TP 4
Initiation à la théorie des jeu et retour sur la programmation dynamique

1 Introduction à la théorie des jeux

Exercice 1.1 *Proposer une fonction Python renvoyant la liste d'adjacence des noeuds sortants du graphe du Jeu de Nim pour n batons,* `Nim(n)`*. Cette liste d'adjacence sera renvoyée sous la forme d'un dictionnaire de clé le couple* `('Va',i)` *(respectivement* `('Vb',i)`*) pour un noeud de la partition V_a (respectivement V_b) i le nombre de bâtonnets.*

```
def Nim(n):
    D={}
    for k in range(1,n+1):
        D[('Va',k)]=[]
        D[('Vb',k)]=[]
        for i in range(1,4);
             if k-i>=0:
                  D[('Va',k)].append(('Vb',k-1))
                  D[('Vb',k)].append(('Va',k-i))
        return D
```

Exercice 1.2 *Proposer une fonction* `transpose(G)`*, prenant la liste d'adjacence, des noeuds sortant, G d'un graphe et renvoyant la liste transposé, c-a-d la liste des antécédents, noeuds entrants, sous forme de dictionnaire où la clé est un noeud u de G et la valeur, la liste des noeuds v tel qu'il existe une arrete de u vers v.(par exemple le couple (Va, i) (respectivement (Vb, i)) pour un noeud de la partition V_a (respectivement V_b) i le nombre de bâtonnets et la valeur la liste des antécédents.)*

```
def transpose(G):
      D={}
      for v in G:
          for u in G[v]:
              if u in D:
                 D[u].append(v)
              else:
                 D[u]=[v]
      return D
```

1.1 Vocabulaire : arène, condition de victoire, positions gagnantes et stratégies

1.2 Attracteur

1.3 Algorithme de calcul de l'attracteur

Exercice 1.3 *Programmer une fonction Python prenant comme paramètre la liste d'adjacence G, la liste des noeuds de W et renvoyant la liste des noeuds de $\mathcal{A}$, pour le joueur a.*

```
def attracteur(G)
     A=[]
     tG=transpose(G)
     degG={}
     for v in G:
          degG[v]=len(G[v])
     def parcour(u):
          if u not in A:
             A.append(u)
          if u in tG:
             for v in tG[u]:
                  degG[v]-=1
                  if v[0]='Va' or degG[v]==0:
                        parcour(v)

     for u in W:
         parcour(u)
     return A
```

On peut programmer en directe la fonction parcour ainsi :

```
def parcour(u,A,degG,GT):
   if u in GT:
      for v in GT[u]:
          if v not in A:
              degG[v]-=1
              if v[0]=='Va' or degG[v]==0:
                   A.append(v)
                   if v in GT:
                        attente.append(v)
    while len(attente)!=0:
             v=attente.pop()
             for w in GT[v]:
                  if w not in A:
                       degG[w]-=1
                       if w[0]=='Va' or degG[w]==0:
                                 A.append(w)
                                 if w in GT
                                       attente.append(w)
```

Exercice 1.4 *Programmer une fonction* `phi(G,u)` *renvoyant une stratégie gagnante, c-a-d , qui renvoie pour le graphe G, d'attracteur A, le noeud $\phi(u)$ que doit joueur le joueur A.*

```
def phi(G,u):
      A=attracteur(G)
      for v in G[u]:
           if v in A:
              return v
       return False
```

On peut faire un peu mieux en créant le dictionnaire de la stratégie :

```
def phi(G):
      Phi={}
      A=attracteur(G)
```

```
    for v in G:
        if v in A:
            for u in G[v]:
                if u in A:
                    Phi[v]=u
    return Phi
```