TIPE | 2023

Comment développer une stratégie gagnante au jeu du Pic'arêtes ?

/01

SOMMAIRE

— Jeux combinatoires

Définition

- 2 joueurs (jouent alternativement)
- nombre fini de configurations possibles
- configuration de départ précise
- information totale pour les 2 joueurs
- pas de hasard
- vainqueur = dernier à jouer
- pas de match nul

Image par Elmer L. Geissler de Pixabay.

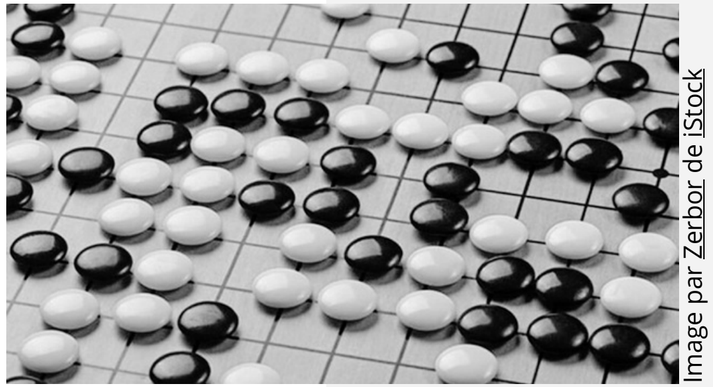
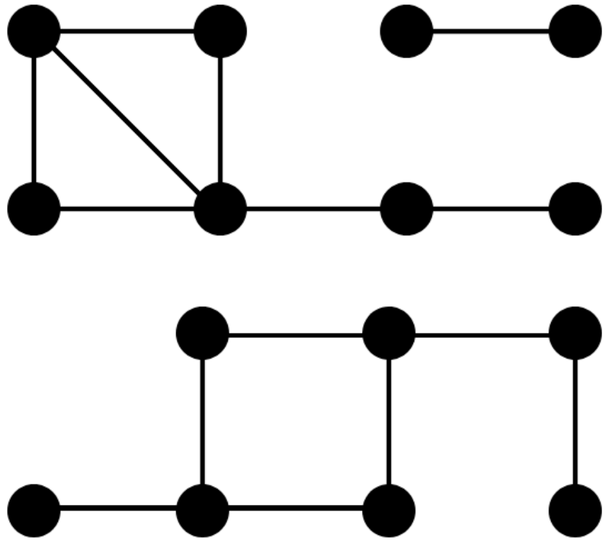


Image par Zerbor de iStock

— Jeu du Pic'arêtes

Basé sur la thèse d'Eric Duchêne

- Jeu combinatoire
- Règles
- Problème



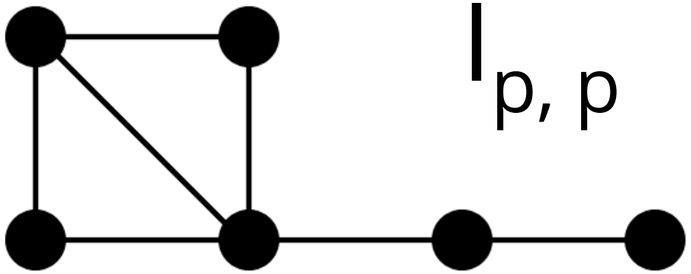
Classification des graphes connexes :

Parité du nombre d'arêtes, {I,P}

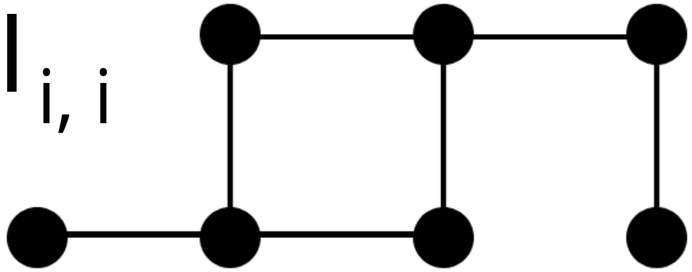
Parité du nombre de sommets non isolés, {i,p}

$A_{s,\ t}$

Parité du nombre d'arêtes appartenant à des arbres pendants, {i,p,∅}



$I_{p,\ p}$



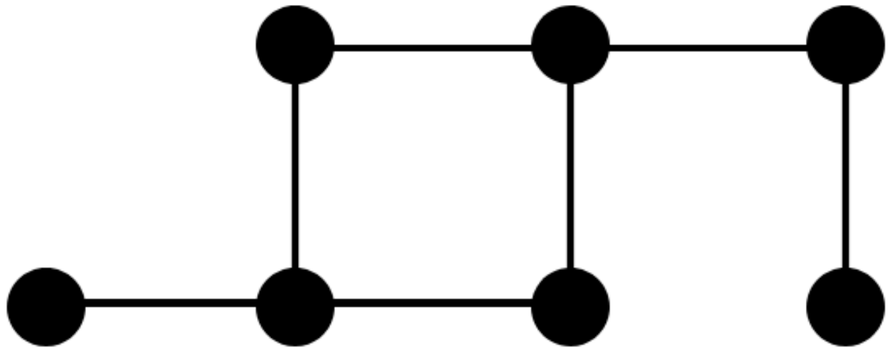
$I_{i,\ i}$

Sommet pendant :

Soit A un sommet d'un graphe non orienté.

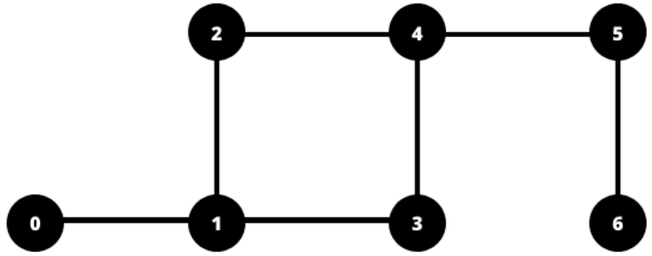
A est dit pendant si:

- il n'a qu'un voisin
- il a au maximum un voisin non pendant



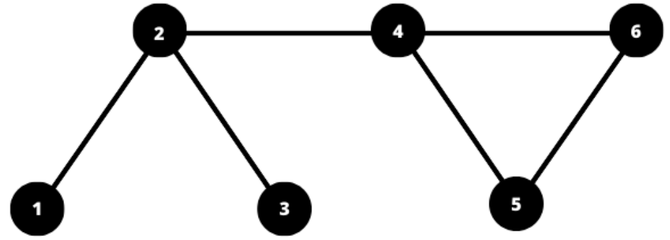
```
Fonction PENDANT g =
    On crée le tableau etat
    Fonction VISITE x =
        On ouvre x
        Si x a un voisin
            etat.x -> pendant
            VISITE (unique voisin de x)
        Sinon
            Pour tout y voisin de x
                VISITE y
            Si plus de 1 voisin est pendant
                etat.x -> lié
            Sinon
                etat.x -> pendant
    VISITE (premier sommet de g)
```

```
1   type graphe = int list array   (* liste d'adjacence *)
2   type etat = Vierge | Ouvert | Lie | Pendant
3
    graphe -> etat array
4   let pendant (g : graphe) =
5     let etat = Array.make (Array.length g) Vierge in   (* crée un tableau où tous les sommets sont vierges *)
6
7     let rec visite x pere =    (*determine si le sommet x en entrée fait partie d'un arbre pendant*)
8       etat.(x) <- Ouvert;
9       if List.length g.(x) = 1 then   (* si x n'a qu'un voisin *)
10        begin
11          etat.(x) <- Pendant;
12          visite (List.hd g.(x)) x  (* on visite le voisin de x *)
13        end
14      else
15        begin
16          List.iter (fun voisin ->  if etat.(voisin) = Vierge then visite voisin x) g.(x);
17          let nb_lie = List.fold_left (fun acc voisin -> acc + if etat.(voisin) = Lie then 1 else 0) 0 g.(x) in
18          let acyclique = List.fold_left
19            (fun acc voisin -> if voisin <> pere && etat.(voisin) = Ouvert then false else acc) true g.(x) in
20            (* si il y a un cycle, alors x n'est pas pendant *)
21          if nb_lie = 0 && acyclique
22            then etat.(x) <- Pendant
23            else etat.(x) <- Lie
24        end
25      in
26
27    visite 0 (-1);
28    etat    (*renvoie le tableau *)
```



etat array = [|Pendant; Lie; Lie; Lie; Lie; Pendant; Pendant|]

etat array = [|Pendant; Pendant; Pendant; Lie; Pendant; Pendant; Pendant|]



etat array = [|Pendant; Lie; Pendant; Lie; Lie; Lie|]

etat array = [|Pendant; Pendant; Pendant; Pendant; Pendant; Lie|]

CONCLUSION

- Quelles solutions ?

- Projet pour l'année prochaine

BIBLIOGRAPHIE

Nature	Auteurs	Titre	Compléments
Ouvrage	Eric Duchêne	Jeux combinatoires sur les graphes.	Thèse - Université Joseph-Fourier - Grenoble I, 2006.
Ouvrage	Eric Duchêne et Aline Parreau	Jeux combinatoires	https://perso.liris.cnrs.fr/eric.duchene/articles/coursCGT.pdf
Vidéo	JAlgs	Sprague-Grundy-Theorem (Game Theory part 2)	https://www.youtube.com/watch?v=GRlGknQEOW8&ab_channel=JAlgs
Ouvrage	Alain Busser	Jeux et graphes : la théorie des graphes de 5 à 95 ans	Editions Ellipses, 2020 ISBN : 9782340041035

```
1   type graphe = int list array   (* liste d'adjacence *)
2   type etat = Vierge | Ouvert | Lie | Pendant
3
    graphe -> etat array
4   let pendant (g : graphe) =
5     let etat = Array.make (Array.length g) Vierge in   (* crée un tableau où tous les sommets sont vierges *)
6
7
8     let rec visite x pere =    (*determine si le sommet x en entrée fait partie d'un arbre pendant*)
9       etat.(x) <- Ouvert;
10
11      if List.length g.(x) = 1 then   (* si x n'a qu'un voisin *)
12        begin
13          etat.(x) <- Pendant;
14          visite (List.hd g.(x)) x  (* on visite le voisin de x *)
15        end
16
17      else
18        begin
19          List.iter (fun voisin ->  if etat.(voisin) = Vierge then visite voisin x) g.(x);
20          let nb_lie = List.fold_left (fun acc voisin -> acc + if etat.(voisin) = Lie then 1 else 0) 0 g.(x) in
21          let acyclique = List.fold_left
22            (fun acc voisin -> if voisin <> pere && etat.(voisin) = Ouvert then false else acc) true g.(x) in
23            (* si il y a un cycle, alors x n'est pas pendant *)
24          if nb_lie = 0 && acyclique
25            then etat.(x) <- Pendant
26            else etat.(x) <- Lie
27        end
28      in
29
30    visite 0 (-1);
31    etat    (*renvoie le tableau *)
```