Jeu de Nim

Dans ce jeu de Nim, deux joueurs notés J_1 et J_2 se retrouvent en face d'un ensemble de n bâtonnets en bois. Chacun à leur tour, ils peuvent en retirer un certain nombre :

Le joueur J_1 peut en retirer 1 ou 2

Le joueur J_2 peut en retirer 1 ou 3

Celui qui prend le dernier bâtonnet perd : **Autrement dit, celui qui se retrouve sans bâtonnet a gagné**.

Pour le jeu de Nim avec n bâtonnets au départ, on notera (k, i) la situation où le joueur $i \in \{1, 2\}$ joue, avec k bâtonnets placés devant lui, et on a donc :

- L'ensemble des positions contrôlées par le joueur 1 est $S_1 = \{(k, 1) \mid 0 \leq k \leq n\}$.
- L'ensemble des positions contrôlées par le joueur 2 est $S_2 = \{(k, 2) \mid 0 \leq k \leq n\}$.

Les arêtes correspondent aux coups qui peuvent être joués. On a donc les arêtes :

- $(k, 1) \to (k - q, 2)$ pour $q \in \{1, 2\}$ et $k - q \geq 0$, lorsque le joueur J_1 joue
- $(k, 2) \to (k - q, 1)$ pour $q \in \{1, 3\}$ et $k - q \geq 0$, lorsque le joueur J_2 joue.

L'unique position finale gagnante pour le joueur J_1 est $(0, 1)$, et de même $(0, 2)$ pour le joueur J_2.

1) Représentation du graphe du jeu de Nim

L'arène (c'est-à-dire le graphe biparti G associé au jeu) est représentée par un dictionnaire dont les clés sont les sommets (k, i), avec $0 \leq k \leq n$ et $i \in \{1, 2\}$ et la valeur d'un sommet est la liste des successeurs de ce sommet.

a) Créer une fonction `arene(n)` qui renvoie un dictionnaire `G` représentant le graphe de Nim.

Par exemple, `arene(0)` renvoie le dictionnaire `{(0,1):[],(0,2):[]}`

b) Créer une fonction `sommetsJ1(n)` qui renvoie un dictionnaire `S1` dont les clés sont les sommets de graphe de Nim et la valeur d'un sommet s est un booléen : `True` si $s \in S_1$ et `False` si $s \in S_2$.

c) Créer une fonction `finalJ1(n)` qui renvoie le dictionnaire `F1` dont l'unique clé est le sommet de la position finale gagnante pour J_1 (et la valeur est arbitraire, par exemple 1).

2) Graphe transposé

Écrire la fonction `transpose(G)` qui prend en entrée un graphe représentant l'arène de jeu, représenté sous la forme d'un dictionnaire des successeurs et renvoie le graphe transposé.

Dans la suite de l'énoncé, on cherche à construire les positions gagnantes (attracteur) du premier joueur J_1 et les stratégies gagnantes optimales associées à ces positions.

On propose des algorithmes valables pour tout jeu à deux joueurs défini par

- le dictionnaire `G` codant le graphe du jeu : les clés sont les sommets du graphe et pour tout sommet, sa valeur est la liste de ses successeurs

- le dictionnaire `S1` définissant la partition entre l'ensemble S_1 des sommets contrôlés par le joueur J_1 et l'ensemble S_2 des sommets contrôlés par le joueur J_1 : les clés sont les sommets de graphe et la valeur d'un sommet s est un booléen : `True` si $s \in S_1$ et `False` si $s \in S_2$

- le dictionnaire `F1` définissant l'ensemble F_1 les positions finales gagnantes pour J_1 : les clés sont les positions finales gagnantes pour J_1 (et la valeur est arbitraire, par exemple 1).

Remarque : Les positions finales correspondent aux sommets sans sucesseur. On s'intéresse ici au jeu sans partie nulle, c'est-à-dire que les positions finales qui ne sont pas gagnantes pour J_1 le sont pour J_2. On suppose aussi que toute partie est finie, c'est-à-dire que le graphe est acyclique.

Remarque : On pourra utiliser dans le codage PYTHON les fonctions `max` et `min` d'une liste.

3) Calcul de l'attracteur du premier joueur

L'attracteur du joueur J_1 est l'ensemble des positions à partir desquelles le joueur J_1 admet une stratégie gagnante (lui assurant de gagner quels que soient les coups du joueur J_2).

L'algorithme de calcul de l'attracteur du joueur J_1 est le suivant :

(1) On calcule le graphe transposé, c'est-à-dire on détermine les antécédents de chaque position

(2) On crée un dictionnaire `compteur` dont les clés sont les sommets et la valeur d'un sommet s est son degré sortant, c'est-à-dire le nombre de successeurs de s dans le graphe G.

Lors de l'exécution de l'algorithme, ce dictionnaire `compteur` sera mis à jour de sorte que la valeur d'un sommet est le nombre de successeurs qui ne sont pas des positions gagnantes déjà reconnues pour le joueur J_1.

(3) On crée un attracteur vide (codé par un dictionnaire `A`)

Lors de l'exécution de l'algorithme, ce dictionnaire sera mis à jour de sorte que les clés du dictionnaire `A` sont les positions gagnantes du joueur J_1 déjà connues.

(4) On crée une pile contenant les positions finales gagnantes pour le joueur A

(5) Tant que la pile n'est pas vide :

(a) on dépile un sommet s

(b) si ce sommet est déjà dans l'attracteur A, on ne fait rien

(c) sinon, on ajoute s à l'attracteur A, et pour chaque prédécesseur t de s :

(i) on retranche 1 au compteur des successeurs non gagnants du sommet t

(ii) si t est dans S_1, il est gagnant, on ajoute donc ce sommet t à la pile

(iii) sinon, t est dans S_2 : si tous les successeurs de t sont gagnants,

c'est-à-dire si le compteur de successeurs non gagnants de t est nul,

alors t est gagnant, et on ajoute donc t à la pile.

a) Écrire une fonction `degres_sortants(G)` qui renvoie le dictionnaire `compteur` des degrés sortants des sommets dans le graphe G : les clés sont les sommets et la valeur d'un sommet s est le nombre de successeurs de s dans G.

b) Écrire une fonction `attracteur(G,S1,F1)` qui code l'algorithme décrit ci-dessus et renvoie l'attracteur A du joueur J_1 codé par un dictionnaire dont les clés sont les positions gagnantes de J_1 (et les valeurs sont arbitraires).

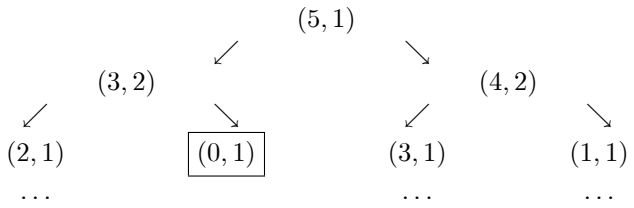
c) On considère ici le jeu de Nim décrit au 1) et on prend $n = 4$.

Préciser les positions gagnantes pour le premier joueur J_1 parmi les dix positions du jeu.

Remarque : Les positions qui ne sont pas gagnantes pour le joueur J_1 sont gagnantes pour le joueur J_2.

4) Cas particulier de l'algorithme Min-Max

On considère l'arbre du jeu de Nim pour $n = 5$ associé au sommet initial $s = (5, 1)$:



(1) Les feuilles de l'arbre correspondent à des positions finales. On attribue la valeur $v(s) = 1$ si s est une position finale gagnante pour J_1, et $v(s) = -1$ si s est une position finale gagnante pour J_2.

(2) Ensuite on définit les valeurs $v(s)$ pour les autres sommets s de manière récursive :

- si s est contrôlé par J_1, sa valeur $v(s)$ est le maximum des valeurs de ses successeurs.
- si s est contrôlé par J_2, sa valeur $v(s)$ est le minimum des valeurs de ses successeurs.

L'algorithme min-max peut donc se programmer de façon récursive.

a) Définir une fonction récursive `minmax(G,S1,F1,s)` qui étant donnés les dictionnaires `G` , `S1` et `F1`, et un sommet s, renvoie la valeur $v(s) \in \{-1, 1\}$ du sommet s.

b) Caractériser le sens de la valeur $v(s)$, c'est-à-dire une CNS pour que $v(s)$ soit égale à 1.

c) La complexité de l'algorithme proposé au a) est trop élevée (généralement exponentielle).

Proposer une nouvelle version donnant une complexité linéaire (en la taille du graphe) :

On utilisera un dictionnaire de mémoïsation `dico` et une fonction auxiliaire récursive.

Le dictionnaire `dico` stocke les couples $(s, v(s))$ déjà connus.

d) Dans le cas du jeu de Nim, il n'est pas nécessaire d'utiliser une fonction auxiliaire récursive, car vu la structure particulière du graphe, on peut déterminer les valeurs $v(s)$ des sommets $s = (k, i)$ en itérant sur k (pour k allant de 0 jusqu'à la valeur associée à s). On ajoute au fur et à mesure les couples $(s, v(s))$ dans le dictionnaire `dico`

Définir une fonction `minmaxNim(s)` qui étant une position s renvoie la valeur $v(s)$ de s dans le cas spécifique du jeu de Nim défini au 1). On définiera n par $s[0]$.

Par exemple, `minmaxNim((0,1))` renvoie `+1` et `minmaxNim((1,1))` renvoie `-1`.

5) Stratégies gagnantes optimales

On note A l'attracteur du joueur J_1, c'est-à-dire l'ensemble des positions gagnantes pour le joueur J_1.

Lorsqu'on se trouve dans une position $s \in A$, le joueur J_1 va jouer pour gagner en un **minimum** de coups alors le joueur J_2 essaie au contraire d'allonger au **maximum** la durée de la partie.

On note $m(s)$ la durée minimale des parties respectant ces règles.

On cherche donc à déterminer pour chaque position gagnante $s \in A$ la valeur de $m(s)$ et une stratégie gagnante, c'est-à-dire d'associer à ce sommet s, lorsqu'il n'est pas une position finale, le successeur (ou l'un des successeurs) permettant au joueur J_1 de gagner en $m(s)$ coups. Pour déterminer le coup optimal à jouer sur une position gagnante, il ne suffit pas de jouer sur une autre position gagnante, mais de jouer sur une position gagnante minimisant la durée de la partie.

a) Lorsque $s \in F_1$, on a $m(s) = 0$.
Pour $s \in A$ qui n'est pas finale, exprimer $m(s)$ en fonction des $m(t)$, où t sont des successeurs de s.

b) Écrire une fonction `nombres(G,S1,F1)` qui renvoie un dictionnaire `M` dont les clés sont les positions gagnantes s de J_1 (c'est-à-dire les sommets $s \in A$), et la valeur de s est l'entier $m(s)$.

c) Écrire une fonction `strategieGagnante(G,S1,F1)` qui renvoie une stratégie gagnante, c'est-à-dire un dictionnaire `Sg` de sorte que si s est une position gagnante non finale pour le joueur J_1, alors `Sg[s]` est un coup lui permettant de gagner en un minimum de temps.

On pourra utiliser le dictionnaire `M` renvoyé par `nombres(G,S1,F1)`.

d) On reprend le jeu de Nim avec $n = 4$.
Préciser ce que renvoient `nombres(G,S1,F1)` et `strategieGagnante(G,S1,F1)`.