
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 :

$$\star \mathcal{A}_1^0 = \{ \textcircled{2} \}$$

$$\star \mathcal{A}_1^1 = \mathcal{A}_1^0 \cup \{ \boxed{3} \}$$

$$\star \mathcal{A}_1^2 = \mathcal{A}_1^1 \cup \{ \textcircled{1}; \textcircled{6} \}$$

$$\star \mathcal{A}_1^3 = \mathcal{A}_1^2 \cup \{ \boxed{1}; \boxed{5} \}$$

$$\star \forall n \geq 4, \mathcal{A}_1^n = \mathcal{A}_1^3$$

$$\text{donc } \mathcal{A}_1 = \{ \textcircled{2}; \boxed{3}; \textcircled{1}; \textcircled{6}; \boxed{1}; \boxed{5} \}.$$

Pour le joueur 2 :

$$\star \mathcal{A}_2^0 = \{ \boxed{4} \}$$

$$\star \mathcal{A}_2^1 = \mathcal{A}_2^0 \cup \{ \textcircled{4} \}$$

$$\star \mathcal{A}_2^2 = \mathcal{A}_2^1 \cup \{ \boxed{2} \}$$

$$\star \mathcal{A}_2^3 = \mathcal{A}_2^2 \cup \{ \textcircled{3}; \textcircled{5} \}$$

$$\star \forall n \geq 4, \mathcal{A}_2^n = \mathcal{A}_2^3$$

$$\text{donc } \mathcal{A}_2 = \{ \boxed{4}; \textcircled{4}; \boxed{2}; \textcircled{3}; \textcircled{5} \}.$$

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

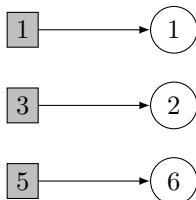
$$\mathcal{A}_1 = \{ \textcircled{2}; \boxed{3}; \textcircled{1}; \textcircled{6}; \boxed{1}; \boxed{5} \}$$

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

$$\mathcal{A}_2 = \{ \boxed{4}; \textcircled{4}; \boxed{2}; \textcircled{3}; \textcircled{5} \}$$

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) 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) def dictionnaire_adjacence(G):
2   A, S1, S2 = G
3   S = S1 + S2
4
5   D = {}
6   for s in S:
7       D[s] = []
8
9   for a in A:
10      D[a[0]].append(a[1])
11
12  return D

```

```

7) def positions_finales_gagnantes(G, i):
2   P = []
3   D = dictionnaire_adjacence(G)
4
5   for s in G[3-i]:
6       if D[s] == []:
7           P.append(s)
8
9   return P

```

```

8) def attrateur(G, i, n):
2   if n == 0:
3       return positions_finales_gagnantes(G, i)
4
5   A = attrateur(G, i, n-1)
6   D = dictionnaire_adjacence(G)
7   N = []
8
9   if n % 2 == 0:
10      for s in G[3-i]:
11          cpt = 0
12          for t in D[s]:
13              if t in A:
14                  cpt += 1
15              if cpt == len(D[s]) and s not in A:
16                  N.append(s)
17   else:
18      for s in G[i]:
19          cpt = 0
20          for t in A:
21              if (s,t) in G[0]:
22                  cpt += 1
23              if cpt > 0 and s not in A:
24                  N.append(s)
25
26  return A + N

```

```

9) def positions_gagnantes(G, i):
2   D = dictionnaire_adjacence(G)
3   A = []
4   N = positions_finales_gagnantes(G, i)
5   k = 0
6
7   while N != []:
8       A = A + N
9       N = []
10
11      if k % 2 == 0:
12          for s in G[i]:
13              cpt = 0
14              for t in A:
15                  if (s,t) in G[0]:
16                      cpt += 1
17              if cpt > 0 and s not in A:
18                  N.append(s)
19      else:
20          for s in G[3-i]:
21              cpt = 0
22              for t in D[s]:
23                  if t in A:
24                      cpt += 1
25              if cpt == len(D[s]) and s not in A:
26                  N.append(s)
27
28      k += 1
29
30  return A

```

II. Algorithme du minmax

```

10) def coups_possibles(position):
2   C = []
3
4   for i in range(len(position)):
5       for j in range(1, position[i] + 1):
6           C.append((i, j))
7
8   return C

```

```

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

```

```

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

```

```

13) from random import choice
2
3 def choix_maxi(scores_coups):
4   M = scores_coups[0][0]
5   L = [scores_coups[0]]
6
7   for i in range(1, len(scores_coups)):
8       if scores_coups[i][0] > M:
9           M = scores_coups[i][0]
10          L = [scores_coups[i]]
11      elif scores_coups[i][0] == M:
12          L.append(scores_coups[i])
13
14  return choice(L)

```

```

14) def choix_mini(scores_coups):
2     M = scores_coups[0][0]
3     L = [scores_coups[0]]
4
5     for i in range(1, len(scores_coups)):
6         if scores_coups[i][0] < M:
7             M = scores_coups[i][0]
8             L = [scores_coups[i]]
9         elif scores_coups[i][0] == M:
10            L.append(scores_coups[i])
11
12    return choice(L)

```

```

15) from math import inf
2
3 def minmax(position, joueur):
4     if sum(position) == 0:
5         if joueur == 1:
6             return (-inf, None)
7         else:
8             return (+inf, None)
9
10    L = coups_possibles(position)
11    scores_coups = []
12    for coup in L:
13        jouer_coup(position, coup)
14        score = minmax(position, 3-joueur)[0]
15        annuler_coup(position, coup)
16        scores_coups.append((score, coup))
17
18    if joueur == 1:
19        return choix_maxi(scores_coups)
20    else:
21        return choix_mini(scores_coups)
22
23 D = {}
24
25 def minmax_memo(position, joueur):
26     code = tuple(position + [joueur])
27
28     if code in D:
29         return D[code]
30
31     if sum(position) == 0:
32         if joueur == 1:
33             res = (-inf, None)
34         else:
35             res = (+inf, None)
36     else:
37         L = coups_possibles(position)
38         scores_coups = []
39         for coup in L:
40             jouer_coup(position, coup)
41             score = minmax_memo(position, 3-joueur)[0]
42             annuler_coup(position, coup)
43             scores_coups.append((score, coup))
44
45         if joueur == 1:
46             res = choix_maxi(scores_coups)
47         else:
48             res = choix_mini(scores_coups)
49
50    D[code] = res
51    return res

```