DEVOIR MAISON

À rendre le Jeudi 20 février 2025

I. Calcul des attracteurs et des positions gagnantes

1) Un graphe (S, A) est **biparti** lorsque l'ensemble S des sommets est la réunion disjointe de deux sous-ensembles S_1 et S_2 et que chaque arête a une extrémité dans S_1 et l'autre dans S_2 (il n'existe pas d'arête reliant deux sommets de S_1 ou deux sommets de S_2).

2) Pour le joueur 1 :

- $\star$ $\mathcal{A}_1^0 = \{(2)\}$
- $\star$ $\mathcal{A}_1^1 = \mathcal{A}_1^0 \cup \{[3]\}$
- $\star$ $\mathcal{A}_1^2 = \mathcal{A}_1^1 \cup \{(1);(6)\}$
- $\star$ $\mathcal{A}_1^3 = \mathcal{A}_1^2 \cup \{[1];[5]\}$
- $\star$ $\forall n \geqslant 4, \quad \mathcal{A}_1^n = \mathcal{A}_1^3$

donc $\mathcal{A}_1 = \{(2);[3];(1);(6);[1];[5]\}$.

Pour le joueur 2 :

- $\star$ $\mathcal{A}_2^0 = \{[4]\}$
- $\star$ $\mathcal{A}_2^1 = \mathcal{A}_2^0 \cup \{(4)\}$
- $\star$ $\mathcal{A}_2^2 = \mathcal{A}_2^1 \cup \{[2]\}$
- $\star$ $\mathcal{A}_2^3 = \mathcal{A}_2^2 \cup \{(3);(5)\}$
- $\star$ $\forall n \geqslant 4, \quad \mathcal{A}_2^n = \mathcal{A}_2^3$

donc $\mathcal{A}_2 = \{[4];(4);[2];(3);(5)\}$.

3) Les positions gagnantes pour le joueur 1 sont éléments de l'attracteur $\mathcal{A}_1$:

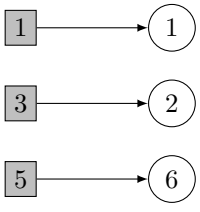
$$\mathcal{A}_1 = \{(2);[3];(1);(6);[1];[5]\}$$

De même, les positions gagnantes pour le joueur 2 sont éléments de l'attracteur $\mathcal{A}_2$:

$$\mathcal{A}_2 = \{[4];(4);[2];(3);(5)\}$$

Il n'y a pas d'autre positions que celles dans $\mathcal{A}_1$ et $\mathcal{A}_2$ donc il n'y a pas de positions nulles.

4) Pour définir une stratégie gagnante, il faut rester dans l'attracteur $\mathcal{A}_1$. On peut donc définir la stratégie suivante :



5)
```python
def est_biparti(G):
    A, S1, S2 = G

    for a in A:
        if a[0] in S1 and a[1] in S1:
            return False
        if a[0] in S2 and a[1] in S2:
            return False

    return True
```

6)
```python
def dictionnaire_adjacence(G):
    A, S1, S2 = G
    S = S1 + S2

    D = {}
    for s in S:
        D[s] = []

    for a in A:
        D[a[0]].append(a[1])

    return D
```

7)
```python
def positions_finales_gagnantes(G, i):
    P = []
    D = dictionnaire_adjacence(G)

    for s in G[3-i]:
        if D[s] == []:
            P.append(s)

    return P
```

8)
```python
def attrateur(G, i, n):
    if n == 0:
        return positions_finales_gagnantes(G, i)

    A = attrateur(G, i, n-1)
    D = dictionnaire_adjacence(G)
    N = []

    if n % 2 == 0:
        for s in G[3-i]:
            cpt = 0
            for t in D[s]:
                if t in A:
                    cpt += 1
            if cpt == len(D[s]) and s not in A:
                N.append(s)
    else:
        for s in G[i]:
            cpt = 0
            for t in A:
                if (s,t) in G[0]:
                    cpt += 1
            if cpt > 0 and s not in A:
                N.append(s)

    return A + N
```

9)
```
def positions_gagnantes(G, i):
    D = dictionnaire_adjacence(G)
    A = []
    N = positions_finales_gagnantes(G, i)
    k = 0

    while N != []:
        A = A + N
        N = []

        if k % 2 == 0:
            for s in G[i]:
                cpt = 0
                for t in A:
                    if (s,t) in G[0]:
                        cpt += 1
                if cpt > 0 and s not in A:
                    N.append(s)
        else:
            for s in G[3-i]:
                cpt = 0
                for t in D[s]:
                    if t in A:
                        cpt += 1
                if cpt == len(D[s]) and s not in A:
                    N.append(s)

        k += 1

    return A
```

II. Algorithme du minmax

10)
```
def coups_possibles(position):
    C = []

    for i in range(len(position)):
        for j in range(1, position[i] + 1):
            C.append((i, j))

    return C
```

11)
```
def jouer_coup(position, coup):
    (i, j) = coup
    position[i] -= j
```

12)
```
def annuler_coup(position, coup):
    (i, j) = coup
    position[i] += j
```

13)
```
from random import choice

def choix_maxi(scores_coups):
    M = scores_coups[0][0]
    L = [scores_coups[0]]

    for i in range(1, len(scores_coups)):
        if scores_coups[i][0] > M:
            M = scores_coups[i][0]
            L = [scores_coups[i]]
        elif scores_coups[i][0] == M:
            L.append(scores_coups[i])

    return choice(L)
```

14)
```
def choix_mini(scores_coups):
    M = scores_coups[0][0]
    L = [scores_coups[0]]

    for i in range(1, len(scores_coups)):
        if scores_coups[i][0] < M:
            M = scores_coups[i][0]
            L = [scores_coups[i]]
        elif scores_coups[i][0] == M:
            L.append(scores_coups[i])

    return choice(L)
```

15)
```
from math import inf

def minmax(position, joueur):
    if sum(position) == 0:
        if joueur == 1:
            return (-inf, None)
        else:
            return (+inf, None)

    L = coups_possibles(position)
    scores_coups = []
    for coup in L:
        jouer_coup(position, coup)
        score = minmax(position, 3-joueur)[0]
        annuler_coup(position, coup)
        scores_coups.append((score, coup))

    if joueur == 1:
        return choix_maxi(scores_coups)
    else:
        return choix_mini(scores_coups)

D = {}

def minmax_memo(position, joueur):
    code = tuple(position + [joueur])

    if code in D:
        return D[code]

    if sum(position) == 0:
        if joueur == 1:
            res = (-inf, None)
        else:
            res = (+inf, None)
    else:
        L = coups_possibles(position)
        scores_coups = []
        for coup in L:
            jouer_coup(position, coup)
            score = minmax_memo(position, 3-joueur)[0]
            annuler_coup(position, coup)
            scores_coups.append((score, coup))

        if joueur == 1:
            res = choix_maxi(scores_coups)
        else:
            res = choix_mini(scores_coups)

    D[code] = res
    return res
```