# TP 4
# Initiation à la théorie des jeu et retour sur la programmation dynamique

## 1 Introduction à la théorie des jeux

Dans ce chapitre, on donne une introduction à l'étude de la théorie des jeux. Les jeux que l'on va étudier sont des jeux à information complète, ce qui exclut la plupart des jeux de cartes (chaque joueur n'a pas connaissance du jeu adverse et éventuellement du reste du paquet), et sans hasard. On ne s'intéresse qu'à des jeux d'accessibilité à deux joueurs, qui peuvent être modélisés par des graphes orientés, jouer un coup consiste à suivre un arc. Dans un premier temps, on discute de jeux qui sont suffisamment simples pour pouvoir être résolus complètement : l'espace des états possibles n'est pas trop grand. Dans un second temps, on aborde la notion de stratégie avec heuristique, qui peut s'appliquer aux jeux qu'il est impossible (à l'heure actuelle, au vu des contraintes matérielles) de résoudre complètement, comme le jeu d'échecs ou le jeux de go.

### 1.1 Jeu d'accessibilité sur un graphe

Introduction. Dans l'émission Fort Boyard, le duel de bâtonnets [avec le maître des ténèbres est un jeu à deux joueurs. Au départ, un tas comporte 20 bâtonnets, et à son tour un joueur peut en enlever 1,2 ou 3 . Le perdant est celui qui prend le dernier bâtonnet. Voici un exemple de partie :

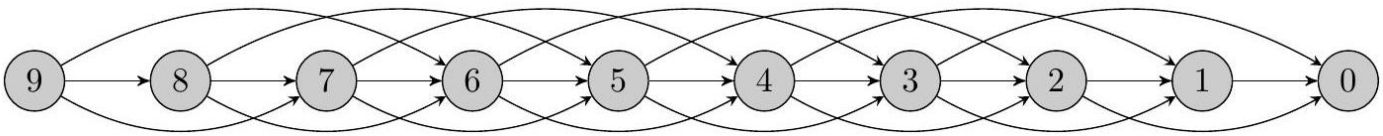
$$20 \longrightarrow 17 \longrightarrow 16 \longrightarrow 15 \longrightarrow 12 \longrightarrow 10 \longrightarrow 8 \longrightarrow 7 \longrightarrow 4 \longrightarrow 3 \longrightarrow 1 \longrightarrow 0$$

Ce jeu est particulièrement simple, et il est facile de voir que c'est le joueur qui joue en premier qui gagne, s'il suit la stratégie de laisser à l'autre joueur un nombre de bâtonnets qui est de la forme $4n + 1$.

Modélisation par un 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. Dans le jeu précédent, les arcs issus du sommet 7 pointent vers 6,5 et 4 , comme on le voit figure 18.1 .




Un jeu de Nim avec seulement 9 bâtonnets au départ

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.

**Propriété 1.1** *Si le graphe $G$ est sans circuit, toute partie est finie.*

Généralisation : graphe biparti. Il est utile dans la pratique de «dédoubler» un graphe comme le précédent en un graphe biparti, de sorte qu'une partie dans le jeu se résume simplement à un chemin dans le graphe biparti, sans avoir à distinguer quel joueur joue.

**Définition 1.1** *Un graphe $G = (V, E)$ est dit biparti s'il existe une partition de $V$ en deux sous-ensembles $V_a$ et $V_b$, telle qu'aucun arc du graphe ne relie deux sommets de $V_a$, ou deux sommets de $V_b$.*

**Exemple 1.1** *Le graphe biparti associé au graphe de la figure 18.1 est le suivant.*

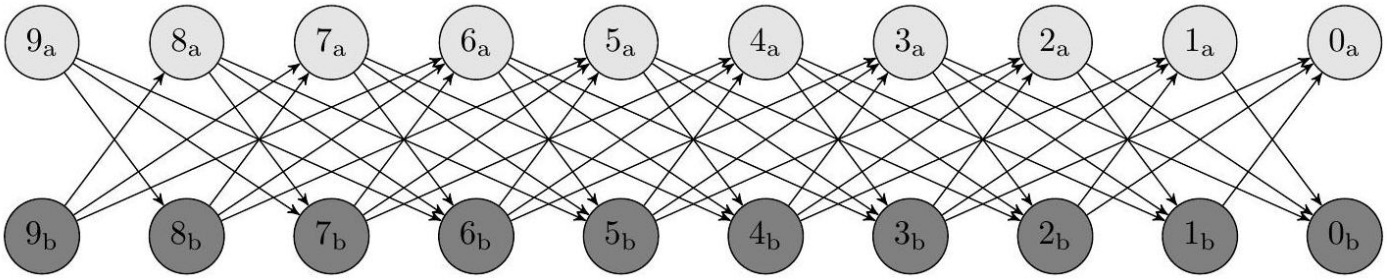



*Figure 18.2 - Le graphe biparti associé au jeu de Nim avec 9 bâtonnets au départ. Le joueur a commence (en $9_a$ ), et donc le sommet $9_b$ ne sera jamais atteint.*

**Exercice 1.1** *Proposer une fonction Python renvoyant la liste d'adjacence des noeuds sortants du graphe du Jeu de Nim pour n batons,* `Nim(n)`*. Cette liste d'adjacence sera renvoyée sous la forme d'un dictionnaire de clé le couple* `('Va',i)` *( respectivement* `('Vb',i)`*) pour un noeud de la partition $V_a$ ( respectivement $V_b$) i le nombre de bâtonnets.*

**Exercice 1.2** *Proposer une fonction* `transpose(G)`*, prenant la liste d'adjacence, des noeuds sortant, G d'un graphe et renvoyant la liste transposé, c-a-d la liste des antécédents, noeuds entrants, sous forme de dictionnaire où la clé est un noeud u de G et la valeur, la liste des noeuds v tel qu'il existe une arrete de u vers v.(par exemple le couple $(Va, i)$ ( respectivement $(Vb, i)$) pour un noeud de la partition $V_a$ ( respectivement $V_b$) i le nombre de bâtonnets et la valeur la liste des antécédents.)*

## 1.2 Vocabulaire : arène, condition de victoire, positions gagnantes et stratégies

On commence par définir une arène dans un jeu à deux joueurs.

**Définition 1.2** *(Arène). Un graphe de jeu (aussi appelé une arène) est un triplet $(G, V_1, V_2)$, où $G = (V, E)$ est un graphe orienté, avec $V$ partitionné en $V_1 \cup V_2$.*

Les sommets d'un tel graphe se partitionnent donc en deux ensembles $V_1$ et $V_2$, $V_1$ étant l'ensemble des sommets contrôlés par le premier joueur et $V_2$ ceux contrôlés par le second.

**Définition 1.3** *(Partie). Une partie sur un tel graphe depuis un sommet $v_0$ se déroule comme suit :*

- *le joueur contrôlant $v_0$ choisit, s'il en existe un, un sommet $v_1$ tel que $(v_0, v_1) \in E$ ;*
- *le joueur contrôlant $v_1$ choisit, s'il en existe un, un sommet $v_2$ tel que $(v_1, v_2) \in E$ ;*
- *etc...*

*Autrement dit, une partie $\mathcal{P}$ est un chemin $v_0, v_1, \ldots$, maximal dans le graphe, possiblement infini, où chaque joueur contrôlant un sommet choisit un arc vers un autre sommet du graphe si c'est possible, sinon la partie s'arête.*

**Remarque 1.1** *Souvent, comme dans la section précédente, le graphe considéré sera biparti, de sorte que les joueurs jouent alternativement : aucun arc du graphe ne relie deux sommets de $V_1$ ou de $V_2$.*

**Définition 1.4** *(Condition de victoire). Une condition de victoire (pour le premier joueur) est un sous-ensemble des parties possibles. Le complémentaire est la condition de victoire du second joueur.*

En théorie des jeux, il existe de multiples conditions de victoire. La plus simple (et la seule au programme) est la condition d'atteignabilité.

**Définition 1.5** *(Condition d'atteignabilité). Un condition d'atteignabilité pour le premier joueur est un sous-sensemble $W$ de $V$. La partie est remportée par le premier joueur si celle-ci visite un sommet de $W$, remportée par le second joueur dans le cas contraire.*

**Remarque 1.2** *(Sommet de degré sortant nul). Dans la modélisation précédente, on a implicitement supposé qu'un sommet $v$ de degré sortant nul contrôlé par le second joueur est dans l'ensemble $W$ : si le deuxième joueur ne peut pas jouer, le premier gagne. Ceci n'est pas restrictif : si on voulait déclarer le second joueur vainqueur lorsque la partie aboutit à $v$, on pourrait rajouter un sommet $v'$ contrôlé par le second joueur, et des arcs $(v, v')$ et $(v', v)$.*

**Exemple 1.2** *(Retour sur le jeu des bâtonnets). Puisque le jeu des bâtonnets est considéré comme perdu par le joueur qui prend le dernier bâtonnet, on peut modéliser le jeu comme un jeu d'accessibilité en enlevant les sommets $0_x$, et en prenant $W = \{1_b\}$.*

**Définition 1.6** *(Stratégie). Pour $G = (V, E)$ un graphe, notons $V_i^{>0}$ l'ensemble des sommets de degré sortant non nul contrôlés par le joueur $i \in \{1, 2\}$. Une stratégie est une application $\varphi : V_i^{>0} \to V$, telle que $\forall s \in V_i^{>0}$, $(s, \varphi(s)) \in E$.*

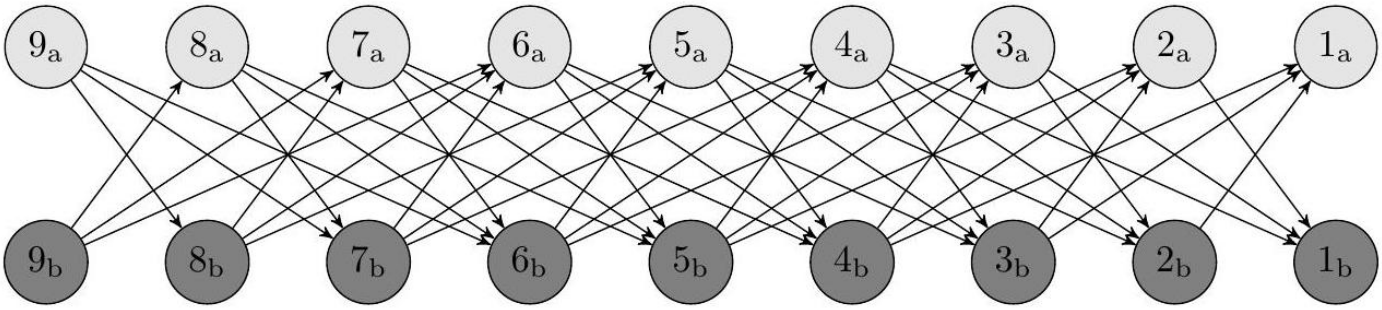



Figure 18.3 - Jeu d'accessibilité des bâtonnets : la condition de victoire pour le premier joueur se résume à atteindre $1_b$. Le premier joueur gagne si ce sommet est atteint, le second gagne dans le cas contraire.

On dit que le joueur $i (i \in \{1, 2\})$ respecte la stratégie $\varphi$ lors d'une partie $\mathcal{P} = (v_0, v_1, \ldots)$ si

$$\forall j \quad v_j \in V_i^{>0} \Longrightarrow v_{j+1} = \varphi(v_j)$$

Autrement dit, lorsque c'est au joueur $i$ de jouer (et qu'il peut jouer !), il choisit le sommet donné par la stratégie.

**Remarque 1.3** *Les stratégies considérées ici sont en fait sans mémoire, c'est-à-dire que le prochain sommet joué ne dépend que du sommet courant et pas des précédents. C'est tout à fait adapté pour des conditions d'accessibilité. Avec d'autres conditions de victoire, il faudrait envisager des stratégies où le choix du prochain sommet dépend de tous les précédents.*

**Définition 1.7** *(Stratégie gagnante). Une stratégie $\varphi$ est dite gagnante pour le joueur $i (i \in \{1, 2\})$ depuis le sommet $v_0 \in V$, si toute partie jouée depuis $v_0$ où le joueur $i$ respecte la stratégie $\varphi$ est gagnante pour $i$.*

***Définition 1.8 Exemple 1.3*** *(Jeu des bâtonnets). On considère le graphe des bâtonnets, avec n bâtonnets au départ, de sommets $V = \{1_a, \ldots, n_a, 1_b, \ldots, n_b\} = V_1 \cup V_2$ avec $V_1 = \{1_a, \ldots, n_a\}$ et $V_2 = \{1_b, \ldots, n_b\}$ comme défini précédemment. La stratégie consistant (si possible) à laisser à l'autre joueur un nombre de bâtonnets qui est de la forme $4n + 1$ est une stratégie gagnante pour chaque joueur, depuis un sommet qui n'est pas de la forme $(4n + 1)_a$ ou $(4n + 1)_b$.*

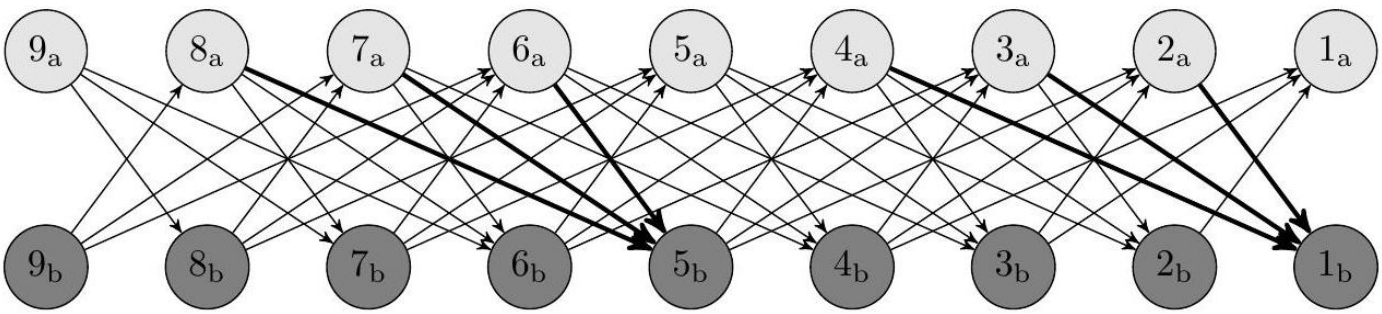


*Figure 18.4 - Jeu d'accessibilité des bâtonnets : en gras, les arcs d'une stratégie gagnante pour le premier joueur. Les positions $9_a$ et $5_a$ étant perdantes, la définition de la stratégie sur ces sommets n'a pas d'importance.*

***Définition 1.9*** *(Position gagnante). Un sommet $v \in V_i$ est appelé position gagnante pour le joueur $i$ si celui-ci possède une stratégie gagnante depuis $v$.*

*Dans l'exemple précédent, on voit que $9_a$ n'est pas une position gagnante pour le joueur $a$, contrairement à $8_a, 7_a$ et $6_a$.*

## 1.3 Attracteur

*Dans un jeu d'accessibilité, résoudre le jeu consiste essentiellement 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, qui doit pour gagner atteindre un ensemble $W$. L'outil adéquat est l'attracteur.*

*Ensembles définissant l'attracteur. On se donne donc un graphe $G = (V, E)$, et deux ensembles $V_1, V_2$ formant une partition de $V$, et $W$ l'ensemble de sommets définissant la condition de victoire du premier joueur. On définit incrémentalement une suite de sous-ensembles de sommets comme suit :*

$$\mathcal{A}_0 = W \quad et \quad \forall j \geq 0 \quad \mathcal{A}_{j+1} = \mathcal{A}_j \cup \{v \in V_1 \,|\exists v' \in \mathcal{A}_i|\, (v, v') \in E\} \cup \{v \in V_2 \,|\forall v' \in V|\, (v, v') \in E \Rightarrow v' \in \mathcal{A}_i\}$$

*En français : l'ensemble $\mathcal{A}_{j+1}$ est constitué*

- *des sommets de $\mathcal{A}_j$*
- *des sommets contrôlés par le premier joueur pour lesquels il existe au moins un arc permettant de rejoindre un sommet de $\mathcal{A}_j$*
- *des sommets contrôlés par le second joueur dont tous les arcs aboutissent à des sommets de $\mathcal{A}_j$.*

*La suite $(\mathcal{A}_j)_{j\geq 0}$ est clairement croissante au sens de l'inclusion, l'attracteur de $W$ est défini par l'union croissante des $(\mathcal{A}_j)$.*

***Définition 1.10*** *L'attracteur de $W$ est l'ensemble $\mathcal{A} = \cup_{j=0}^{+\infty} \mathcal{A}_j$*

*On va voir que l'attracteur contient précisément les positions gagnantes du premier joueur.*

***Théorème 1.1*** *Avec les notations précédentes, le premier joueur possède une stratégie gagnante pour tout sommet de $\mathcal{A}$, son adversaire possède une stratégie gagante pour tout sommet de ${}^c\mathcal{A}$.*

*Démonstration. Pour tout sommet $v$ de $V$, on pose*

$$\text{Rang}(v) = \inf\{j \geq 0 \mid v \in \mathcal{A}_j\} \in \mathbb{N} \cup \{+\infty\}$$

*(le rang est fini pour chaque sommet de l'attracteur, sinon il vaut $+\infty$ ). Montrons par récurrence sur le rang que les sommets de $\mathcal{A}$ sont des positions gagnantes du premier joueur :*

- *C'est évident pour les sommets de rang 0 , qui sont dans $W$ ;*
- *Supposons la propriété démontrée jusqu'au rang $j \geq 0$, et considérons un sommet $v$ de rang $j+1$ (qui est donc dans $\mathcal{A}_{j+1} \backslash \mathcal{A}_j$).*
- *s'il est contrôlé par le premier joueur, alors par définition il existe un sommet $v'$ de $\mathcal{A}_j$ et l'arc $(v, v')$ appartient à $E$. Le joueur peut donc jouer cet arc. Puisqu'il possède par hypothèse de récurrence une stratégie gagnante depuis $v'$, il en possède une depuis $v$ ;*
- *s'il est contrôlé par le second joueur, par définition, tout arc depuis $v$ aboutit à un sommet de $\mathcal{A}_j$ : quel que soit le choix du deuxième joueur, le premier possède une stratégie gagnante depuis le sommet choisi, donc $v$ est également une position gagnante pour le premier joueur.*

*Inversement, si $v \notin \mathcal{A}$, on vérifie que $v$ est une position gagnante du second joueur en construisant de la même manière une stratégie gagnante pour ce joueur : si $v$ est contrôlé par le second joueur, celui-ci peut joueur un sommet $v'$ qui n'est pas non plus dans $\mathcal{A}$, et si $v$ est contrôlé par le premier joueur, tout choix de celui-ci (s'il peut jouer) aboutit à un sommet qui n'est pas non plus dans $\mathcal{A}$. Le second joueur peut donc faire en sorte que la partie évite $W$, il remporte donc la partie.*

***Remarque 1.4*** *Dans la preuve précédente, on a supposé que les sommets de degré sortant nul contrôlés par le deuxième joueur sont dans $W$*

## 1.4 Algorithme de calcul de l'attracteur

*Pour calculer l'attracteur, il suffit essentiellement de faire un parcours de graphe, mais sur le graphe transposé de $G$ : en effet on part de l'ensemble $W$ 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 ${}^tG$ ), pour gérer la condition « tout arc sortant d'un sommet contrôlé par le deuxième joueur aboutit à un sommet de l'attracteur».*

*Terminaison de l'algorithme. Il s'agit essentiellement d'un parcours de graphe : en dehors des appels à la fonction récursive parcours, l'algorithme termine. De plus, un appel à* parcours($u$) *n'est réllement utile que si $u \notin \mathcal{A}$, il y aura donc au plus un appel utile pour chaque sommet. Ainsi, l'algorithme termine.*

*Correction de l'algorithme. On veut montrer qu'à la fin de l'algorithme, l'ensemble $\mathcal{A}$ contient précisément l'attracteur associé à $W$. Montrons déja que la propriété «tout élément de $\mathcal{A}$ est un élément de l'attracteur» est un invariant de l'algorithme.*

- *au début, $\mathcal{A}$ est vide, donc la propriété est vérifiée.*
- *si parcours $(u)$ est lancé dans la boucle principale, c'est que $u \in W$, donc $u$ est bien dans l'attracteur.*
- *sinon, c'est que parcours $(v)$ est lancé pendant parcours $(u)$, avec $u$ dans l'attracteur. Si $v \in V_1$, cela signifie que l'arc $v \to u$ est présent dans le graphe $G$, donc par définition $v$ est dans l'attracteur. Sinon, cela signifie que $n_v = 0$. Or $n_v$ est décrémenté (au plus) une fois par sommet $u$ de l'attracteur tel que l'arc $v \to u$ est présent dans le graphe : si $n_v$ atteint 0 , cela signifie que tous les sommets que l'on peut atteindre depuis $v$ sont dans l'attracteur, donc par définition $v$ l'est aussi.*

**Algorithme 18.19 :** Calcul de l'attracteur

**Entrée :** Un graphe $G = (V, E)$ donné par listes d'adjacence, une partition de $V$ en deux ensembles $V_1 \cup V_2$, un ensemble $W$
**Sortie :** L'attracteur associé à $W$

```
A ← ∅;
Pour chaque sommet v, calculer n_v, le degré sortant de v dans G;
Calculer ᵗG, graphe transposé de G;
Fonction parcours(u) :
    si u ∉ A alors
        A ← A ∪ {u};
        pour tout voisin v de u dans ᵗG faire
            n_v ← n_v − 1;
            si v ∈ V_1 ou n_v = 0 alors
                parcours(v)
pour tout sommet u de W faire
    parcours(u)
```

*Montrons maintenant que tout sommet de l'attracteur se retrouve dans l'ensemble $\mathcal{A}$ à la fin de l'algorithme. Supposons que ce ne soit pas le cas et considérons un sommet $v$ de l'attracteur, de rang minimal parmi ceux qui ne se retrouvent pas dans $\mathcal{A}$.*

*$-v$ n'est pas de rang 0 ;*

- *si $v \in V_1$, alors par hypothèse il existe un sommet $u$ dans l'attracteur, de rang strictement inférieur, tel que l'arc $(v, u)$ soit présent dans le graphe. Dans l'appel* parcours($u$) *où $u$ est ajouté à $\mathcal{A}$, parcours $(v)$ aurait du être lancé, c'est absurde.*
- *sinon, $v \in V_2$. Par hypothèse, tous les sommets accessibles depuis $v$ dans $G$ sont dans l'attracteur, et ont un rang strictement inférieur par définition. Notons $u$ le dernier à être ajouté à $\mathcal{A}$ : lorsque $v$ est examiné dans la liste d'adjacence de $u$ dans ${}^tG, n_v$ passe à zéro et parcours $(v)$ aurait du être lancé, c'est donc absurde.*

*Complexité de l'algorithme. L'algorithme peut être implémenté avec une complexité $O(|V| + |E|)$ : il suffit pour cela d'encoder $V_1, W$ et $\mathcal{A}$ comme des listes de booléens.*

***Exercice 1.3*** *Programmer une fonction Python prenant comme paramètre la liste d'adjacence $G$, la liste des noeuds de $W$ et renvoyant la liste des noeuds de $\mathcal{A}$, pour le joueur $a$.*

***Exercice 1.4*** *Programmer une fonction* `phi(G,u)` *renvoyant une stratégie gagnante, c-a-d , qui renvoie pour le graphe G, d'attracteur A, le noeud $\phi(u)$ que doit joueur le joueur A.*

*On peut aussi proposer :*

*Modification de l'algorithme pour le calcul d'une stratégie Pour calculer en parallèle une stratégie pour le joueur 1, il suffit de reprendre l'algorithme : lorsqu'on lance parcours $(v)$ dans parcours $(u)$, on peut poser $\varphi(v) = u$. Pour calculer une stratégie gagnante pour le joueur 2 sur un sommet $u \in V_2$ n'appartenant pas à l'attracteur, il suffit de parcourir la liste d'adjacence de u à la recherche d'un sommet v qui n'est pas non plus dans l'attracteur (le coût total du calcul de la stratégie est également en $O(|V| + |E|)$). Ceci est laissé en exercice !*

***Remarque 1.5*** *Dans cette partie, on s'est placé dans le cadre le plus simple des jeux d'accessibilité : le joueur 1 gagne s'il atteint un certain ensemble d'états, sinon c'est le joueur 2. Une situation un peu plus complexe, mentionnée dans le programme officiel, est la suivante : le joueur 1 gagne s'il atteint un certain ensemble $W_1$, le joueur 2 gagne s'il atteint un ensemble $W_2$, autrement la partie est nulle. Dans ce cas, en supposant le graphe sans circuit [2], il suffit essentiellement d'appliquer deux fois l'algorithme : les sommets du graphes se répartissent en trois morceaux :*

- *les sommets gagnants pour le premier joueur (l'attracteur de $W_1$ ) ;*
- *les sommets gagnants pour le second joueur (l'attracteur de $W_2$ ) ;*
- *les autres sommets, où si chaque joueur joue de manière optimale, la partie est nulle.*

*Un exemple est le jeu du morpion, sur un carré $3 \times 3$ : la position initiale n'est une position gagnante ni pour le premier joueur, ni pour le second.*

## 2 L'algorithme de Floyd Warshall.

*En informatique, l'algorithme de Floyd-Warshall est un algorithme pour déterminer les distances des plus courts chemins entre toutes les paires de sommets dans un graphe orienté et pondéré, en temps cubique au nombre de sommets.*

*L'algorithme de Floyd-Warshall prend en entrée un graphe orienté et valué, décrit par une matrice d'adjacence donnant le poids d'un arc lorsqu'il existe et la valeur $+\infty$ sinon. Le poids d'un chemin entre deux sommets est la somme des poids sur les arcs constituant ce chemin. Les arcs du graphe peuvent avoir des poids négatifs, mais le graphe ne doit pas posséder de cycle de poids strictement négatif. L'algorithme calcule, pour chaque paire de sommets, le poids minimal parmi tous les chemins entre ces deux sommets.*

*On suppose que les sommets de G sont $\{1, 2, 3, 4, ..., n\}$. Il résout successivement les sous-problèmes suivants :*

*$W^k_{i,j}$ est le poids minimal d'un chemin du sommet i au sommet j n'empruntant que des sommets intermédiaires dans $\{1, 2, 3, ..., k\}$ s'il en existe un, et $+\infty$ sinon. On note $W^k$ le tableau des $W^k_{i,j}$. Pour $k = 0$, $W^0$ est la matrice d'adjacence définissant G. Maintenant, pour trouver une relation de récurrence, on considère un chemin p entre i et j de poids minimal dont les sommets intermédiaires sont dans $\{1, 2, 3, ..., k\}$. De deux choses l'une :*

- *soit p n'emprunte pas le sommet k ;*
- *soit p emprunte exactement une fois le sommet k (car les circuits sont de poids positifs ou nuls) et p est donc la concaténation de deux chemins, entre i et k et k et j respectivement, dont les sommets intermédiaires sont dans $\{1, 2, 3, ..., k-1\}$.*

*L'observation ci-dessus donne la relation de récurrence :*

$$W^k_{i,j} = \min_{(i,j)\in[|1,n|]^2}(W^{k-1}_{i,j}, W^{k-1}_{i,k} + W^{k-1}_{k,j})$$

*pour tous $i, j$ etk dans $\{1, 2, 3, 4..., n\}$. Ainsi on résout les sous-problèmes par valeur de k croissante.*

***Exercice 2.1*** *On supposera que le graphe G, nous est donné sous la forme d'une liste d'ajacence stocker dans un dictionnaire* `G`*, où* `G[i]` *renvoie la liste des uplets $(v_j, p_j)$ de telle sorte que $p_j$ soit le poids ( relatif de l'arrête de $u_i$ vers $v_j$.*

1. *Programmer une fonction* `Adj(G)` *retournant sous forme de tableau, la matrice d'ajacence évoquée ci-dessus et prenant comme paramètre le dictionnaire* `G` *de la liste d'ajacence du graphe ( on prendra garde au fait que la ligne $i = 0$ est la première ligne).*

2. *Proposez une fonction* `cycle(A)`*, renvoyant true si la matrice d'ajdacence n'a aucun cocycle de longueur* 1 *de poids négatif et* `False` *sinon.*
3. *Programmer une fonction* `FW(A,k)` *prenant comme paramètre le tableau* $A$ *de la matrice d'adjacence et renvoyant* $W^k$ *et s'arrête si le graphe pocède un cocycle de poids négatif.*
4. *Terminer le programme de la fonction* `FloydWarshall(G)` *renvoyant la distance du plus cours chemin.*