TP N°6 : Implémentation du bassin d'attraction.

Le jeu de Nim est un jeu à information complète où deux joueurs jouent chacun leur tour pour prendre des jetons d'un tas commun. Le but est de laisser le dernier jeton pour gagner. Dans ce TP, nous allons implémenter en Python les fonctions permettant de calculer le bassin d'attraction d'un joueur pour ce jeu et l'algorithme de Minimax pour trouver le meilleur coup à jouer (TP suivant).

Le bassin d'attraction est l'ensemble des états du jeu qui sont gagnants pour un joueur donné, si ce joueur joue optimalement. L'algorithme de Minimax est utilisé pour déterminer le meilleur coup à jouer pour un joueur en utilisant l'arbre des différents états du jeu et en prenant en compte les coups possibles des joueurs suivants. Il permet d'optimiser les gains d'un joueur en maximisant les gains et minimisant les pertes potentielles.

Les fonctions pour créer le graphe et l'arbre du jeu de Nim à un tas seront données et nous allons les utiliser pour créer les exemples de jeu et les tests.

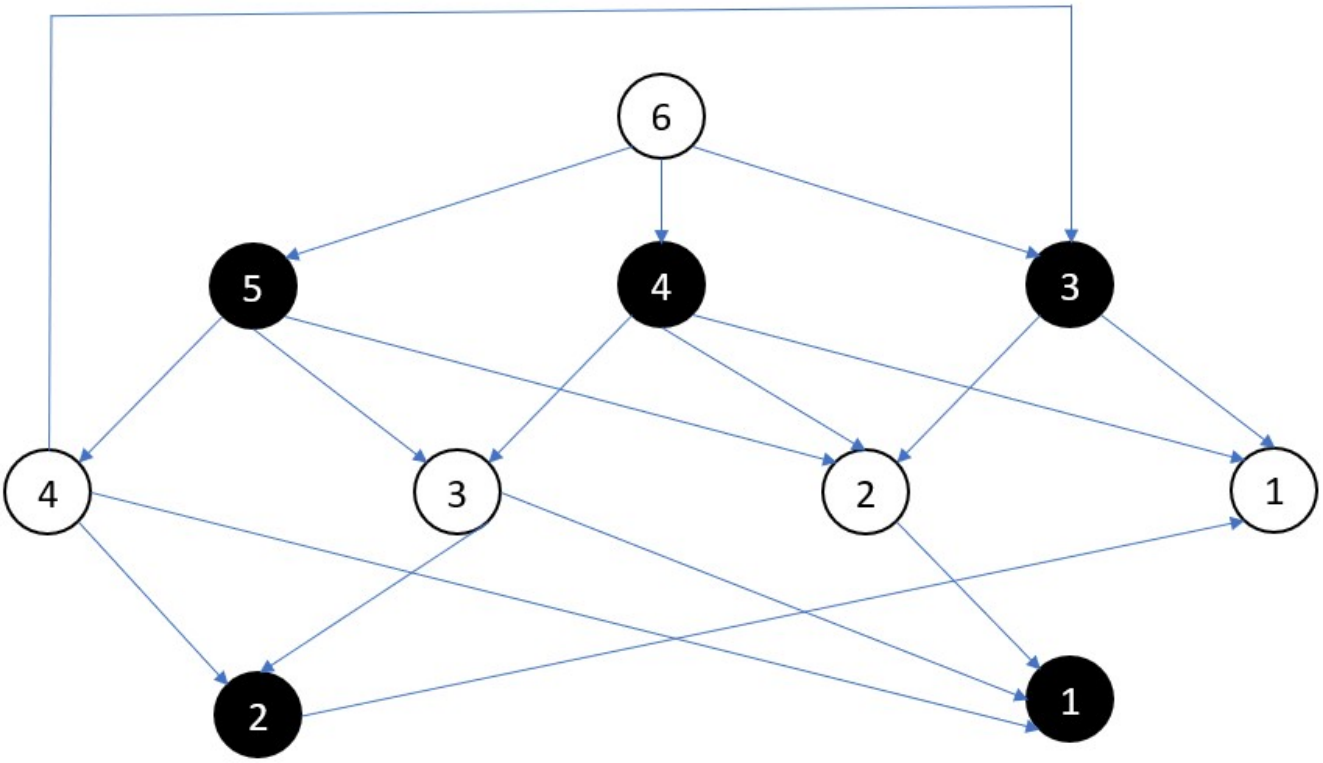
Préambule

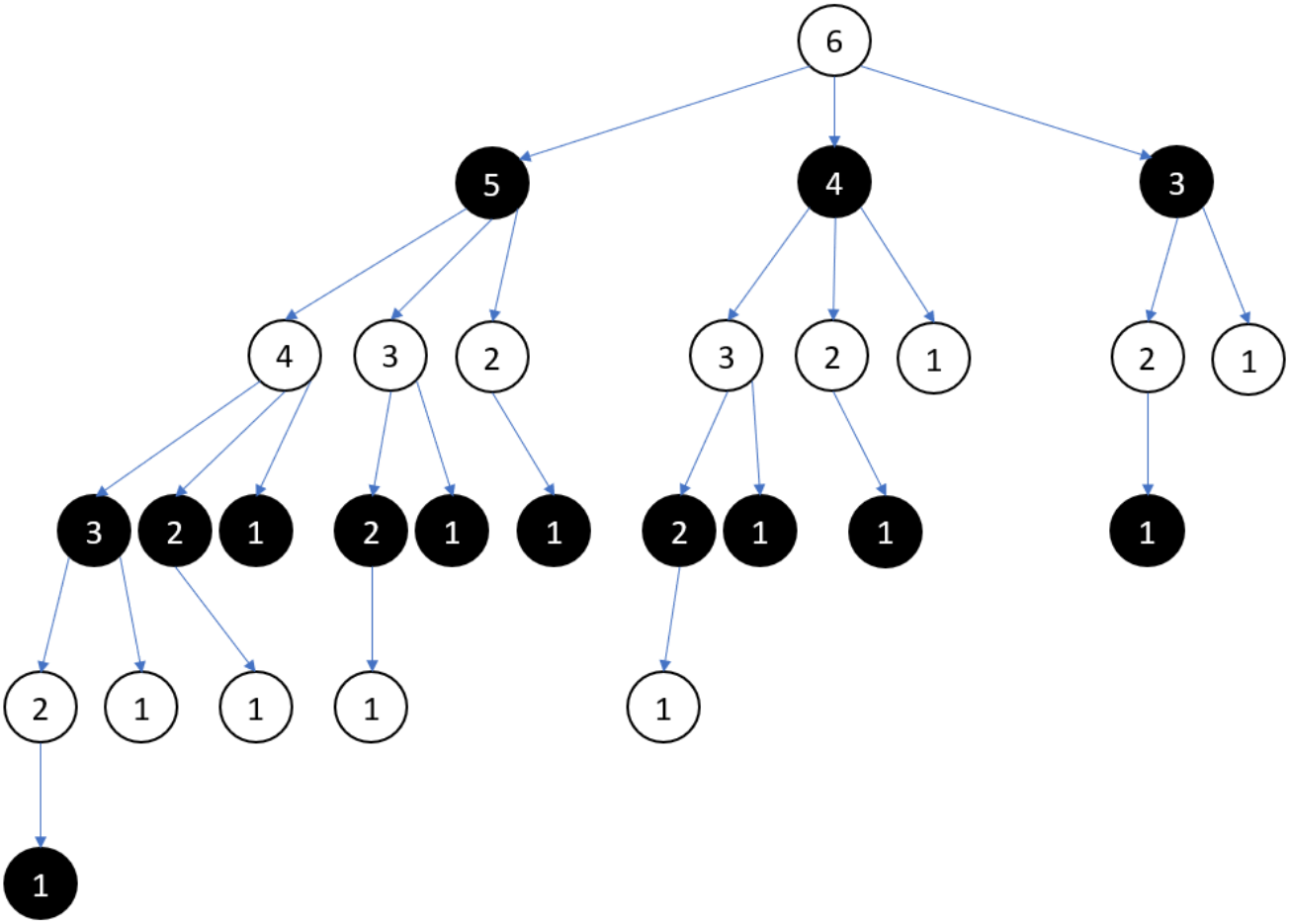
1 Récupérer le fichier TPini.py se trouvant sur cahier de prépa.

Sur ce fichier figure deux fonctions `Graphe_Nim(n)` et `Arbre_Nim(n)` qui prend en argument un entier n correspondant au nombre de bâtons et qui renvoie un triplet (S,A,E) de trois listes.

- La première liste est une liste d'entiers qui représente les sommets du graphe.
- La seconde liste est une liste de listes d'entiers qui représente les arêtes du graphe ou les fils de chaque nœud de l'arbre.
- La troisième liste est une liste de chaînes de caractères qui représente les étiquettes des sommets ou des nœuds, l'étiquette "i ;k" indique qu'il reste i bâtons et que c'est le joueur k qui doit jouer.

2 Regarder ce que renvoie les deux fonctions pour $n = 6$, on pourra ajouter les indices de S représentant les sommets du graphe ou arbre aux deux figures suivantes :


6
5
4
3
4
3
2
1
2
1


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

Bassin d'attraction

L'objectif de ce TP est de construire une fonction de calcul de bassin d'attraction qui fonctionne pour tous les graphes de jeux. Cette fonction prend en entrée un graphe de jeu et renvoie le bassin d'attraction d'un joueur donné. Le bassin d'attraction est l'ensemble des états du jeu qui sont gagnants pour ce joueur, s'il joue de manière optimale. Cette fonction doit être linéaire par rapport à la somme du nombre d'arêtes et de sommets.

3 Écrire une fonction `predsuccesseur(S,A)` qui prend en argument une liste de sommets S et d'arêtes A et qui renvoie :

- une liste suc telle que suc[i] est la liste des successeurs du sommet i ;
- une liste nbsuc telle que nbsuc[i] est le nombre de successeurs du sommet i ;
- une liste pred telle que pred[i] est la liste des prédécesseurs du sommet i ;
- une liste nbpred telle que nbpred[i] est le nombre de prédécesseurs du sommet i.

La fonction sera linéaire par rapport au nombre d'arêtes.

4 Écrire une fonction `J0_Nim(S,A,E)` qui prend en argument le triplet (S,A,E) renvoyer par la fonction graphe_nim(n) et qui renvoie dans l'ordre :

- La liste des sommets contrôlés par le joueur J_0, la variable sera nommée `S0` par la suite ;
- La liste des sommets gagnants pour le joueur J_0 , la variable sera nommée `F0` par la suite.

5 Écrire une fonction `Bassin_attraction(S,A,F0,S0)` qui prend en argument des variables sus-cités, et renvoie la liste des sommets dans le bassin d'attraction du joueur J_0. On rappelle qu'on détermine ce bassin de manière itérative en rajoutant dans la liste des sommets déjà connus :

- Les sommets contrôlés par J_0 dont une arête pointe vers un sommet du bassin déjà connu
- Les sommets contrôlés par J_1 dont toutes les arêtes pointent vers des sommets du bassin déjà connus.

et ce, tant qu'on trouve de nouveaux sommets à rajouter dans le bassin entre chaque itération. On initialise le bassin d'attraction à F_0.

Pour cela, on va définir une fonction `m_et_p(s,attr,compt,pred,S0)` qui prend en argument un entier s représentant un sommet du bassin d'attraction, attr une liste d'entiers contenant les sommets déjà dans le bassin d'attraction, une liste d'entier compt qui contient un compteur pour chaque sommet, et une liste pred tel que pred[u] est la liste des prédécesseurs de u.

- La fonction rajoutera s à attr s'il ne l'est pas
- Pour chaque u prédécesseur de s :
  - on baissera le compteur associé à u de 1
  - Si u est dans S0 ou son compteur est arrivé à 0, on lance `m_et_p(u,attr,compt,pred,S0)`

La fonction bassin d'attraction initialisera attr à [], compt[i] au nombre de successeurs du sommet i, pred[i] à sa liste des prédécesseurs. et lancera la fonction `m_et_p` sur tous les points de F_0.

Le principe est de partir des points de F_0, élément trivial du bassin d'attraction et pour chaque arête pointant vers ces éléments :

- l'origine est dans S_0 donc contrôlés par le joueur J_0 : c'est un nouveau élément du bassin, récursivement on regarde les arrêtes pointant sur lui ;
- l'origine n'est pas dans S_0 mais son compteur est arrivé à 0 : alors ces successeurs sont dans le bassin d'attraction, c'est un élément du bassin d'attraction, récursivement on regarde les arrêtes pointant sur lui.

6 Écrire une fonction `strat(S,A,E,p)` qui donne le successeur de p qu'il faut choisir pour suivre une stratégie gagnante, on procédera de la manière suivante :

- Si un des successeur de p est dans le bassin d'attraction, on choisit un point du bassin au hasard.
- Si aucun d'entre eux l'est, on choisit un successeur au hasard.

On peut se rendre compte qu'il y a sans doute mieux à faire que de choisir un point au hasard, surtout si le jeu possède un gain en cas de victoire, on voudrait alors tenter de maximiser ce gain, ou de minimiser des pertes. C'est le but de l'algorithme du minimax, que nous verrons au TP suivant.