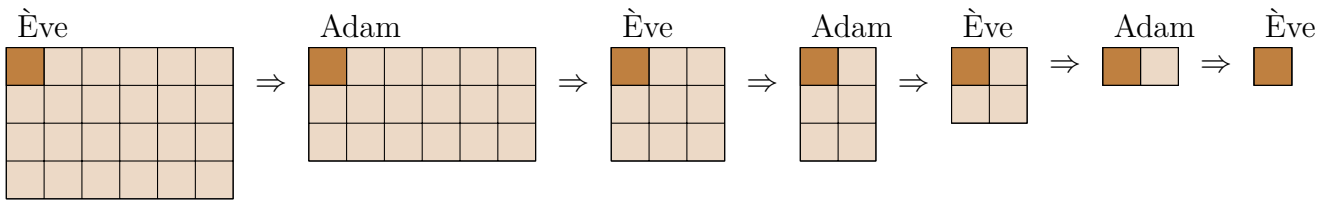


## ? Informatique du Tronc Commun TD n°5 Théorie des jeux

### Exercice n°1 Jeu du chocolat empoisonné

On dispose d'une plaquette de chocolat de dimensions  $m \times n$ . Le carré de chocolat dans le coin en haut à gauche est empoisonné. Chaque joueur à son tour coupe suivant une coupe verticale ou horizontale une part de la plaquette de chocolat qu'il doit manger. Le joueur qui mange le carré empoisonné a perdu.

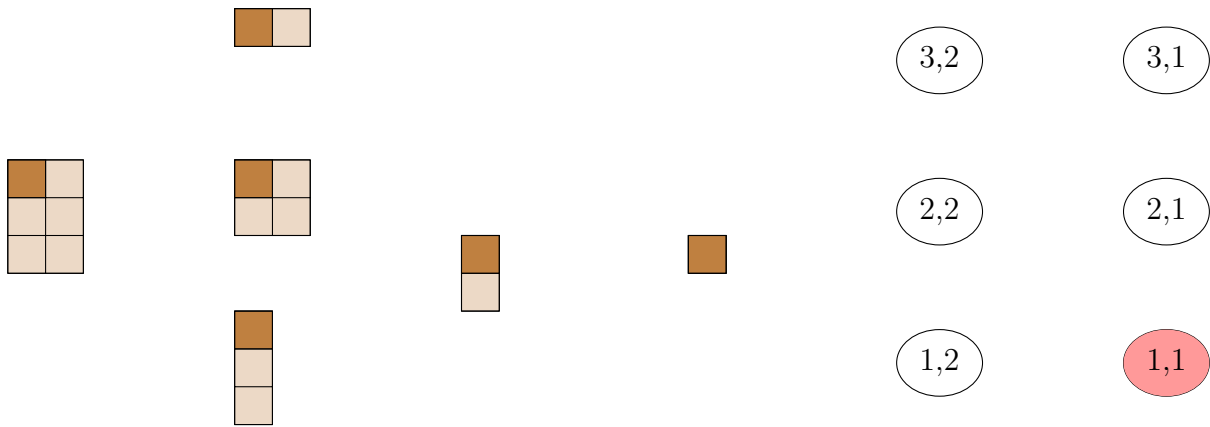
Voici l'exemple d'une partie avec une tablette de dimensions  $4 \times 6$ . Les joueurs se nomment Adam et Eve. C'est Ève qui commence :



À ce jeu nous pouvons associer un graphe orienté  $(S, A)$  appelé Arène du jeu. Les sommets du graphe sont les états possibles de la plaque de chocolat, ici on peut décrire un état par un couple  $(a, b)$  avec  $0 < a \leq m$  et  $0 < b \leq n$ .

Un arc relie le sommet  $s_1$  au sommet  $s_2$  lorsqu'un joueur peut, en un coup, passer de l'état  $s_1$  à l'état  $s_2$ .

Q1. Compléter l'arène du jeu ci-dessous en indiquant les flèches permettant de passer d'un sommet à un autre.



Q2. Compléter le graphe du jeu (ci-dessus à droite) pour une tablette initiale  $3 \times 2$ , en indiquant les arêtes.

Q3. Que peut-on dire la position  $(1, 1)$  pour le joueur qui doit jouer (gagnante ou perdante?) ?

Que peut-on en déduire sur les positions  $(1, 2)$ ,  $(2, 1)$ ,  $(3, 1)$  (gagnante ou perdante?) ? En déduire que la position  $(2, 2)$  est une position perdante. Conclure sur la nature de la position  $(3, 2)$  ?

Q4. Pour la tablette  $3 \times 2$ , que peut-on dire du joueur qui commence ?

### Exercice n°2 Jeu du chocolat empoisonné (2) : détermination du noyau

Considérons un graphe orienté acyclique  $(S, A)$  associé à un jeu d'accessibilité.

**Propriété :** Dans un graphe acyclique il existe des sommets sans successeur.

**Définition :** Un sous-ensemble  $S'$  de sommets est dit stable si tout sommet de  $S'$  n'a aucun successeur dans  $S'$ . Un sous-ensemble  $S'$  de sommets est dit absorbant si tout sommet n'appartenant pas à  $S'$  possède au moins un successeur dans  $S'$ . Un sous-ensemble  $S'$  de sommets est un noyau s'il est à la fois stable et absorbant.

**Théorème :** Tout graphe orienté et acyclique possède un unique noyau.

Pour calculer le noyau de  $(S, A)$  il suffit de :

- chercher un sommet  $s$  de  $(S, A)$  sans successeur, il fait partie du noyau ;
- supprimer de  $(S, A)$ ,  $s$  et ses prédécesseurs ;
- recommencer tant qu'il reste des sommets.

Q1. Mettre en œuvre l’algorithme précédent pour calculer le noyau du jeu de Chomp (2,3) étudié dans l’exercice précédent.

Remarque. Dans le cas du jeu de Chomp les éléments du noyau sont les positions perdantes et les autres sommets les positions gagnantes. La stratégie gagnante consiste, pour chaque sommet qui n’est pas dans le noyau, à jouer un coup qui s’y ramène.

On considère un graphe acyclique  $(S, A)$ . Il est représenté en machine par la donnée d’un dictionnaire  $d$  dont les clefs sont les sommets et les valeurs les successeurs des clefs (sous forme de liste).

Q2. Rédiger une fonction `sansSuccesseur(d:dict)->int` qui renvoie un sommet sans successeur d’un graphe représenté par son dictionnaire  $d$ .

Q3. Rédiger une fonction `predecesseurs(d:dict,j:int)->list` qui renvoie la liste de tous les prédécesseurs du sommet  $j$ .

Q4. Rédiger une fonction `supprimeSommet(d:dict,j:int)->dict` qui supprime le sommet  $j$  du graphe représenté par  $d$ . La fonction modifie le dictionnaire  $d$ .

Q5. En déduire une fonction `noyau(d:dict)->list` qui renvoie la liste des sommets constituant le noyau de  $(S, A)$ .

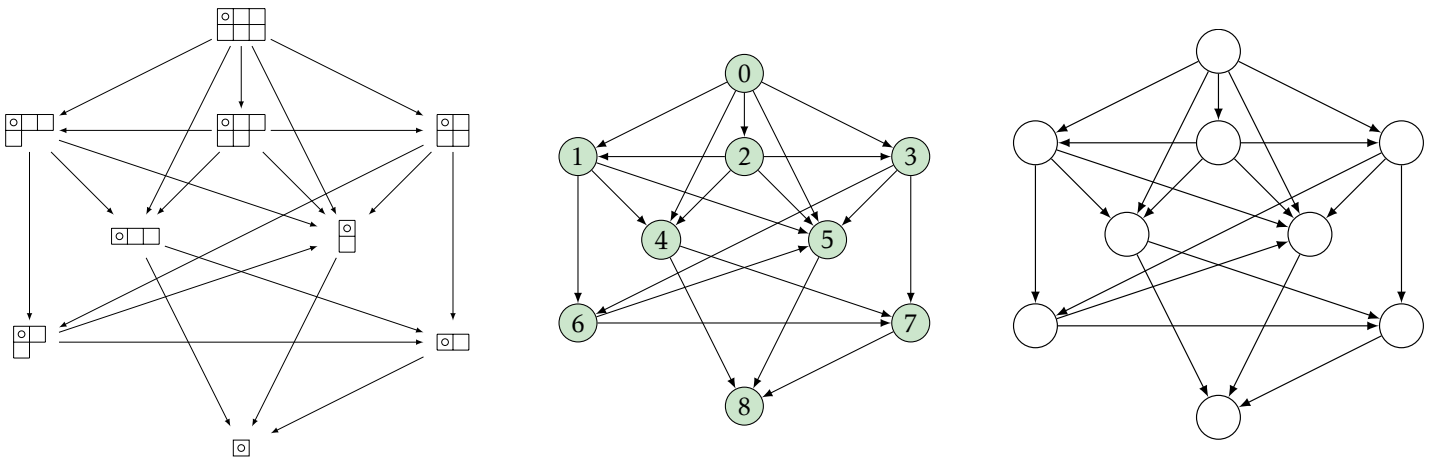
### Exercice n°3 Jeu du chocolat empoisonné (3) : Fonction de Sprague-Grundy

On considère un graphe orienté acyclique  $(S, A)$  associé à un jeu impartial dans lequel le perdant est celui qui ne peut plus jouer. On définit le nombre de Grundy,  $n(s)$  d’un sommet  $s \in S$  de la façon suivante :

- si  $s$  est une position sans successeur (position finale),  $n(s) = 0$ ;
- sinon, le nombre de Grundy  $n(s)$  est le plus petit entier positif ou nul qui n’est pas le nombre de Grundy d’un des successeurs de  $s$  (c’est-à-dire les positions directement accessibles depuis  $s$ ).

Le noyau du graphe correspond aux sommets  $s$  vérifiant  $n(s) = 0$ .

On considère le jeu de Chomp (2,4) pour lequel un joueur peut manger à chaque tour un rectangle (qui peut être composé de plusieurs carreaux) dont on donne l’arène de jeu ci-dessous.



Q1. Reporter sur le graphe ci-dessus à droite le nombre de Grundy de chacun de ses sommets.

On représente un graphe par un dictionnaire  $d$  dont les clefs sont les sommets et les valeurs la liste des successeurs de la clef correspondante. Le but de cet exercice est de construire un dictionnaire  $n$  dont les clefs sont les sommets et les valeurs les nombre de Grundy de ces sommets.

Q2. Rédiger une fonction `sans_nb_grundy(d:dict, n:dict)->int` qui renvoie un sommet  $s$  ne possédant pas encore de nombre de Grundy mais dont tous les successeurs en possèdent. Cette fonction renverra `None` dans le cas où un tel sommet n’existe pas.

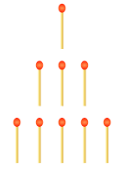
Q3. En déduire une fonction `grundy(d:dict)->dict` qui renvoie le dictionnaire  $n$  des nombres de Grundy des différents sommets composants ce graphe représenté par le dictionnaire  $d$ .

## Exercice n°4 Jeu de Marienbad

Le jeu de Marienbad est un jeu avec plusieurs tas d'allumettes. Chaque joueur peut retirer autant d'allumettes qu'il veut sur une seule ligne. Le perdant est celui qui retire la dernière allumette.

Il est connu pour apparaître dans le film *L'année dernière à Marienbad* d'Alain Resnais (Marienbad est une ville de République Tchèque).

Pour la modélisation par un graphe nous allons considérer une situation plus simple : le jeu a trois lignes dont la première ligne possède une seule allumette et les deux suivantes possèdent deux allumettes  $\begin{array}{c} | \\ || \\ || \end{array}$ . Nous considérerons



que les situations  $\begin{array}{c} | \\ || \\ | \end{array}$  et  $\begin{array}{c} | \\ | \\ || \end{array}$  sont identiques (cela permet de limiter le nombre de sommets dans le graphe).

- Q1. Représenter le graphe du jeu en indiquant les sommets (indice : il y en a 8) et les arcs les reliant (correspondant à des coups qui peuvent être joués).
- Q2. Créer le graphe biparti associé et déterminer l'attracteur du joueur J1 (le premier qui joue).  
Vous griserez les états qui sont dans l'attracteur en indiquant le rang de chaque état dans l'attracteur.

### Modélisation d'un tel jeu

- Q3. Proposer la modélisation d'un état du jeu (de la question 1) à l'aide d'un tuple.
- Q4. Écrire une fonction `est_trie(t:tuple)->bool` qui prend en argument un tuple `t` qui vérifie si une liste est triée par ordre croissant, elle renvoie un booléen.
- Q5. Écrire les instructions qui créent la liste des sommets du graphe de la question 1.
- Q6. On suppose avoir écrit `succ(S:list,p:int)->list` qui prend en argument la liste des sommets `S` du graphe ci-dessus, un sommet `p` et qui renvoie la liste de ses successeurs.  
Écrire une fonction `graphe(list)->dict` qui prend en argument la liste des sommets `S` et qui renvoie le graphe de la question 1 sous la forme d'un dictionnaire d'adjacence.
- Q7. Écrire une fonction `graphe_biparti(S:list)->dict` qui prend en argument la liste des sommets `S` du graphe ci-dessus et qui renvoie l'arène du jeu (c'est à dire un graphe biparti).

On souhaite maintenant calculer l'attracteur et une stratégie gagnante sur ce graphe. On suppose avoir un graphe biparti `G` qui représente l'arène de jeu, `V1` les positions de victoire pour le joueur J1 et `V2` les positions de victoire pour le joueur J2.

- Q8. Écrire une fonction `nb_succ(G:dict)->dict` qui prend en argument un graphe `G` et qui renvoie un dictionnaire dont les clés sont les sommets de `G` et dont la valeur associée est le nombre de successeurs.
- Q9. Écrire un programme `pred(G:dict)->dict` qui prend en argument un graphe `G` et qui renvoie un dictionnaire dont les clés sont les sommets de `G` et dont la valeur associée est la liste des prédécesseurs du sommet.
- Q10. Compléter le programme récursif `modif_attracteur(s:int,D:dict,TG:dict,S1:list,attr:dict,n:int)->dict` qui prend en argument un sommet `s`, le dictionnaire du nombre `D` de successeurs qui ne sont pas dans l'attracteur, `TG` le dictionnaire des prédécesseurs de chaque sommet, `S1` les sommets contrôlés par le joueur 1, `attr` un dictionnaire représentant des sommets qui sont dans l'attracteur (non complet a priori) et qui code la partie récursive de l'algorithme et `n` une majoration du rang du sommet `s`. Ce programme teste si `s` est dans l'attracteur si ce n'est pas le cas, il le rajoute dans l'attracteur et fait les opérations adéquates pour trouver les sommets qui sont dans l'attracteur via un appel récursif.

```

1 def modif_attracteur(s,D,TG,S1,attr,n):
2     if # s est-il dans l'attracteur ?
3         attracteur[s]=n # Rajout de s et majoration du rang
4     for # Parcours des prédécesseurs de s
5         D[pred]=D[pred]-1 # Un successeur de plus dans l'attracteur
6         if # Condition d'appartenance à l'attracteur
7             # Appel récursif

```

Q11. En déduire une fonction `calcul_attracteur(G:dict,S1:dict,V1:dict)->dict` qui calcule l'attracteur du joueur 1 où V1 est un dictionnaire des positions de victoire.

Q12. Compléter la fonction `condition_gagnanteJ2(G:dict,att1:dict,att2:dict,s:int)->int` qui prend en argument un sommet `s` du graphe `G`, les attracteurs `att1` et `att2` et qui renvoie un sommet optimisé si nous souhaitons gagner la partie pour le joueur 2 (c'est-à-dire une position gagnante si nous sommes dans l'attracteur du joueur 2, ...).

```

1 def condition_gagnanteJ2(G,attr1,attr2,s):
2     if s in ..... # Je dois gagner si je joue bien ....
3         for x in G[s]:
4             if ..... and attr2[x]<attr2[s]: # Condition +
5             amelioration du rang
6                 return som
7     elif s in attr1: # J'ai perdu a priori
8         return .....
9     else: # Je cherche à avoir un match nul
10        for .....:
11            if .....:
                return x

```