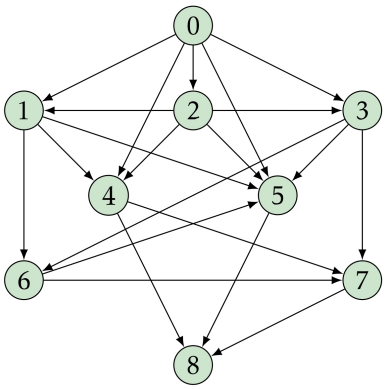
TP : Jeux

I) Jeu de Chom(2,3)

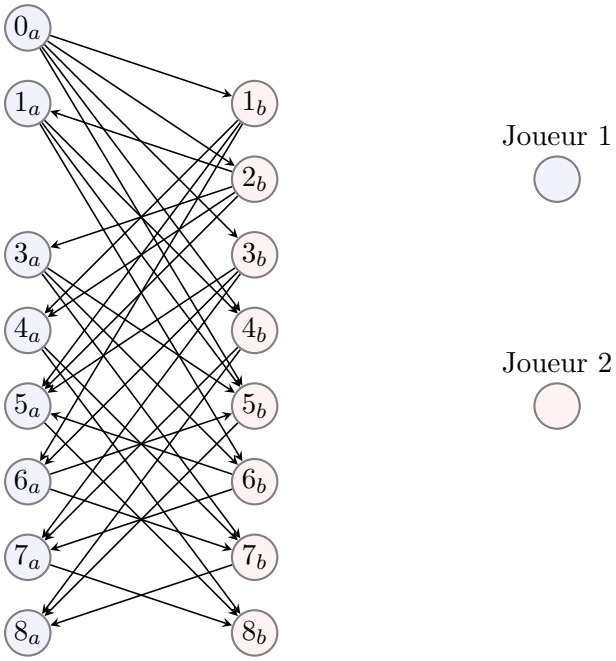
Le but de ce TP est de faire tourner l'algorithme Max-Min sur le graphe biparti du jeu de chomp(2,3) vu en cours.

Exercice 1 (Graphe de départ)
Ce graphe est aussi appelé arène, il ne contient pas l'information « qui joue ».



Créer une liste d'adjacence `G0` codant le graphe ci-dessus.

Exercice 2 (Création du graphe biparti)
Nous allons coder le graphe biparti à l'aide d'un dictionnaire, en essayant de structurer le code. Voici le graphe biparti :



1) Dans un premier temps, on code les arêtes des sommets x_a vers les sommets y_b par une liste d'adjacence G_a : `Ga[0]` contient la liste `[1, 2, 3, 4, 5]` car le joueur J_1, depuis le sommet 0, peut aller vers les sommets 1 à 5.

Vous êtes invités à utiliser `G0` construit à l'exercice précédent autant que possible.

Dans un premier temps, plutôt qu'une absence de sommet 2_a, on codera un sommet isolé : la liste des flèches depuis 2_a sera vide, *id est* `Ga[2]==[]`.

2) Créer la liste G_b des flèches des sommets y_b vers les sommets x_a.

3) Créer un dictionnaire `G` dont les clefs sont les sommets du graphe biparti au format chaine de caractère (`'0a'`), et les valeurs associées aux clefs des listes de clefs :

```
>>> G['1b']
['4a', '5a', '6a']
```

4) (Bonus) Créer une fonction qui prend un graphe de jeu (donné par une liste d'adjacence sous forme de liste de liste, comme `G0`) et qui retourne le graphe biparti associé sous forme de dictionnaire.

Exercice 3 (Max-Min)
Le but de l'exercice est de coder l'algorithme Max-Min décrit dans le cours : Lors d'un parcours en profondeur du graphe, on remonte les valeurs alternativement minimales (si c'est J_2 qui joue) ou maximales (si c'est J_1) :

1) Écrire une fonction `h(s)` qui prend en argument un sommet `s` du graphe biparti – donc une clef du dictionnaire construit à l'exercice précédent – et qui retourne 1 si le sommet est un état final pour J_1, -1 si c'est un état final pour J_2, et 0 sinon.

2) Écrire une fonction, `maximin(s)` qui retourne `h(s)` si le sommet `s` n'a pas de successeurs, et sinon le maximum sur les successeurs u de `s` du (minimum sur les successeurs u' de u du (maximum sur les successeurs de u' etc)) : on s'aidera d'une fonction `minimax(s)`.

3) Modifier les fonctions précédentes pour que les appels récursifs s'arrêtent au bout de n étapes : `maximin(s, n)` et `minimax(s, n)`.

On testera toutes ses fonctions sur le graphe précédent.