IV Jeux d'accessibilité à deux joueurs

17 novembre 2025

1 Généralités

Dans ce chapitre, nous allons voir comment modéliser certains jeux à deux joueurs (comme le jeu de nim, le morpion, le puissance 4, le jeu de hex), où chaque joueur joue alternativement. Nous allons voir comment modéliser ces jeux par des graphes, comment déterminer si certaines positions sont gagnantes ou non pour l'un des deux joueurs, et comment déterminer une stratégie pour l'un ou l'autre des joueurs.

Nous nommerons les deux joueurs J_0 et J_1, le joueur J_0 étant le premier joueur à jouer.

Exemple 1.0.1 (jeu de Nim)**.**
Ce jeu est le plus simple qui soit. On installe n bâtonnets devant les deux joueurs, jouant chacun son tour. À chaque tour, le joueur a le droit d'enlever un certain nombre de bâtonnets (classiquement : 1,2 ou 3). Le joueur qui ne peut plus jouer a perdu (l'autre joueur a alors gagné). L'ensemble des nombres de bâtonnets que l'on peut retirer à chaque tour sera appelé *règle du jeu de Nim* (la règle classique est $\{1, 2, 3\}$).

On remarquera qu'il ne peut y avoir de match nul, ni de partie infinie.

Nous choisirons ce jeu comme illustration dans ce chapitre.

Exemple 1.0.2 (morpion)**.**
On joue sur un plateau de $n \times n$ cases (classiquement : $n = 3$). Chaque joueur joue à tour de rôle en choisissant une case non vide. Dès qu'un joueur aligne p cases (classiquement : $p = 3$, les alignements diagonaux étant autorisés), il gagne.

Si le plateau est rempli sans qu'aucun joueur n'ait aligné p cases, il y a match nul. Il ne peut y avoir de partie infinie.

Exemple 1.0.3 (puissance 4)**.**
On joue sur un plateau vertical de 7×7 cases. Chaque joueur jour tour à tour, en choisissant une colonne non remplie et en faisant tomber dedans un jeton. Dès qu'un joueur aligne 4 cases (les alignements diagonaux étant autorisés), il gagne.

Si le plateau est rempli sans qu'aucun joueur n'ait aligné p cases, il y a match nul. Il ne peut y avoir de partie infinie.

On modélise l'ensemble des états du jeu par un graphe orienté (S, A), dont l'ensemble des sommets est partitionné en deux ensembles S_0 et S_1. Les sommets de S_0 représentent les situations où le joueur 0 joue (et donc choisit son coup), les sommets de S_1 représentent les situations où le joueur 1 joue. Nous représenterons les sommets contrôlés par le joueur J_0 par des cercles, et ceux contrôlés par J_1 par des carrés. Certains sommets correspondent à des conditions de victoire pour l'un ou l'autre des joueurs (on les représente dans ce cours par des sommets entourés deux fois), et un sommet correspond au coup de départ (on le représente avec une flèche entrante).

Une arête du graphe représente un coup que l'un ou l'autre des joueurs peut effectuer. Les joueurs jouent à tour de rôle, ce qui signifie qu'il n'existe pas d'arêtes reliant deux sommets de S_0 ou deux sommets de S_1. On remarquera que ce n'est pas une condition très importante : il suffirait d'intercaler un sommet (fictif) contrôlé par J_1 au milieu d'une arête entre deux sommets contrôlés par J_0 pour revenir à notre cadre formel. Cela se formalise comme suit.

Définition 1.0.4 (graphe biparti)**.**
Un graphe (S, A) est biparti s'il existe une partition de S en deux parties S_0 et S_1 telles qu'aucune arête de S ne relie deux sommets de S_0, ou deux sommets de S_1.

Voici un exemple de modélisation du jeu de Nim.

Exemple 1.0.5 (Graphe du jeu de Nim, voir figures 1 et 2)**.**
Pour le jeu de Nim, avec n bâtonnets au départ, on notera (k, i) la situation où le joueur i joue, avec k bâtonnets placés devant lui, et l'on aura donc

$$S_0 = \{\, (k, 0) \mid 0 \leqslant k \leqslant n \,\},$$
$$S_1 = \{\, (k, 1) \mid 0 \leqslant k \leqslant n - 1 \,\}.$$

Si q est un élément de la règle (dont un nombre de bâtonnets que l'on peut retirer), et si $k - q \geqslant 0$, il y aura alors les arêtes suivantes (chaque joueur peut enlever q bâtonnets aux k bâtonnets présents, l'autre joueur se retrouve alors avec $k - q$ bâtonnets) :

$$(k, 0) \to (k - q, 1),$$
$$(k, 1) \to (k - q, 0).$$

Les sommets où J_0 gagne sont les sommets contrôlés par J_1 et dont le numéro est strictement inférieur au minimum des éléments de la règle.

On remarquera qu'il y a plusieurs manières de présenter un même graphe...

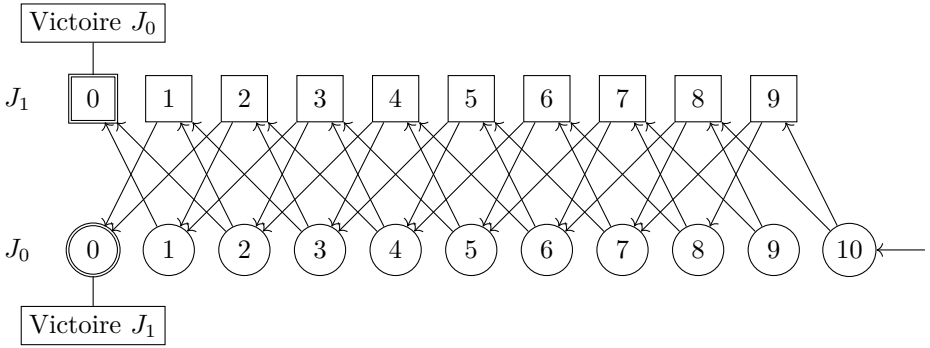


Figure 1 – Graphe du jeu de Nim, règle $= \{1, 2\}$, $n = 10$.

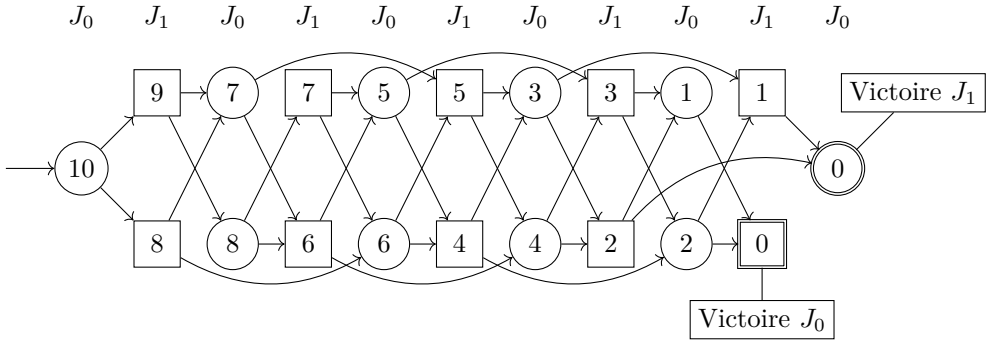


Figure 2 – Graphe du jeu de Nim, règle $= \{1, 2\}$, $n = 10$.

2 Stratégies et positions gagnantes

Dans toute cette partie, on considère un graphe orienté (S, A), dont les sommets sont partitionnés en deux ensembles S_0 et S_1 (sommets contrôlés par les joueurs J_0 et J_1 respectivement). On suppose qu'il existe des parties V_0 et V_1 de S, non vides et disjointes, qui représentent respectivement les ensembles de sommets où J_0 et J_1 gagnent la partie (on parle parfois de *positions de victoire*).

On appellera parfois *position* les sommets de ce graphe. Une position est donc l'état du jeu à un moment donné de la partie.

2.1 Définitions

Définition 2.1.1 (partie)**.**
Une *partie* est un chemin dans le graphe, *i.e.* une suite (s_n) de positions telle que pour tout i, $(s_i, s_{i+1}) \in A$, finie ou infinie.

Définition 2.1.2 (partie gagnante)**.**
Une partie est dite *gagnante pour* J_0 si elle est finie, si sa dernière position appartient à V_0, et si aucune de ses autres positions n'appartient à $V_0 \cup V_1$.

Une partie est dite *gagnante pour* J_1 si elle est finie, si sa dernière position appartient à V_1, et si aucune de ses autres positions n'appartient à $V_0 \cup V_1$.

Remarque 2.1.3.
Un match nul est alors soit une partie finie se terminant en une position (non dans $V_0 \cup V_1$) de laquelle ne part aucune arête, soit une partie infinie dont aucune position n'est dans $V_0 \cup V_1$.

Définition 2.1.4 (stratégie sans mémoire)**.**
Une *stratégie sans mémoire* est la donnée des choix de coups que réalisera un joueur donné pour chaque position qu'il contrôle, en fonction uniquement de la position actuelle.

Formellement, une stratégie sans mémoire pour J_0 est une fonction $s : S_0 \to S$ telle que $\forall x \in S_0$, $(x, s(x)) \in A$.

Une stratégie sans mémoire pour J_1 est une fonction $s : S_1 \to S$ telle que $\forall x \in S_1$, $(x, s(x)) \in A$.

Remarque 2.1.5.
Seules les stratégies sans mémoire sont à votre programme. On pourra donc parler de stratégie, sans risque de confusion.

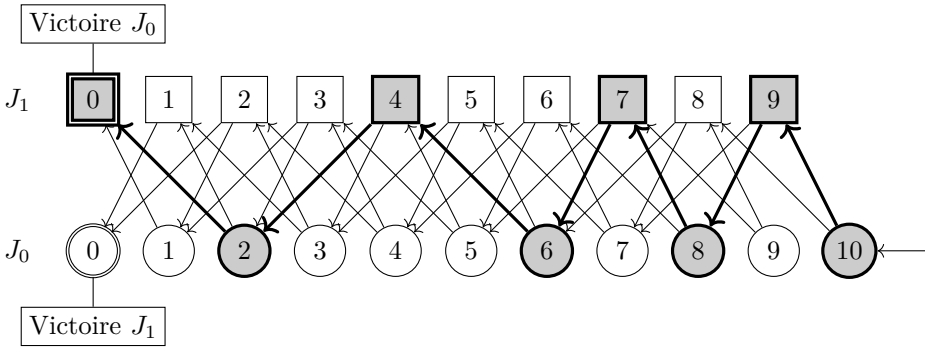


Figure 3 – Un exemple de partie gagnante pour J_0.

Définition 2.1.6 (stratégie gagnante)**.**
Une stratégie s est *gagnante* pour un joueur si toutes les parties où les coups joués par ce joueur se font selon cette stratégie sont gagnantes (pour ce joueur).

Définition 2.1.7 (position gagnante)**.**
Une position est dite *gagnante* pour un joueur s'il existe une stratégie gagnante pour le joueur, lorsque l'on considère que le jeu commence en cette position.

Remarque 2.1.8.
Les positions de V_0 sont bien évidemment gagnantes pour J_0 et celles de V_1 le sont pour J_1. Nous allons voir comment calculer les autres positions gagnantes.

Exemple 2.1.9.
Dans le graphe de la figure 1, la position $(2, 0)$ est gagnante pour J_0. La stratégie qui consiste à jouer en $(0, 1)$ est en effet gagnante pour J_0...

Exercice 2.1.10.
Créer le graphe de ce jeu de Nim en Python et les conditions de victoire ($n = 10$, $R = \{1, 2\}$ pour commencer, puis écrire une fonction prenant en argument n et R).

On utilisera un dictionnaire d'adjacence, avec des valeurs `None` pour créer un ensemble.

Par exemple, voici un dictionnaire représentant l'ensemble $\{(0, 0), (1, 2)\}$:

```
d = {(0,0) :None,
     (1,2) :None}
```

2.2 Calcul des positions gagnantes

L'idée de base est fort simple. Pour une position x contrôlée par J_0 ($x \in V_0$) :

- ▷ s'il existe une arête $(x, y) \in A$ telle que y est une position gagnantes pour J_0, alors x est aussi une position gagnante pour J_0 (une stratégie gagnante de J_0 est de joueur sur y, voir les figures 4b, 4d, 4f et 4h) ;
- ▷ si pour toutes les arêtes $(x, y) \in A$ partant de x, y est une position gagnante pour J_1, alors x est aussi une position gagnante pour J_1 (quelle que soit la stratégie de J_0 en x, J_1 possède une stratégie gagnante pour les coups suivants, voir les figures 4c, 4e, et 4g).

Remarque 2.2.1.
Si x n'a aucun successeur (c'est une position « cul de sac »), alors x n'est pas une position gagnante pour un joueur. Il faut donc exclure ce cas dans l'écriture formelle des ensembles.

On peut donc calculer les positions gagnantes par récurrence, à partir des positions de victoire V_0 (pour J_0) et de V_1 (pour J_1).

Définition 2.2.2 (attracteur)**.**
On définit pour le joueur n^o i :

$$\mathscr{A}_0^i = V_i$$

et pour tout $n \in \mathbb{N}$:

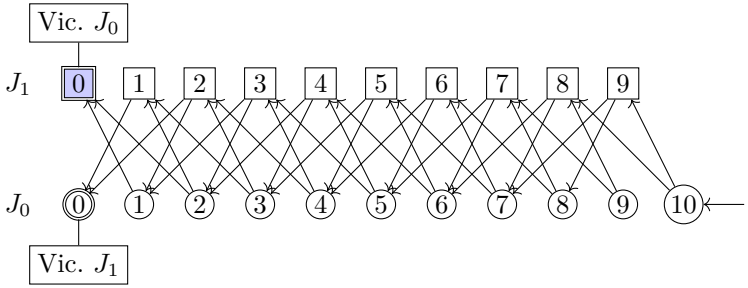
$$\begin{aligned}\mathscr{A}_{n+1}^i = \mathscr{A}_n^i \cup \left\{ x \in S_i \;\middle|\; \exists y \in \mathscr{A}_n^i,\ (x, y) \in A \right\} \\ \cup \left\{ x \in S \setminus S_i \;\middle|\; \exists y \in S,\ (x, y) \in A \quad \text{et} \quad \forall y \in S,\ (x, y) \in A \Rightarrow y \in \mathscr{A}_n^i \right\}.\end{aligned}$$

Ainsi, $\mathscr{A}_n^i$ est l'ensemble des positions gagnantes pour le joueur n^o i en au plus n coups.

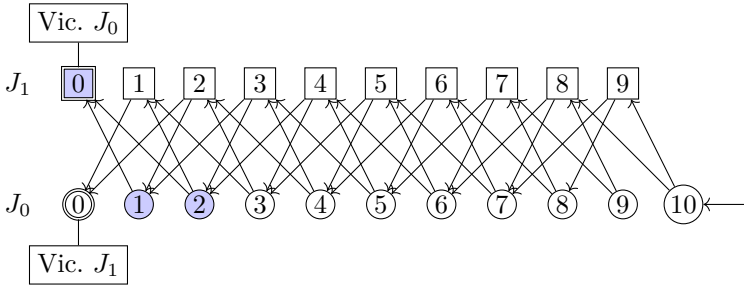
On définit alors l'*attracteur* du joueur n^o i comme l'ensemble des positions gagnantes pour le joueur n^o i, soit

$$\mathscr{A}^i = \bigcup_{n \in \mathbb{N}} \mathscr{A}_n^i.$$

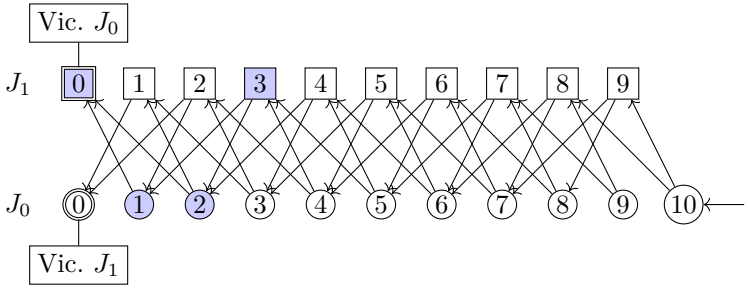
Exemple 2.2.3.
On a représenté dans les figures 4a à 4h les différentes étapes de l'évolution de l'attracteur du joueur J_0 pour le graphe de la figure 1. On remarque que la position de départ est gagnante pour le joueur J_0 : ce dernier est certain de gagner la partie, s'il joue de manière optimale.



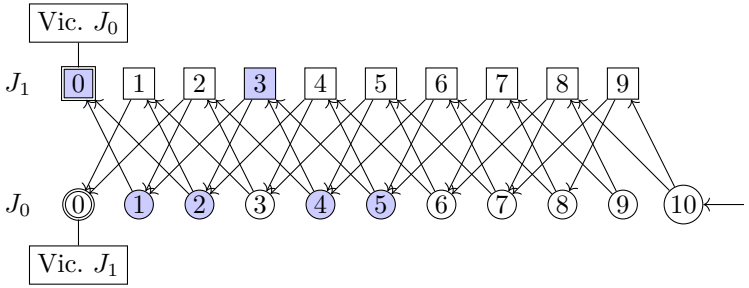
(a) Exemple de $\mathscr{A}_0^0$.



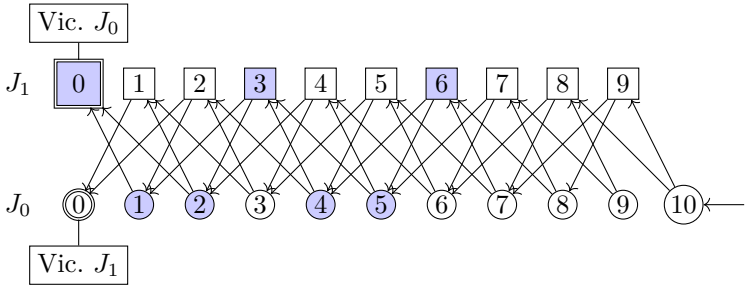
(b) Exemple de $\mathscr{A}_1^0$.



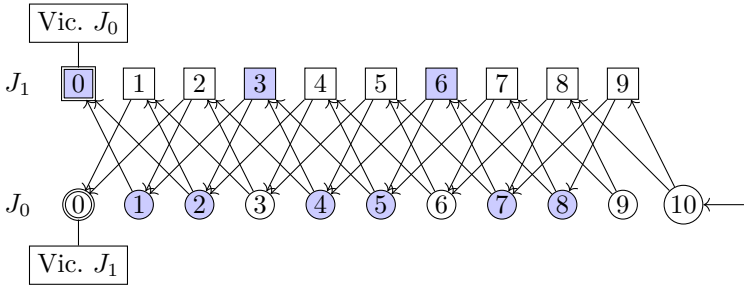
(c) Exemple de $\mathscr{A}_2^0$.



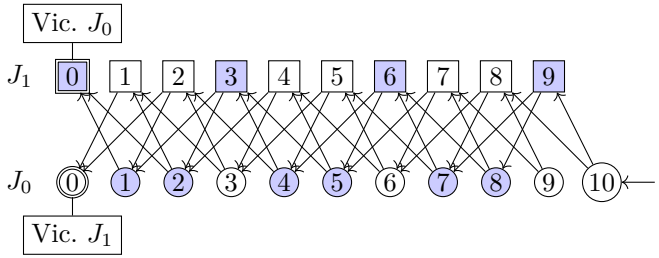
(d) Exemple de $\mathscr{A}_3^0$.



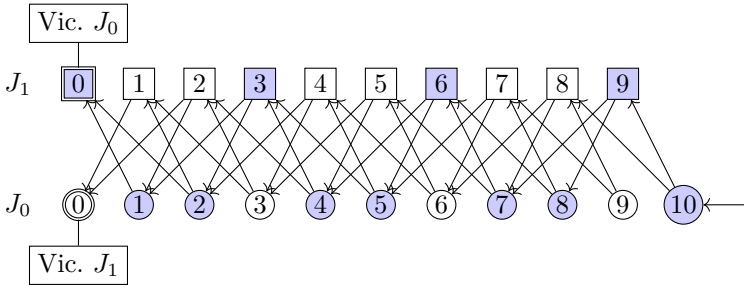
(e) Exemple de $\mathscr{A}_4^0$.



(f) Exemple de $\mathscr{A}_5^0$.



(g) Exemple de $\mathscr{A}_6^0$.



(h) Exemple de $\mathscr{A}_7^0$.

Figure 4 – Évolution des attracteurs pour le jeu de Nim

2.3 Calcul d'une stratégie gagnante

Ainsi, sur le graphe (S, A) :

- ▷ les positions gagnantes pour le joueur J_0 sont celles de $\mathscr{A}^0$;
- ▷ les positions gagnantes pour le joueur J_1 sont celles de $\mathscr{A}^1$;
- ▷ les autres positions sont celles conduisant *a priori* à un match nul, si les deux joueurs jouent de manière optimale.

On peut donc aisément déterminer une stratégie s pour le joueur n° 0, qui sera gagnante si la position de départ est dans $\mathscr{A}^0$. En effet, considérons une position x contrôlée par ce joueur ($x \in S_0$), et examinons les différents cas possibles.

- Soit $x \in \mathscr{A}^0$, dans ce cas il existe $n \in \mathbb{N}$ tel que $x \in \mathscr{A}^0_n$.
  - Si $x \in V_0$, alors il n'y a rien à faire : c'est une position de victoire, qui termine la partie.
  - Sinon, $n \geqslant 1$, et par définition il existe $y \in \mathscr{A}^0_{n-1}$ tel que $(x, y) \in A$. Le joueur J_0 peut donc jouer en y, qui est une position gagnante (plus proche de la victoire). On pose alors $s(x) = y$.
- Soit $x \in \mathscr{A}^1$, dans ce cas il existe $n \in \mathbb{N}$ tel que $x \in \mathscr{A}^1_n$.
  - Si $x \in V_1$, alors il n'y a rien à faire : c'est une position de défaite, qui termine la partie.
  - Sinon, $n \geqslant 1$, dans ce cas pour tout $y \in S$ vérifiant $(x, y) \in A$, alors $y \in \mathscr{A}^1_{n-1}$. Tous les coups possibles pour le joueur J_0 mènent donc à des positions gagnantes pour l'autre joueur. Le joueur peut jouer n'importe quel coup (ils sont tous perdants pour lui). On pose alors $s(x) = y$ pour un tel y.
- Sinon, $x \notin (\mathscr{A}^0 \cup \mathscr{A}^1)$. C'est une position de match nul, à partir de laquelle le joueur J_0 ne peut jouer sur une autre position gagnante, mais qui n'aboutit pas non plus qu'à des positions perdantes pour lui. Il existe donc $y \in S$ vérifiant $(x, y) \in A$ et $y \notin \mathscr{A}^1$: il convient donc de jouer sur ce y : on pose $s(x) = y$.

3 Algorithmes

3.1 Rappels et notations

On reprend les notations de ce chapitre.

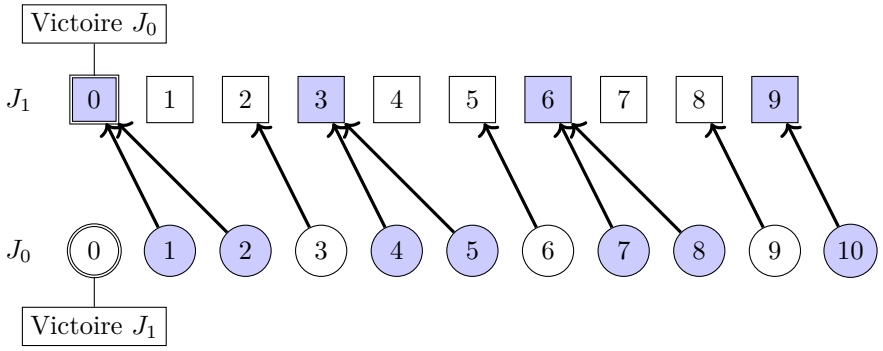


Figure 5 – Exemple de stratégie gagnante pour J_0.

Définition 3.1.1 (degré d'un sommet)**.**
Si $(x, y) \in A$, on dit que x est un *prédécesseur* de y, et y est un *successeur* de x, *i.e.* si une arête va de x vers y.

Si $x \in S$, on appelle *degré sortant* de x son nombre de successeurs, *i.e.* le nombre d'arêtes sortant du sommet x, que l'on note parfois $d^+(x)$. Formellement,

$$d^+(x) = \text{Card}\,\{\, y \in S \mid (x, y) \in A \,\}\,.$$

Si $x \in S$, on appelle *degré entrant* de x son nombre de prédécesseurs, *i.e.* le nombre d'arêtes entrant dans le sommet x, que l'on note parfois $d^-(x)$. Formellement,

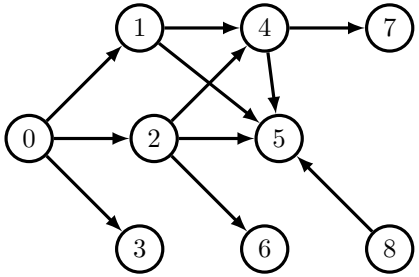
$$d^-(x) = \text{Card}\,\{\, y \in S \mid (y, x) \in A \,\}\,.$$

Exemple 3.1.2.
Dans notre exemple du jeu de Nim de la figure 1, alors le sommet $(1, 0)$ (valeur 1, contrôlée par J_0) est un prédécesseur du sommet $(0, 1)$ et un successeur du sommet $(3, 1)$.

Le sommet $(1, 0)$ a un degré sortant de 1 : une seule arête sort de ce sommet. Son degré sortant est 2 : deux arêtes entrent dans ce sommet.

Exercice 3.1.3.
Calculer les degrés entrants et sortants des sommets du graphe suivant.

Écrire en python sa représentation par dictionnaire de listes d'adjacence.



Définition 3.1.4 (rang d'une position)**.**
Soit x une position gagnante pour J_0 ($x \in \mathscr{A}^0$). On appelle *rang* de x le nombre de coups qu'il faut à J_0 pour gagner à partir de x si les deux joueurs jouent de manière optimale, que l'on note $\mathrm{rg}(x)$.

Formellement :

$$\mathrm{rg}(x) = \min \left\{ n \in \mathbb{N} \;\middle|\; x \in \mathscr{A}_n^0 \right\}.$$

3.2 Calcul des attracteurs

Voici un algorithme pour calculer l'attracteur du joueur J_0.

- On calcule les degrés sortants de chaque position, que l'on affecte à une structure de données. Cette structure comptera le nombre de successeurs de chaque sommets qui ne sont pas des sommets gagnants pour $\mathscr{A}_0$. Lorsque ce compteur atteindra 0 pour un sommet, on saura alors que ce sommet sera gagnant pour $\mathscr{A}_0$.
- On calcule les antécédents de chaque position (ce qui revient à déterminer le graphe obtenu à partir du graphe du jeu, et dont les flèches sont « renversées »).
- On crée un attracteur vide (cela peut-être un dictionnaire).
- On commence le calcul sur les positions de victoire de J_0, que l'on sait être des sommets gagnants.
- Pour chaque sommet que l'on sait être gagnant : si le sommet n'a pas déjà été rencontré, donc s'il n'est pas déjà dans l'attracteur que l'on est en train de construire,
  - on ajoute ce sommet à l'attracteur
  - pour chaque prédécesseur de ce sommet :
    - on retranche 1 à son compteur de successeurs non gagnants ;
    - si ce prédecesseur est dans S_0, il est gagnant, on déclenche donc un appel récursif dessus ;
    - sinon, si ce prédecesseur n'a plus que des successeurs gagnants (ce qui se lit dans le compteur associé), il est gagnant, on déclenche donc un appel récursif dessus.

Remarque 3.2.1.
Cet algorithme a une complexité temporelle en $\mathcal{O}(|S| + |A|)$.

On supposera dans cette partie que le graphe est décrit par un dictionnaire d'adjacences, que l'on note `A` : les clefs de `A` sont les sommets du graphe, la valeur associée à une clef est la liste des successeurs de ce sommet.

Exemple 3.2.2 (dictionnaire d'adjacence du jeu de Nim)**.**
Voici ce que l'on écrirait en Python pour créer le graphe du jeu de Nim de la figure 1.

```
1  A = {(1,0) :[(0,1)],
2       (1,1) :[(0,0)]
3       (10,0) :[(8,1),(9,1)]}
4  for k in range(2,10) :
5      for i in [0,1] :
6          A[(k,i)] = [(k-2,1-i),(k-1,1-i)]
```

On suppose aussi que les ensembles S_0, S_1, V_0, V_1 sont décrits par des dictionnaires, dont les clefs sont les sommets de ces ensembles, et dont les valeurs sont arbitraires. On pourra donc tester la présence d'un élément dans un de ces ensembles en temps $\mathcal{O}(1)$.

Exemple 3.2.3 (Positions des joueurs et condition de victoire du jeu de Nim)**.**
Voici ce que l'on écrirait en Python pour créer les positions des joueurs et les conditions de victoire du jeu de Nim de la figure 1.

```
7  V0 = {(0,1) :None}
8  V1 = {(1,0) :None}
9  S0 = {(k,0) :None for k in range(11)}
10 S1 = {(k,1) :None for k in range(10)}
```

Notamment, on pourra tester la présence d'un sommet s dans $S0$ par la commande `s in S0`, en temps $\mathcal{O}(1)$.

Commençons par écrire la fonction qui prend en argument le dictionnaire d'adjacence et qui renvoie le dictionnaire d'adjacence du graphe « renversé », *i.e.* le dictionnaire dont les clefs sont les sommets du graphe et dont la valeur associée à une clef est la liste des prédecesseur de ce sommet.

```
Fonction renverse_graphe(A) :
    B ← dictionnaire vide
    Pour chaque x De A, Faire
        B[x] ← liste vide
    Pour chaque x De A, Faire
        Pour chaque y De A[x], Faire
            Ajouter x à B[y]
    Renvoyer B
Fin Fonction
```

Il est aussi facile de créer un dictionnaire comptant les degrés sortants.

```
Fonction dplus(A) :
    d ← dictionnaire vide
    Pour chaque x De A, Faire
        d[x] ← longueur de A[x]
    Renvoyer d
Fin Fonction
```

Nous pouvons maintenant écrire une fonction récursive qui marque les sommets gagnants, et effectue la propagation détaillée ci-dessus. Elle prend en argument un sommet gagnant `s`, le dictionnaire des prédecesseurs `B`, l'ensemble des sommets contrôlés par J_0, `S0`, ainsi que le dictionnaire `d` comptant le nombre de successeurs non gagnants de chaque sommet.

Ici, l'attracteur est supposé être décrit par un dictionnaire, exactement comme l'ensemble `S0`.

Cette fonction est à effets de bord !

```
Fonction marque_propage(s,d,B,S0,attracteur) :
    Si s ∉ attracteur Alors
        attracteur[s] ← valeur arbitraire
        Pour chaque x De B[s], Faire
            d[x] diminué de 1
            Si x ∈ S0 ou d[x]=0 Alors
                marque_propage(x,d,B,S0,attracteur)
Fin Fonction
```

Il suffit maintenant de tout englober dans une seule fonction.

```
Fonction calcule_attracteur(A,S0,V0) :
    d ← dplus(A)
    B ← renverse_graphe(A)
    attracteur ← dictionnaire vide
    Pour chaque s De V0, Faire
        marque_propage(s,d,B,S0,attracteur)
    Renvoyer attracteur
Fin Fonction
```

3.3 Calcul d'une stratégie gagnante

Le calcul d'une stratégie gagnante repose fondamentalement sur le calcul des attracteurs. Il convient cependant de modifier légèrement le pseudo-code donné précédemment. En effet, pour déterminer le coup à jouer sur une position gagnante, il ne suffit pas de jouer sur une autre position gagnante, mais de jouer sur une position gagnante plus proche d'une condition de victoire (V_0 pour le joueur J_0).

Il suffit alors, lors du calcul de l'attracteur, de numéroter les sommets gagnants, en partant de la valeur 0 sur la condition de victoire, et en l'incrémentant de 1 à chaque appel récursif. Si un sommet gagnant est ainsi étiqueté avec la valeur $n \in \mathbb{N}$, cela signifie que son rang est inférieur ou égal à n, et que le joueur pourra gagner la partie en moins de n coups.

On pourra remarquer que l'on ne calcule pas ici exactement le rang d'une position gagnante, mais plutôt une majoration du rang, ce qui est suffisant dans notre cadre.

Commençons par réécrire les fonctions `marque_propage` et `calcule_attracteur` de la partie précédente.

```
Fonction marque_propage2(s,d,B,S0,attracteur,n) :
    Si s ∉ attracteur Alors
        attracteur[s] ← n
        Pour chaque x De B[s], Faire
            d[x] diminué de 1
            Si x ∈ S0 ou d[x]=0 Alors
                marque_propage2(x,d,B,S0,attracteur,n+1)
Fin Fonction
```

```
Fonction calcule_attracteur2(A,S0,V0) :
    d ← dplus(A)
    B ← renverse_graphe(A)
    attracteur ← dictionnaire vide
    Pour chaque s De V0, Faire
        marque_propage2(s,d,B,S0,attracteur,0)
    Renvoyer attracteur
Fin Fonction
```

Ainsi, la fonction précédente renvoie un dictionnaie `attracteur` construit de telle sorte que si `s` est une position gagnante, alors `attracteur[s]` est un entier n vérifiant les conditions discutées au début de cette partie.

On peut alors écrire une fonction qui crée une stratégie pour le joueur J_0. Cette fonction prendra comme argument le dictionnaire `A`, les attracteurs des joueurs J_0 (`attr0`) et J_1 (`attr1`), ainsi qu'une position `s` contrôlée par J_0, que l'on suppose non gagnante et non terminale (*i.e.* cette position possède au moins un successeur).

```
Fonction strategie(A,attr0,attr1,s) :
    Si s∈attr0 Alors
        Pour chaque x De A[s], Faire
            Si x∈attr0 et attr0[x]<attr0[s] Alors
                Renvoyer x
    Sinon si s∈attr1 Alors
        Renvoyer un élément quelconque de A[s]
    Sinon
        Pour chaque x De A[s], Faire
            Si x∉ attr1 Alors
                Renvoyer x
Fin Fonction
```

4 Exercices

4.1 Jeu de Nim

Dresser le graphe du jeu et les positions gagnantes pour chaque joueur pour le jeu de Nim où :

▷ on commence avec 15 bâtonnets ;

▷ à chaque tour, on peut enlever 2, 3 ou 5 bâtonnets (la règle est $\{2, 3, 5\}$).

Représenter une stratégie gagnante pour le joueur concerné, en surlignant les arêtes de la stratégie.

4.2 Un jeu arbitraire

Pour le jeu décrit dans la figure 6, déterminer les attracteurs de J_0 et de J_1. Est-ce qu'un joueur est certain de gagner ? Déterminer une stratégie gagnante pour ce joueur.

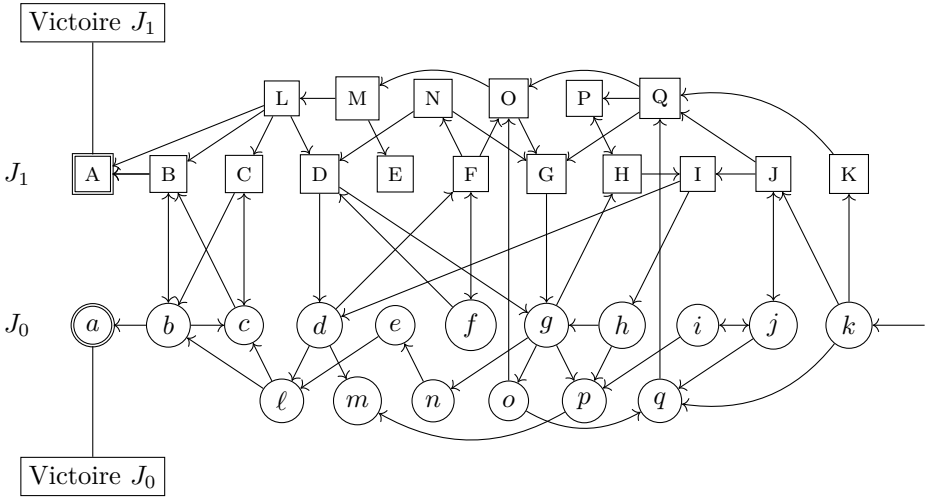


FIGURE 6 – Un jeu sur un graphe.