Jeu sur un graphe

Dans ce problème, on considère un graphe non orienté G à n sommets, numérotés de 0 à $n-1$.
Le jeu est un jeu à deux joueurs : le joueur J_1 le joueur J_2, qui doivent poser des jetons sur les sommets du graphe G en respectant les règles suivantes :

- au début de la partie, tous les sommets sont libres ;
- c'est le joueur J_1 commence la partie ;
- lors de son tour, un joueur doit choisir un sommet vide de G et poser un jeton avec son numéro dessus en respectant la contrainte notée (C) suivante :
  (C) : « deux sommets adjacents (ie reliés par une arrête du graphe) non vides ne doivent pas être occupés par des jetons de même numéro. »
- si le joueur courant à un coup possible, il le joue et on passe au tour de l'autre joueur ;
- sinon (il n'y a plus de coup possible respectant(C)), le joueur courant a perdu la partie.

Dans la suite, on appelle **état** un graphe dont certains des sommets sont occupés par des jetons, **état terminal** un état où le joueur courant ne peut plus jouer, et **position gagnante** un état à partir duquel il existe une stratégie gagnante pour le joueur courant.

I Généralités

1. Est-il possible d'avoir une partie qui ne termine pas en un nombre fini de coups ? Si oui, donner un exemple d'une telle partie ; sinon, donner un majorant du nombre de coups d'une partie en fonction du nombre de sommets n du graphe G.

2. Est-il possible d'avoir match nul dans ce jeu ? Justifier brièvement.

La figure 1 ci-dessous représente un exemple de graphe à 11 sommets, et un état du jeu pour une partie en cours. Les sommets dont les numéros comportent un indice correspondent à des sommets occupés par des jetons : les sommets 0_1 et 8_1 sont donc occupés par des jetons numérotés 1, le sommet 3_2 est occupé par un jeton 2. Les sommets sans indices $(2, 4, 5, \dots)$ sont libres.

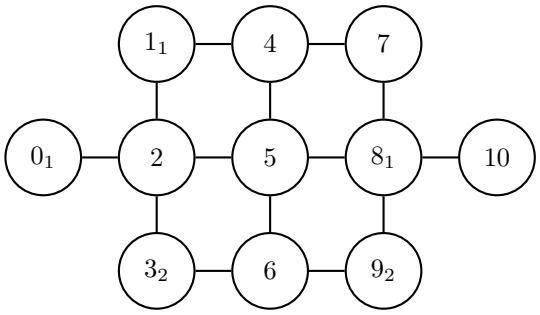

fig 1 : un état du jeu

C'est au tour de J_2 de jouer. Pour un état du jeu donné, un **coup** est le numéro du sommet sur lequel le joueur courant choisit de placer un jeton.

3. Donner (en justifiant rapidement vos réponses) successivement, à partir de l'état de la figure 1 :

a) la liste `valides` des coups valides pour J_2, c'est-à-dire respectant la contrainte (C).

b) la liste `gains` des coups de `valides` conduisant à un état terminal gagnant pour J_2.

c) la liste `posG` des coups de `valides` conduisant à une position gagnante pour J_2.

4. On considère dans cette question uniquement un graphe G réduit à un unique cycle comportant $n \geqslant 4$ sommets avec n pair, comme illustré ci-dessous (les sommets sont numérotés en tournant autour du cercle dans le sens trigonométrique). Justifier qu'une stratégie gagnante pour J_2 consiste à jouer systématiquement sur le sommet d'indice juste après celui sur lequel J_1 vient de jouer : si J_1 vient de poser son jeton sur le sommet $k \leqslant n-2$, respectivement en $n-1$, alors J_2 pose le sien en $k+1$, respectivement en 0.

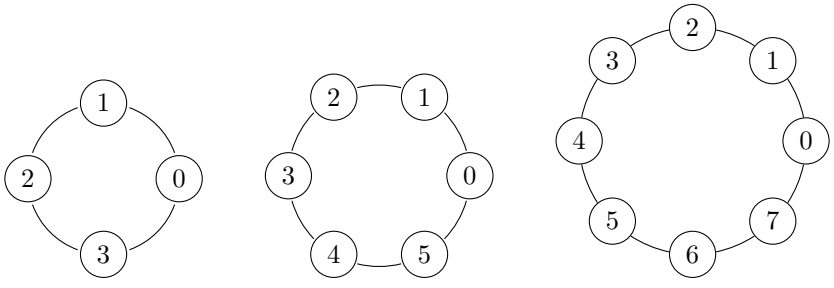

fig 2 : Trois cycles comportant respectivement 4, 6 et 8 sommets.

5. On considère dans cette question uniquement un « graphe étoilé » G comportant $n \geqslant 4$ sommets, comme illustré ci-dessous. Donner en fonction de n le joueur possédant une stratégie gagnante et la décrire brièvement.

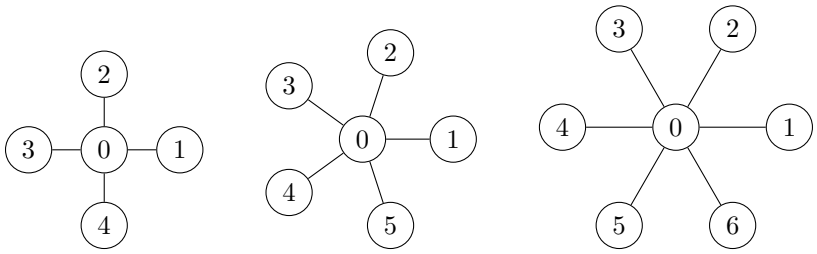

fig 3 : Trois étoiles comportant respectivement 5, 6 et 7 sommets.

II Implémentation du jeu

On se propose de représenter un graphe G de taille n par son dictionnaire d'adjacence, plus précisément par un dictionnaire `G` dont les clefs sont les numéros des sommets $s \in [\![0, n-1]\!]$ et `G[s]` est la liste des sommets adjacents à `s`.

1. Écrire une fonction `creerGraphe(n:int)->dict` prenant en argument un entier n et renvoyant le dictionnaire d'adjacence d'un graphe G ayant n sommets et aucune arête pour le moment.

2. Écrire une fonction `ajouterArete(G:dict,i:int,j:int)->None` prenant en arguments le dictionnaire d'adjacence d'un graphe non orienté G et deux entiers distincts i et j correspondant à des sommets de G, et modifie `G` pour lui ajouter l'arête $i - j$.

3. Si un graphe non orienté G possède n sommets, quel est le nombre maximal p d'arêtes que peut posséder G ?

4. On suppose avoir importé la fonction `randint` du module `random` par la commande

```
from random import randint
```

L'aide fournie par Python sur cette fonction renvoie le message suivant :

```
>>> help(randint)
Help on method randint in module random :

randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both end points.
```

En déduire une fonction `plateau(n:int,p:int)->dict` qui prend en argument le nombre n de sommet et le nombre p d'arêtes du graphe et qui renvoie le dictionnaire d'adjacence du graphe non orienté G, les arêtes étant obtenue de façon aléatoires. On supposera que n et p satisfont la condition de la question précédente (et qu'il est donc possible de créer un tel graphe).

Pour représenter un état du jeu, on utilisera un dictionnaire `etat` dont les clefs sont les sommets occupés par des jetons et si s est un sommet occupé par un jeton alors `etat[s]` est le numéro du jeton placé sur le sommet `s`.
Par exemple le dictionnaire `etat` associé à l'état représenté à la figure 1 est

```
etat = {0:1,1:1,3:2,8:1,9:2}
```

5. Écrire une fonction `joueur(etat:dict)->int` qui prend en argument un dictionnaire représentant un état du jeu et qui renvoie le numéro (1 ou 2) du joueur qui doit jouer le prochain coup.

6. Écrire une fonction `test(G:dict,s:int,etat:dict,j:int)->bool` prenant en arguments le dictionnaire d'adjacence d'un graphe non orienté, un numéro d'un sommet s, un dictionnaire `etat` représentant un état du jeu et un entier $j \in \{1, 2\}$ et renvoyant `True` si le sommet `s` ne possède aucun voisin occupé par un jeton du joueur `j`, et `False` sinon.

7. Écrire une fonction `possibles(G:dict,etat:dict)->list` prenant en argument le dictionnaire d'adjacence du graphe G, un dictionnaire représentant un état du jeu et qui renvoie la liste des coups possibles (c'est-à-dire respectant la contrainte (C)) pour le joueur courant.

8. Déterminer la complexité de votre fonction `possibles` dans le pire cas, en fonction du nombre de sommets n et/ou du nombre d'arêtes p du graphe G.

9. Comment peut-on déterminer si le dictionnaire `etat` représente un état terminal ?

III Recherche de stratégies gagnantes

On va chercher dans cette partie à définir une stratégie gagnante pour le joueur J_2.

1. Pour commencer, on se contente d'utiliser une stratégie similaire à celle utilisée dans le cas d'un graphe circulaire (cf **I.4**) : si J_1 vient de jouer sur un sommet s, on va chercher le sommet, sur lequel J_2 peut jouer, le plus proche de s ; la distance entre deux sommets étant le nombre minimal d'arêtes à emprunter pour les rejoindre.

a) Pourquoi un parcours de graphe en largeur est-il plus adapté à cette recherche qu'un parcours en profondeur ?

b) Compléter, en rajoutant autant de lignes que nécessaire aux deux endroits précisés, la fonction suivante de sorte qu'elle renvoie le numéro d'un sommet sur lequel J_2 peut jouer en suivant la stratégie précédente ; la fonction `suivant(G:dict,etat:dict,s:int)->int` prend en argument le dictionnaire d'adjacence du graphe G, un dictionnaire représentant un état du jeu et le numéro du sommet s sur lequel J_1 vient de jouer.

```
def suivant(G,etat,s) :
    dicoPossibles = {x : False for x in G}
    for x in possibles(G,etat) :
        dicoPossibles[x] = True
    Q = [s]
    vus = {x : False for x in G}
    while        # à compléter
        s = Q[0]
        Q = Q[1:]
        vus[s] = True
        for voisin in G[s] :
                # à compléter
    return s
```

c) Dans la fonction précédente, quel est l'intérêt d'utiliser un dictionnaire `dicoPossibles` plutôt que la liste renvoyée par la fonction `possibles` ?

d) Cette façon de jouer permet-elle au joueur J_2 d'avoir une stratégie gagnante sur n'importe quel graphe G ?

On souhaite maintenant identifier si une position donnée du jeu est gagnante pour le joueur J_2. Pour cela, on se propose d'écrire une fonction récursive nommée `gagnantJ2(G:dict,etat:dict)->bool`, reposant sur l'algorithme MinMax, prenant en argument le dictionnaire d'adjacence du graphe G, un dictionnaire représentant un état du jeu dont le principe est le suivant :

- si l'état actuel est un état terminal, la fonction renvoie `True` si J_2 gagne et `False` sinon ;
- sinon, on appelle cette fonction récursivement sur tous les coups possibles du joueur courant :
  - ◇ si le joueur courant est J_2 et qu'au moins l'un des coups possibles amène à une position gagnante pour J_2 alors la position était gagnante pour J_2.
  - ◇ si le joueur courant est J_1 et qu'au moins l'un des coups possibles amène à une position gagnante pour J_1 alors la position n'était pas gagnante pour J_2.

2. Implémenter la fonction récursive `gagnantJ2(G,etat)` décrite ci-dessus.

Si besoin, on rappelle que ***`del(etat[s])`*** *supprime la clef* ***`s`*** *du dictionnaire* ***`etat`*** *et la valeur qui lui est associée, à condition que* ***`s`*** *soit effectivement une clef de ce dictionnaire.*

3. Donner une majoration de la complexité de votre fonction en fonction du nombre de sommets n de G.

Afin d'améliorer la rapidité de la fonction précédente, on souhaite utiliser la méthode de mémöisation : on va utiliser un dictionnaire `PG2` dont les clefs correspondent aux états du jeu et la valeur associée est `True`, respectivement `False`, si l'état correspond à une position gagnante pour J_2, respectivement pour J_1.

4. Pourquoi ne peut-on pas utiliser le dictionnaire `etat` comme clef de `PG2` ?

5. Encodage du dictionnaire `etat` :
À un dictionnaire `etat`, on associe l'entier

$$f(\texttt{etat}) = \sum_{c \in \texttt{etat}} \texttt{etat}[c] \times 3^c$$

Le dictionnaire associé à l'état représenté à la figure 1 est pas exemple

```
etat = {0:1,1:1,3:2,8:1,9:2}
```

il est donc représenté par l'entier

$$f(\texttt{etat}) = 1 \times 3^0 + 1 \times 3^1 + 2 \times 3^3 + 1 \times 3^8 + 2 \times 3^9 = 45985$$

a) Écrire une fonction `encode(etat:dict)->int` qui prend en argument un dictionnaire représentant un état du jeu et qui renvoie l'entier qui lui est associé par la fonction f précédente.

b) Écrire une fonction `decode(k:int)` qui prend en argument un entier k et qui renvoie le dictionnaire correspondant : c'est-à-dire le dictionnaire `etat` tel que $f(\texttt{etat}) = k$

6. Dans la suite, on considère une variable globale `PG2`, initialisée par

```
PG2 = {}
```

Modifier votre fonction `gagnantJ2(G,etat)` en utilisant le dictionnaire `PG2` et la technique de mémoïsation.

7. On suppose que $f(\texttt{etat})$ est effectivement une clef du dictionnaire `PG2` dont la valeur est `True`, l'état correspondant correspond donc à une position gagnante pour J_2. Le joueur J_2 possède donc une stratégie gagnante depuis cet état.

Écrire une fonction `jouerJ2(G:dict,etat:dict)->int` qui prend en argument le dictionnaire d'adjacence du graphe G, un dictionnaire représentant un état du jeu et qui renvoie le numéro d'un sommet de G sur lequel J_2 doit placer son jeton pour suivre sa stratégie gagnante. Cette fonction pourra utiliser le dictionnaire `PG2` défini comme une variable globale.

8. Pourquoi serait-il intéressant de mettre en place une heuristique pour améliorer la recherche d'une stratégie gagnante ?

9. Proposer un moyen de définir une telle heuristique en expliquant vos choix. *Aucun code n'est demandé dans cette question.*