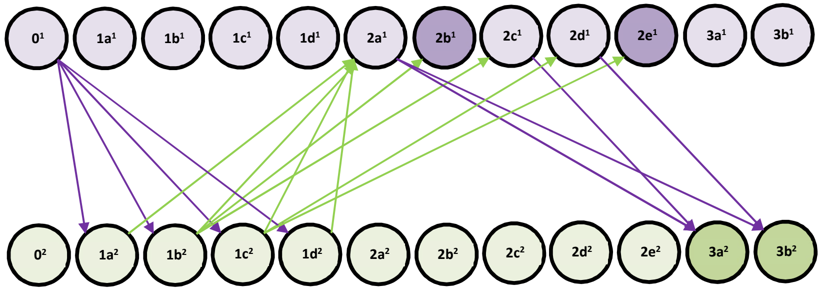
### *Partie 2 : Graphe biparti*

Dans le cas des jeux d'accessibilité à 2 joueurs, on peut représenter le graphe orienté précédent sous la forme d'un graphe **biparti** en doublant les sommets, en indiquant leurs noms par le numéro du joueur (J1 ou J2) et en choisissant le joueur qui commence (J1) :




L'ensemble des sommets S={0,1a,1b,1c,1d,2a,2b,2c,2d,2e,3a,3b} de ce graphe se décompose en deux ensembles S1={0,2a,2b,2c,2d,2e} et S2={1a,1b,1c,1d,3a,3b} tels que :

S = S1 U S2 et S1 ∩ S2 = Ø avec Si l'ensemble des positions depuis lesquels le joueur Ji jouera.

Comme dit précédemment, dans un graphe biparti, les arêtes ne peuvent relier qu'un sommet de S1 à un sommet de S2 et inversement.

**Q9-** En remarquant que les sommets du joueur J1 ont un nombre de piles de même parité que x0, créer la **fonction sommets_12(G,C,N)** prenant en paramètre le graphe G, N et C, et retournant les 2 Tuples des sommets S1 et S2 des joueurs J1 et J2

**Q10-** Créer les Tuples S1 et S2 dans le cas C=N=2.
Vérifier les résultats avec le graphe biparti symbolisé ci-dessus.

### *Les parties*

**Une partie est un parcours du jeu**, c'est-à-dire un chemin fini ou infini de sommets du graphe. Une **partie est déclarée gagnée lorsqu'elle se termine dans un état final de J1 (F1) ou de J2 (F2)**, ce qui définit le gagnant.

**Q11-** Pour C=N=2, et en prenant à chaque fois le premier successeur identifié dans le graphe, afficher une partie et préciser le joueur gagnant.

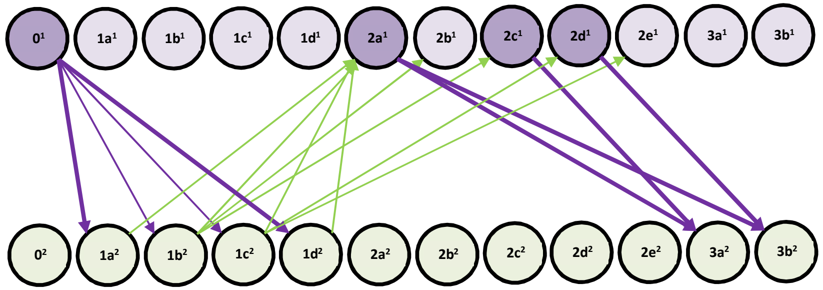
Vérifier :

```
Joueur: 1
Jeu: [[0, 1], [0, 1], [1, 1], [1, 1]]
Joueur: 2
Jeu: [[0, 2], [1, 1], [1, 1]]
Joueur: 1
Jeu: [[0, 2], [1, 2]]
Joueur: 2
Jeu: [[0, 4]]
Le joueur 2 a gagné
Cela correspond à 0 - 1a - 2a - 3a
```

Intelligence artificielle
Théorie des jeux
Jeu des tablettes

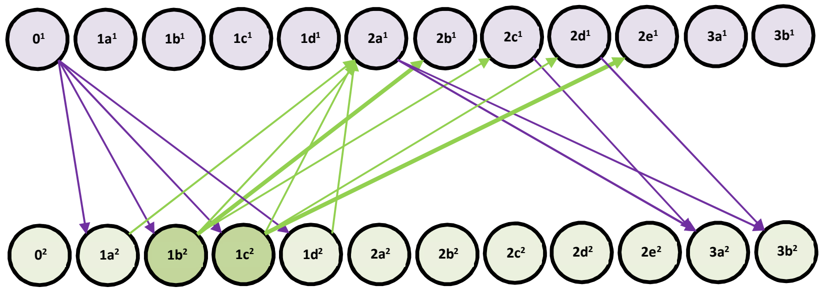
### Stratégies et positions gagnantes

Une **stratégie est une méthode à appliquer à chaque coup**. Nous venons de réaliser la stratégie suivante : « à chaque coup, on choisit la première solution que notre algorithme a déterminé dans le graphe ». Nous nous intéressons aux stratégies sans mémoire qui ne dépendent que de l'état actuel du jeu (on ne peut se dire : la dernière fois, l'adversaire a fait ça, alors je fais ça…).
Une **stratégie est dite gagnante pour un joueur si**, en jouant cette stratégie, **toute partie est finie et se termine dans un état gagnant pour ce joueur**. On dit alors que la partie est jouée suivant cette stratégie.
Sur le graphe biparti précédent, nous représentons en gras une stratégie gagnante pour le joueur J1 :




Une **position x est dite gagnante s'il existe une stratégie gagnante depuis ce sommet**. Autrement dit, si le joueur qui y joue a beaucoup de mémoire, quels que soient les coups de son adversaire, il pourra forcer ce dernier à perdre. Cela ne veut pas dire que l'autre joueur est perdant/ne peut pas gagner, cela veut dire que si le joueur disposant d'une position gagnante ne commet aucune erreur, il gagnera. Les positions {0,2a,2c,2d} sont gagnantes atteignables par le joueur J1.

Sur le graphe biparti ci-dessus, la position 0 est une position gagnante pour le joueur J1. En effet, en suivant la stratégie gagnante depuis 0, il gagnera quel que soit le chemin suivi (positions $1b^2$ et $1c^2$ à proscrire). Si le joueur J1 va sur $1b^2$ ou $1c^2$, le joueur J2 dispose alors d'une position gagnante :




Les positions {1b,1c} sont gagnantes atteignables par le joueur J2.

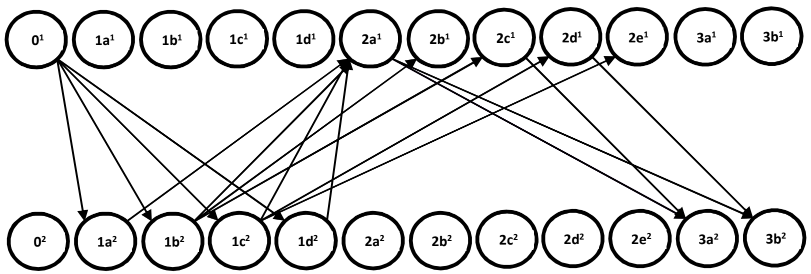
Finalement, les positions {0,1b,1c,2a,2c,2d} sont gagnantes pour le joueur qui y joue, quel qu'il soit.

### Détermination des positions gagnantes

Dans toute cette partie, je vais essayer de privilégier les termes « elle n'est pas gagnante » plutôt que « elle est non gagnante », ou encore pire « elle est perdante », car ce que l'on peut dire, c'est qu'une position est gagnante ou ne l'est pas.
Pour déterminer si une position est gagnante pour un joueur, on propose une méthode récursive où l'objectif de chacun est identique : ne pas atteindre une position n'ayant pas de successeurs.

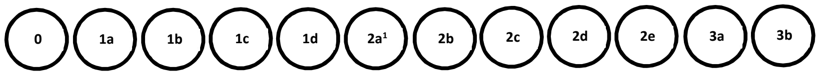
Ainsi, une position :
- Est gagnante s'il existe au moins un successeur qui n'est pas gagnant
- N'est pas gagnante si :
    - Elle n'a pas de successeurs
    - Aucun de ses successeurs n'est pas gagnant = Tous ses successeurs sont gagnants

Soit le schéma ci-dessous :




**Q12-** En vous aidant du schéma ci-dessus, griser les positions gagnantes pour le joueur qui y joue sur le bilan ci-dessous

Bilan des positions gagnantes :




On remarque que le joueur J1 dispose d'une position gagnante en début de partie dans le jeu avec N=C=2.

**Q13-** Créer la **fonction est_gagnante(G,x)** prenant en paramètre le graphe du jeu et la position x (liste ou Tuple) et renvoyant le booléen True si la position est gagnante pour le joueur qui y joue, et False sinon

Vérifier votre proposition de la question précédente.
On souhaite mettre en place un dictionnaire des positions gagnantes afin de mémoriser si une position du graphe est gagnante ou non pour le joueur qui y joue.

**Q14-** Définir une **fonction dico_gagnant(G)** qui retourne un dictionnaire dont les clés sont les positions du graphe et les valeurs, le booléen True ou False indiquant si la position est gagnante ou non.
Exemple d'exécution :

```
dico_g = dico_gagnant(Graphe)
print(dico_g)
#affiche {((0, 1), (0, 1), (1, 1), (1, 1)): True, ((0, 2), (1, 1), (1, 1)): False,
((0, 1), (0, 2), (1, 1)): True, ((0, 1), (1, 1), (1, 2)): True, ((0, 1), (0, 1), (1
, 2)): False, ((0, 2), (1, 2)): True, ((0, 3), (1, 1)): False, ((0, 2), (0, 2)): Tr
ue, ((1, 2), (1, 2)): True, ((0, 1), (1, 3)): False, ((0, 4),): False, ((1, 4),): F
alse}
```

La fonction est_gagnante réalisée précédemment :
- Recalcule plusieurs fois le statut est_gagnant de sous-situations lors du calcul du statut d'une seule situation
- Est appelée et refait tout le travail lors du remplissage du dictionnaire

On se propose donc d'optimiser la création du dictionnaire en utilisant la technique de mémoïsation qui retiendra l'état gagnant ou non de chaque situation rencontrée et renverra directement le dictionnaire de tout le graphe.

**Q15-** Proposer une **fonction dico_gagnant_opt(G)** renvoyant le dictionnaire des états gagnants des positions du graphe avec mémoïsation
Vérifier que vous obtenez le même résultat qu'avec dico_gagnant.

**Q16-** Comparer les temps d'exécution de dico_gagnant et dico_gagnant_opt pour C=N=3

Remarque : Pour déceler d'éventuelles erreurs de programmation, ajouter une instruction **print(**'mémo'**)** lorsque la mémoïsation joue son rôle afin de vérifier que c'est bien le cas.

Pour la suite, on écrira : dico_gagnant **=** dico_gagnant_opt

### Etude des positions gagnantes au départ

On souhaite étudier les cas du tableau proposé ci-dessous afin de compléter chaque case avec le numéro du joueur disposant d'une position gagnante au départ du jeu pour différentes valeurs de N et C.

| N = | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| C = 1 | | | | |
| C = 2 | | | | |
| C = 3 | | | | 1 |
| C = 4 | | | 2 | 1 |

.

**Q17-** Mettre en place le code nécessaire et remplir le tableau proposé
Le jeu classique que l'on peut acheter possède C=4 et N=3. Malheureusement, nous avons vu que nous ne pouvons créer le graphe des cases grises… Nous verrons plus tard qu'un algorithme appelé min-max permet de mettre en place une stratégie de jeu sans avoir de graphe à disposition, ce qui réussir à remplir les cases grises.