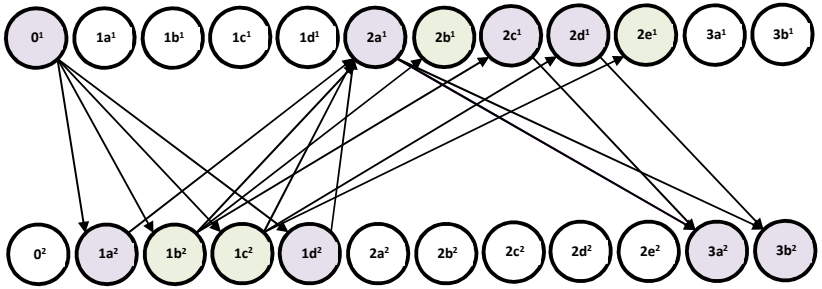
Partie 3 : Les attracteurs

La définition des positions gagnantes déterminées précédemment ({0,2a,2c,2d}∈S1 gagnantes atteignables par J1 et {1b,1c}∈S2 gagnantes atteignables par J2) peut être étendue aux sommets de S2 gagnants pour J1 et aux sommets de S1 gagnants pour J2. En effet, on remarque qu'une position qui n'est pas gagnante pour le joueur qui y joue signifie que ses successeurs sont tous gagnants pour l'autre joueur. Cette position est donc gagnante pour l'autre joueur. **Ainsi, si aucun match nul n'est possible, toute position est gagnante pour l'un des 2 joueurs au cours de la partie**. Les positions qui ne sont pas gagnantes {1a,1d,2b,2e} sont donc des positions gagnantes pour l'adversaire du joueur qui y joue.

On définit ainsi les **attracteurs d'un joueur** (ou **région gagnante**), c'est-à-dire **l'ensemble des positions (des deux joueurs) qui lui garantissent de pouvoir gagner**.
Dans notre exemple avec C=N=2 :

- Attracteurs du joueur J1 : A1={0,1a,1d,2a,2c,2d,3a,3b}
- Attracteurs du joueur J2 : A2={1b,1c,2b,2e}



Ces attracteurs sont complémentaires, l'union des deux constitue l'ensemble des sommets : S=A1UA2

Nous souhaitons créer la liste des attracteurs de chaque joueur de manière informatique. On note $G = (S, A)$ le graphe avec S l'ensemble des sommets et A l'ensemble des arêtes. Appelons F un ensemble de sommets. On appelle $Attr_1(F)$ la liste des attracteurs de F pour le joueur J1, c'est-à-dire la liste des sommets de S assurant le joueur J1 d'atteindre F.

On appelle $Attr_1^k(F)$ l'ensemble des sommets à partir desquels le joueur J1 peut arriver dans F en au plus i coups. Le calcul des attracteurs se fait par itérations sur k (exemple pour le joueur J1) :

- $Attr_1^0(F) = F$
- Tant que $Attr_1^{k+1}(F) \neq Attr_1^k(F)$
- $$Attr_1^{k+1}(F) = \begin{cases} Attr_1^k(F) \\ U\{x \in S_1 \mid \exists x' \in Attr_1^k(F), (x,x') \in A\}\ (1) \\ U\{x \in S_2 \mid \forall x'/(x,x') \in A : x' \in Attr_1^k(F)\}\ (2) \end{cases}$$

  Autrement dit :
  - Les sommets actuellement trouvés menant à F
  - (1) Les sommets du graphe du joueur J1 ayant au moins une arête conduisant aux sommets menant à F (le joueur J1 choisira ce coup)
  - (2) Les sommets du graphe du joueur J2 dont les arêtes conduisent toutes à des sommets menant à F (le joueur J2 n'aura pas le choix)

On notera que :

- $Attr_1^0(F) \in Attr_1^1(F) \in \cdots \in Attr_1(F)$
- $Attr_1^i(F)$ est une suite convergente avec au plus n itérations avec $n = |S|$ le nombre de sommets
- Dans le cas du jeu des tablettes, à chaque itération, une des deux lignes (1) ou (2) ne trouvera aucun sommet dans S1 ou S2.
- Dans le cas de ce jeu, on pourrait ne pas traiter tous les sommets à chaque itération en se limitant successivement à chaque « étage » du jeu, mais la méthode ci-dessus s'applique à d'autres types de jeux et sera donc programmée telle que proposée.

Calcul des attracteurs

On souhaite créer 2 dictionnaires dAi contenant :

- pour clés, les sommets du graphe,
- pour valeur un booléen True ou False indiquant si le sommet (la clé) est dans les sommets gagnants du joueur Ji.

Lors de l'initialisation des attracteurs des deux joueurs dans le cas de notre jeu, on a :

- Les attracteurs du joueur J1 sont les sommets de S2 sans successeurs
- Les attracteurs du joueur J2 sont les sommets de S1 sans successeurs

A chaque fois donc, le dernier coup du joueur Ji doit mener dans les sommets sans successeurs de l'autre joueur.

Q18- Créer la **fonction init_attracteurs(G,S1)** prenant en paramètre le graphe G du jeu et le Tuple S1 des sommets du joueur J1 et renvoyant les deux dictionnaires attendus.
Dans le cas C=N=2, vérifier :

```
dA1,dA2 = init_attracteurs(Graphe,S1)
```

affiche

```
dA1= {((0, 1), (0, 1), (1, 1), (1, 1)): False, ((0, 2), (1, 1), (1, 1)): False, ((0, 1)
, (0, 2), (1, 1)): False, ((0, 1), (1, 1), (1, 2)): False, ((0, 1), (0, 1), (1, 2)): Fa
lse, ((0, 2), (1, 2)): False, ((0, 3), (1, 1)): False, ((0, 2), (0, 2)): False, ((1, 2)
, (1, 2)): False, ((0, 1), (1, 3)): False, ((0, 4),): True, ((1, 4),): True}
dA2= {((0, 1), (0, 1), (1, 1), (1, 1)): False, ((0, 2), (1, 1), (1, 1)): False, ((0, 1)
, (0, 2), (1, 1)): False, ((0, 1), (1, 1), (1, 2)): False, ((0, 1), (0, 1), (1, 2)): Fa
lse, ((0, 2), (1, 2)): False, ((0, 3), (1, 1)): True, ((0, 2), (0, 2)): False, ((1, 2),
 (1, 2)): False, ((0, 1), (1, 3)): True, ((0, 4),): False, ((1, 4),): False}
```

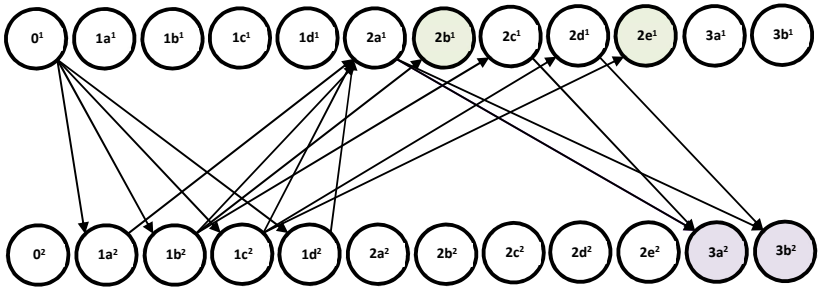
On appelle Cond_1 et Cond_2 les conditions appliquées à une position x pour le joueur Ji telles que:

$$\begin{cases} Cond\_1 : \exists x' \in Attr_i^k(F), (x,x') \in A \\ Cond\_2 : \forall x'/(x,x') \in A : x' \in Attr_i^k(F) \end{cases}$$

Q19- Créer la **fonction cond_1(G,di,x)** prenant en paramètre le graphe du jeu G, le dictionnaire di des attracteurs du joueur Ji et un sommet x du jeu (liste), et renvoyant le booléen True si le sommet respecte la condition (1), False sinon.

Q20- Créer la **fonction cond_2(G,di,x)** prenant en paramètre le graphe du jeu G, le dictionnaire di des attracteurs du joueur Ji et un sommet x du jeu (liste), et renvoyant

le booléen True si le sommet possède des successeurs et respecte la condition (2), False sinon.
Les dictionnaires n'étant qu'initialisés ainsi :



On ne peut vérifier ces conditions que pour certaines positions particulières pour le moment :
- Pour dA1 : vérifier les résultats pour 2a à 2e
- Pour dA2 : vérifier les résultats pour 1a à 1d

Puis :
- Pour dA2 : vérifier les résultats pour 2a à 2e
- Pour dA1 : vérifier les résultats pour 1a à 1d

Q21- Créer la **fonction attracteurs_it(G,di,Si)** prenant en paramètre le graphe G, le dictionnaire di des attracteurs de Ji, et le Tuple Si des positions du joueur Ji, réalisant une itération de la procédure de détermination des attracteurs du joueur Ji en changeant les valeurs dans di (en place), et renvoyant True si au moins un changement (False vers True) a eu lieu, False sinon.

Remarques :
- On veillera à ne traiter que les sommets n'étant pas à True (donc à False) dans di pour ne pas risquer de repasser à False des sommets initialisés à True n'ayant donc pas de successeurs. Et, cela gagnera du temps.
- Cette fonction pourrait être optimisée dans le cas du jeu des tablettes en ne considérant pas tous les sommets False à chaque itération, je ne souhaite pas aller plus loin ici

Q22- Créer la **fonction attracteurs_Ji(G,di,Si)** avec les mêmes paramètres que attracteurs_it réalisant la procédure complète de création des attracteurs du joueur Ji en complétant di.

Q23- Créer enfin **la fonction attracteurs(G,C,N)** créant et renvoyant les dictionnaires dA1 et dA2 des attracteurs des joueurs J1 et J2
Vérifier vos dictionnaires avec les attracteurs proposés précédemment dans le cas C=N=2.

Les positions gagnantes pour le joueur y joue obtenues précédemment peuvent être déterminées à l'aide des attracteurs des deux joueurs. En effet, **une position est gagnante pour le joueur Ji qui s'y trouve si cette position est dans Si et fait partie de ses attracteurs Ai**.

Q24- Créer la **fonction dico_gagnant_att(G,C,N)** prenant en paramètres le graphe G, C et N, et renvoyant le dictionnaire des positions gagnantes. Vérifier que vous obtenez le même dictionnaire qu'avec les fonctions dico_gagnant ou dico_gagnant_opt.

Intelligence artificielle
Théorie des jeux
Jeu des tablettes

Stratégie optimale

On souhaite maintenant créer une fonction qui, pour toute position x du jeu, renvoie la stratégie optimale à jouer, c'est-à-dire le meilleur coup.

On peut définir le meilleur coup à partir de la position x ainsi :
- Si x n'a pas de successeurs, on renvoie une liste vide
- Si x a des successeurs, on choisit de la sorte :
  - Création de la liste des choix L_Choix contenant
    - Les successeurs qui ne sont pas gagnants s'il y en a
    - Tous les successeurs sinon
  - Choix aléatoire d'un élément de L_Choix

Ainsi, **le joueur qui joue choisit à chaque fois une position suivante n'étant pas gagnante pour l'adversaire (donc gagnante pour lui), si elle existe**.
Remarque : cela revient à choisir, pour chaque joueur, un sommet parmi ses attracteurs, mais nous ne le programmerons pas ainsi puisqu'il faut alors différencier le coup de chaque joueur.

Q25- Créer la **fonction strategie_opt(G,dg,x)** prenant en paramètre le graphe G, le dictionnaire gagnant dg et une position x (liste), et renvoyant un choix de successeur respectant le choix du meilleur coup.
En utilisant plusieurs fois la fonction, vérifier :

```
C=N=2
G = graphe(C,N)
dg = dico_gagnant(G)
x0 = init(C,N)
test = strategie_opt(G,dg,x0)
print(test)  # affiche [[0, 1], [0, 1], [1, 2]]
test = strategie_opt(G,dg,x0)
print(test)  # affiche [[0, 1], [1, 1], [1, 1]]
```

Q26- Créer la **fonction strategies_opt(G,dg)** prenant en paramètre le graphe G et le dictionnaire gagnant dg, et renvoyant un dictionnaire dico_s dont chaque clé est une position x du jeu (Tuple), et chaque valeur la solution issue du meilleur coup à jouer depuis x pour le joueur qui y est.

Voici un exemple de résultat obtenu, mais il n'est évidemment pas unique :

```
C=N=2
G = graphe(C,N)
dico_g = dico_gagnant(G)
st_opt = strategies_opt(G,dico_g)
print(st_opt)
''' Résultat
{((0, 1), (0, 1), (1, 1), (1, 1)): [[0, 2], [1, 1], [1, 1]], ((0, 2), (1, 1), (1, 1)):
[[0, 2], [1, 2]], ((0, 1), (0, 2), (1, 1)): [[0, 3], [1, 1]], ((0, 1), (1, 1), (1, 2)):
 [[0, 1], [1, 3]], ((0, 1), (0, 1), (1, 2)): [[0, 2], [1, 2]], ((0, 2), (1, 2)): [[1, 4
]], ((0, 3), (1, 1)): [], ((0, 2), (0, 2)): [[0, 4]], ((1, 2), (1, 2)): [[1, 4]], ((0,
1), (1, 3)): [], ((0, 4),): [], ((1, 4),): []}     '''
```

Simulation de jeu en IA

Nous allons maintenant simuler un jeu où chaque joueur est une intelligence artificielle.

Q27- Créer la **fonction jeu(C,N)** qui affiche le joueur disposant d'une stratégie gagnante au départ, les étapes du jeu en précisant quel joueur joue, et quel joueur gagne

Q28- Utiliser la fonction jeu pour différentes situations et conclure.

Exemples d'exécutions :

```
>>> jeu(2,2)
Le joueur 1 dispose d'une position gagnante
Départ: [[0, 1], [0, 1], [1, 1], [1, 1]]
Joueur: 1
[[0, 2], [1, 1], [1, 1]]
Joueur: 2
[[0, 2], [1, 2]]
Joueur: 1
[[1, 4]]
Joueur: 2
[]
a perdu
>>> jeu(3,3)
Le joueur 1 dispose d'une position gagnante
Départ: [[0, 1], [0, 1], [0, 1], [1, 1], [1, 1], [1, 1], [2, 1], [2, 1], [2, 1]]
Joueur: 1
[[0, 1], [0, 1], [1, 1], [1, 1], [1, 2], [2, 1], [2, 1], [2, 1]]
Joueur: 2
[[0, 1], [0, 1], [1, 1], [1, 1], [1, 2], [2, 1], [2, 2]]
Joueur: 1
[[0, 1], [1, 1], [1, 2], [1, 2], [2, 1], [2, 2]]
Joueur: 2
[[0, 1], [1, 1], [1, 2], [1, 4], [2, 1]]
Joueur: 1
[[0, 1], [1, 2], [1, 2], [1, 4]]
Joueur: 2
[[0, 1], [1, 4], [1, 4]]
Joueur: 1
[[0, 1], [1, 8]]
Joueur: 2
[]
a perdu
>>> jeu(2,3)
Le joueur 2 dispose d'une position gagnante
Départ: [[0, 1], [0, 1], [0, 1], [1, 1], [1, 1], [1, 1]]
Joueur: 1
[[0, 1], [0, 1], [0, 2], [1, 1], [1, 1]]
Joueur: 2
[[0, 2], [0, 2], [1, 1], [1, 1]]
Joueur: 1
[[0, 4], [1, 1], [1, 1]]
Joueur: 2
[[0, 4], [1, 2]]
Joueur: 1
[]
a perdu

>>> jeu(3,2)
Le joueur 2 dispose d'une position gagnante
Départ: [[0, 1], [0, 1], [1, 1], [1, 1], [2, 1], [2, 1]]
Joueur: 1
[[0, 2], [1, 1], [1, 1], [2, 1], [2, 1]]
Joueur: 2
[[0, 2], [1, 1], [1, 1], [2, 2]]
Joueur: 1
[[1, 1], [1, 1], [2, 4]]
Joueur: 2
[[1, 2], [2, 4]]
Joueur: 1
[]
a perdu
```