des joueurs, et une stratégie pour chacun (définie sur chaque position gagnante que le joueur contrôle).
On se concentre d'abord sur les positions gagnantes du premier joueur : Alice, qui doit pour gagner atteindre un sommet de G1.
L'outil adéquat est **l'attracteur**.

**Définition 4.2. Attracteurs**

Soit G = (S,A) un graphe, deux ensembles $S_A$, $S_B$ formant une partition de S et $V_a$ l'ensemble des sommets définissant une victoire pour Alice.
On définit incrémentalement une suite de sous-ensembles de sommets par : ...

- $\mathcal{A}_0 = \mathrm{V}_a$
- $\forall j \geqslant 0,\ \mathcal{A}_{j+1} = \mathcal{A}_j \cup \{s \in \mathrm{S_A} \mid \exists s' \in \mathcal{A}_j,\ (s,s') \in \mathrm{A}\}$
  $\cup \{s \in \mathrm{S_B} \mid \forall s' \in \mathrm{S},\ (s,s') \in \mathrm{A} \Rightarrow\ s' \in \mathcal{A}_j\}$

Autrement dit, $\mathrm{A}_{j+1}$ est constitué :

- des sommets de $\mathrm{A}_j$ ;
- des sommets contrôlés par Alice pour lesquels il existe au moins un arc permettant de rejoindre un sommet de $\mathrm{A}_j$ ;
- des sommets contrôlés par Bob pour lesquels tous les arcs aboutissent sur des sommets de $\mathrm{A}_j$ .

L' **Attracteur** de $\mathrm{G}_1$ est l'ensemble $\mathcal{A} = \bigcup_{j=0}^{+\infty} \mathcal{A}_j$

**Théorème 4.2.**

Avec les notations précédentes, Alice possède une stratégie gagnante pour tout sommet de $\mathcal{A}$ (et uniquement ceux-là).

**Exercice 4.4**

Dans le jeu de Nim à 5 bâtonnets, déterminer l'attracteur associé aux positions gagnantes d'Alice.

**Remarque**

S'il est possible de faire match nul dans le jeu considéré, il faut calculer indépendamment les attracteurs pour Alice, puis les attracteurs pour Bob (les sommets restants étant les positions de match nul).
En revanche, s'il n'est pas possible d'avoir match nul, les attracteurs pour Bob sont le complémentaire des attracteurs pour Alice.

**Calcul de l'attracteur**

Pour calculer les attracteurs, il suffit essentiellement de faire un parcours de graphe, mais sur le graphe transposé de G : en effet, on part de l'ensemble $\mathrm{V}_a$, et on remonte les arcs à l'envers.
Il est utile de calculer les degrés sortants des sommets dans G (qui sont les degrés entrants dans ${}^t\mathrm{G}$), pour gérer la condition : « tout arc sortant d'un sommet de $\mathrm{S}_b$ aboutit à un sommet attracteur ».

**Exercice 4.5**

Ecrire une fonction `transpose(G)` de paramètre un graphe `G` et renvoyant son graphe transposé.

**Exercice 4.6**

Ecrire une fonction `deg_sortant(G)` de paramètre un graphe `G` et renvoyant un dictionnaire dont les clés sont les sommets et les valeurs leur degré sortant.

**Algorithme 1 :** Calcul des attracteurs

**Données :** Un graphe $G = (S, A)$ donné par listes d'adjacence, une partition $S = S_a \uplus S_b$, un ensemble $V_a$
**Résultat :** Les attracteurs associés à $V_a$

```
A ← ∅ ;
pour s ∈ S faire
    n_s ← le degré sortant de s dans G ;
Calculer ᵗG, le graphe transposé de G ;
fonction parcours(s) :
    si s ∉ A alors
        A ← A ∪ {s} ;
        pour s' voisin de s dans ᵗG faire
            n_s' ← n_s' − 1 ;
            si s' ∈ S_a ou n_s' = 0 alors
                parcours(s') ;
pour s ∈ V_a faire
    parcours(s) ;
retourner A
```

**Exercice 4.7**
Ecrire une fonction renvoyant l'attracteur associé à un ensemble de sommets définissant la victoire.

## 5. Construction d'une stratégie gagnante

- Pour calculer en parallèle une stratégie gagnante pour Alice, il suffit de de modifier légèrement l'algorithme : lorsqu'on lance `parcours(s')` dans `parcours(s)`, on peut poser $\varphi(s') = s$.

- Pour calculer une stratégie « non perdante » pour Bob sur un sommet $s \in S_b$ n'appartenant pas aux attracteurs pour $V_a$, il suffit de parcourir la liste d'adjacence de $s$ à la recherche d'un sommet $s'$ qui n'est pas non plus dans $\mathcal{A}$.

# II. Algorithme min-max

Dans le cas d'un jeu à deux joueurs plus complexe, le calcul de l'attracteur n'est pas possible. L'algorithme que nous avons écrit a une complexité en $O(n^3)$, où $n$ est le nombre de sommets du graphe, autrement dit le nombre de positions que l'on peut rencontrer lors d'une partie. Cependant, cet entier n est souvent extrêmement grand : il est estimé de l'ordre de 1032 pour les dames, entre 1043 et 1050 pour le jeu d'échecs, de l'ordre de 10100 pour le jeu de go. Il devient donc nécessaire de s'appuyer non plus sur une évaluation exacte de la position, mais sur une estimation de la valeur de la position atteinte.

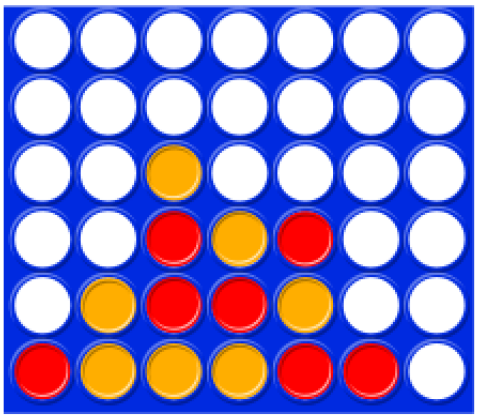
## 1. Heuristique

Dans la suite de cette section, nous supposerons posséder une fonction $h$ qui à toute position légale $p$ du jeu associe une valeur dans $\mathbb{R}$, de sorte que :

- plus $h(p)$ est grand, meilleure est la position pour Adam ;
- plus $h(p)$ est petit, meilleure est la position pour Ève.

Une telle fonction est appelée une heuristique.

**Exemple 4.8 (Jeu du puissance 4)**
Pour illustrer cette section, nous allons prendre l'exemple du Puissance 4 : le but du jeu est d'aligner une suite de quatre pions de même couleur sur une grille comptant six rangées et sept colonnes. Tour à tour, les deux joueurs placent un pion dans la colonne de leur choix, le pion coulisse alors jusqu'à la position la plus basse possible dans la dite colonne à la suite de quoi c'est à l'adversaire de jouer. Le vainqueur est le joueur qui réalise le premier un alignement (horizontal, vertical ou diagonal) consécutif d'au moins quatre pions de sa couleur. Si, alors que toutes les cases de la grille de jeu sont remplies, aucun des deux joueurs n'a réalisé un tel alignement, la partie est déclarée nulle.



Une heuristique simple consiste à attribuer à chaque case une valeur, par exemple le nombre d'alignements potentiels de quatre pions lorsqu'on place un pion à cet emplacement, puis à sommer les cases occupées (positivement pour les pions d'Adam, négativement pour ceux d'Ève).

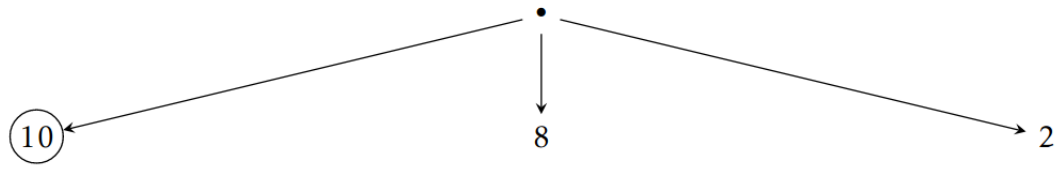
| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | 4 | 5 | 7 | 5 | 4 | 3 |
| 4 | 6 | 8 | 10 | 8 | 6 | 4 |
| 5 | 8 | 11 | 13 | 11 | 8 | 5 |
| 5 | 8 | 11 | 13 | 11 | 8 | 5 |
| 4 | 6 | 8 | 10 | 8 | 6 | 4 |
| 3 | 4 | 5 | 7 | 5 | 4 | 3 |

Par exemple, si on convient que les pions jaunes sont ceux d'Adam, la valeur de l'heuristique de la position présentée sur le plateau du jeu de puissance 4 représenté plus haut est égale à :
$4+5+7+6+8+13+11-3-5-4-8-10-11-11=2$.
Évidemment, l'heuristique sera égale à $+\infty$ pour une position gagnante pour Adam, et à $-\infty$ pour une position gagnante pour Ève.

## 2. Min-Max

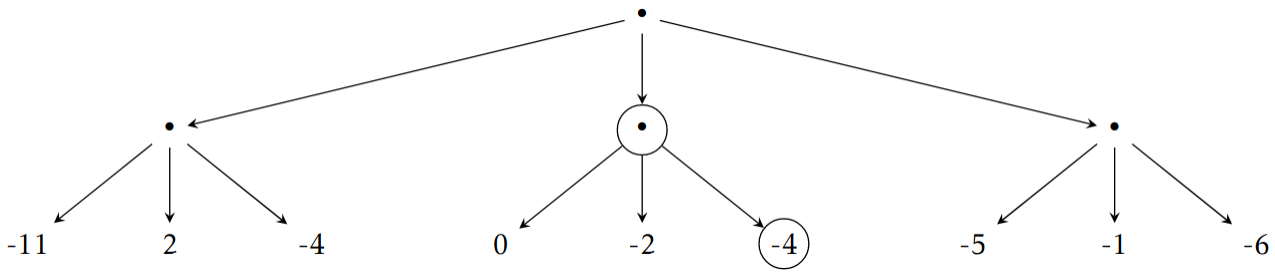
Au moment où l'un des deux protagonistes doit jouer, plusieurs possibilités s'offrent à lui (entre une et sept pour le puissance 4). Une solution simple pour choisir le coup à jouer consiste à calculer l'heuristique correspondant à chacune des configurations atteignables et à jouer celle d'heuristique maximale (pour Adam) ou minimale (pour Ève).

**Exemple 4.9**
Imaginons que c'est à Adam de jouer et que seules 3 possibilités lui sont offertes :

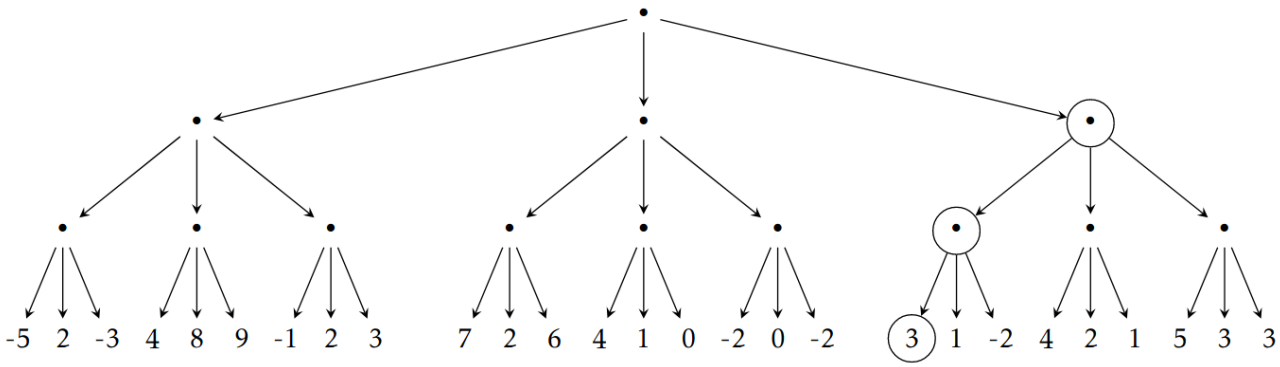



Adam choisira donc le coup le plus à gauche, correspondant à une évaluation égale à 10.

Mais Adam peut aussi tenir compte du coup que va jouer Ève ensuite, et donc calculer l'heuristique de chacune des positions qu'Ève pourra atteindre. Si on observe la figure ci-dessous, on constate qu'il vaut mieux pour Adam jouer le coup central, en partant du principe qu'Ève joue au mieux son coup.



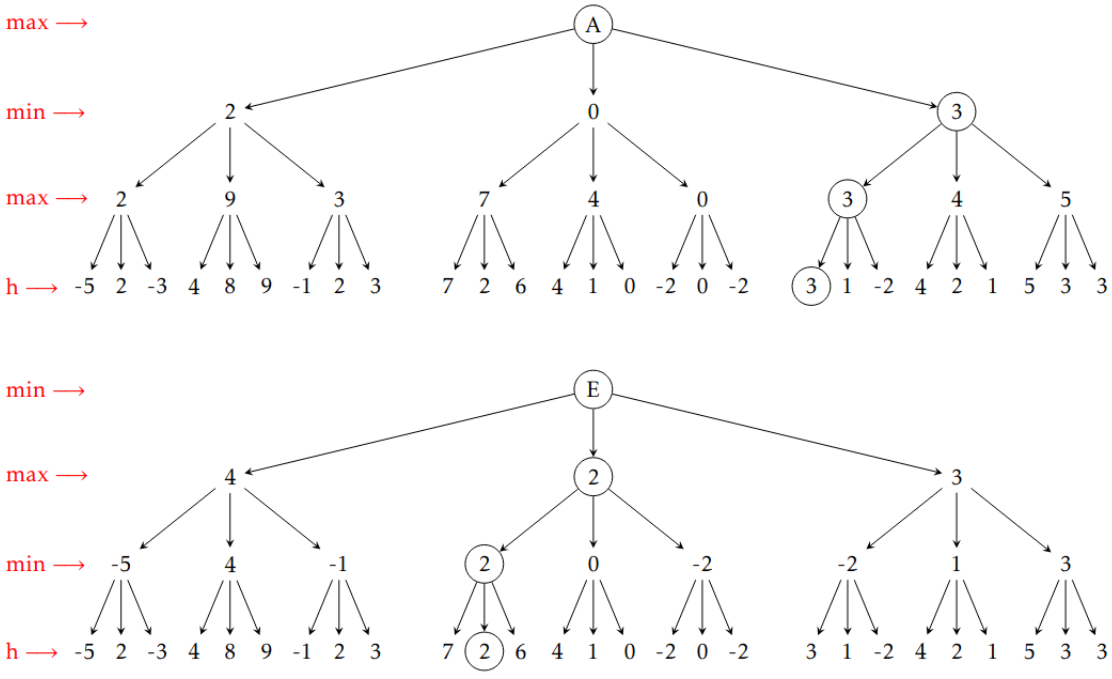
Bien évidemment, on peut réitérer ce raisonnement et tenir compte du coup suivant, joué cette fois par Adam. La figure suivante montre qu'en tenant compte des deux coups suivant, Adam a en fait intérêt à jouer le coup le plus à droite.




On peut répéter ce raisonnement, mais le nombre de configurations à examiner ayant tendance à croitre exponentiellement, il est nécessaire de limiter la profondeur de la recherche.

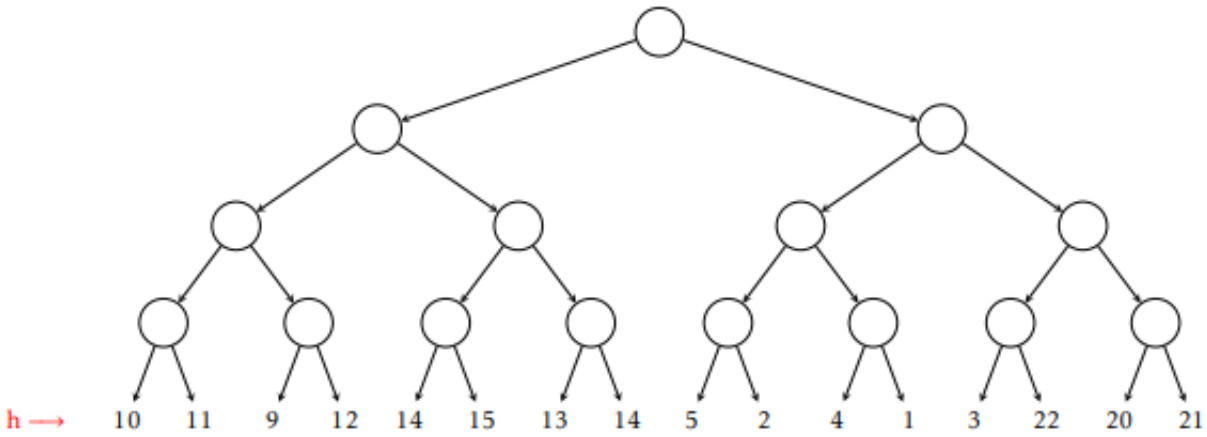
## 3. L'algorithme

Pour calculer le meilleur coup d'Adam, il faut donc commencer par calculer la valeur de l'heuristique de toutes les positions atteignables en $n$ coups (les feuilles de l'arbre). Si $n$ est pair, Ève aura joué le dernier coup : le père de chacune de ces feuilles se verra donc attribuer le minimum des valeurs de ses fils. À l'inverse, si $n$ est impair le père de chacune de ces feuilles se verra attribuer la valeur maximale de ses fils (car Adam aura joué en dernier). Ainsi, de proche en proche chaque position de l'arbre se verra attribuer une valeur.
Voici ci-dessous le résultat de l'algorithme min-max à une profondeur 2 (le premier arbre si c'est à Adam de jouer, le second si c'est à Ève).



**Exercice 4.10**

Calculer la valeur de la position associée à l'arbre ci-dessous, dans le cas où c'est le joueur qui cherche à maximiser l'heuristique qui doit jouer.




Nous allons écrire deux fonctions :

- `maximin(p, n)` (destinée à Adam) va chercher à maximiser l'heuristique après `n` coups en partant de la position `p`, en supposant que son adversaire joue au mieux ;
- `minimax(p, n)` (destinée à Ève) va chercher à minimiser l'heuristique après `n` coups en partant de la position `p`, en supposant que son adversaire joue au mieux.

Ces deux fonctions sont mutuellement récursives : pour calculer `maximin(p, n)` on calcule pour chaque position `p1`,..., `pk` atteignable à partir de `p` la valeur de l'heuristique des positions `minimax(pi, n-1)` avant de choisir la position conduisant à la valeur maximale.
De manière symétrique, pour calculer `minimax(p, n)` on calcule pour chaque position `p1`,..., `pk` atteignable à partir de `p` la valeur de l'heuristique des positions `maximin(pi, n-1)` avant de choisir la position conduisant à la valeur minimale.
Pour la rédaction de l'algorithme, on suppose définies la fonction `h(p)` qui prend pour argument une position du jeu et renvoie la valeur de son heuristique, ainsi que la fonction `successeurs(p)` qui renvoie la liste des positions atteignables à partir de la position `p`.

```
def minimax(p, n):
    if n == 0 or successeurs(p) == []:
        return h(p)
    mini = np.inf
    for pk in successeurs(p):
        s = maximin(pk, n - 1)
        if s < mini:
            mini = s
    return mini
```

```
    def maximin(p, n):
        if n == 0 or successeurs(p) == []:
            return h(p)
        maxi = - np.inf
        for pk in successeurs(p):
            s = minimax(pk, n - 1)
            if s > maxi:
                maxi = s
            return maxi
```