Attracteurs dans un jeu à deux joueurs. Corrigé

Question 1

Les sommets gagnants pour 1 sont 0 et 1. Les sommets gagnants pour 2 sont 2, 3 et 4. En effet :

- si $s_0 = 0$, toute partie est gagnée par 1

- si $s_0 = 1$, en posant $f(1) = 0$, toute f-partie est gagnée par 1

- si $s_0 = 2$ ou $s_0 = 3$, en posant $f(2) = 3$, toute f-partie est gagnée par 2

- si $s_0 = 4$, toute partie est gagnée par 2.

Question 2

On suppose par l'absurde que $R_1 \cap R_2$ n'est pas vide et on considère $s_0 \in R_1 \cap R_2$.

On pose f_1 et f_2 des stratégies gagnantes pour 1 et 2 respectivement.

On définit la partie σ par $s_0 \in R_1 \cap R_2$ et $s_{i+1} = \begin{cases} f_1(s_i) \text{ si } s_i \in S_1 \\ f_2(s_i) \text{ si } s_i \in S_2 \end{cases}$

Alors σ est gagnante pour 1 et pour 2 à la fois, ce qui est absurde, puisqu'une partie ne peut pas à la fois par un sommet de T et éviter tout sommet de T ...

Par définition, lorsque $s \notin R_1$, aucun successeur de s n'est dans R_1 si $s \in S_1$ et il existe un successeur de s qui n'est pas dans R_1 si $s \in S_2$. Ainsi, si $s_0 \notin R_1$, il existe une stratégie gagnante pour le joueur 2 donnant une partie ne passant par aucun sommet de R_1, donc a foritori de T.

Donc $R_1 \cup R_2 = S$.

Question 3

Par définition, si $\sigma = (s_i)_{i\in\mathbb{N}}$ est une partie, alors $(s_i)_{0\leq i\leq k}$ est un chemin de s_0 à s_k.

De plus, $s \in S$ est gagnant pour 1 si et seulement s'il existe un chemin de s à un sommet de T .

Par définition de G^T, les sommets gagnants sont les sommets accessibles depuis T dans G^T.

Question 4

```
def transpose(G):
    n = len(G)
    GT = [[] for k in range(n)]
    for s in range(n):
        for t in G[s]:
            GT[t].append(s)
    return GT
```

La complexité est bien celle attendue car elle correspond à $O\left(\sum_{x\in S}(1 + \deg(x))\right) = O(n + m)$.

Question 5

```
def parcours(G,T):
    GT = transpose(G)
    n = len(GT) ; visites = [False] * len(GT)
    L = []
```

```
def traite(s):
    if not visites[s] :
        visites[s] = True
        L.append(s)
        for t in GT[s]:
            traite(t)
for s in T :  traite(s)
return L
```

Comme dans la fonction précédente, le marquage des sommets garantit qu'on ne parcourt chaque liste d'adjacence qu'au plus une fois, et on y fait des opérations en temps constant pour chaque élément, d'où une complexité $O(n+m)$.

Question 6

La propriété est immédiate pour $i = 0$.

Supposons la propriété vraie au rang i. Soit $s \in X(i+1)$. Le résultat est immédiat si $s \in X(i)$.

Supposons désormais $s \notin X(i)$. Si $s \in S_1$, alors il existe $t \in V(s) \cap X(i)$.

Si le joueur 1 joue ce premier coup (c'est-à-dire $f(s) = t$), on est ramené à $t \in X(i)$, d'où le résultat.

Si $s \in S_2$, alors le joueur 2, quel que soit son jeu, amène à un état $t \in X(i)$, d'où le résultat.

Question 7

Par définition, la suite $(X(i))_{i \in \mathbb{N}}$ est croissante pour l'inclusion.

Considérons $k = \min\{i \in \mathbb{N} \mid X(i+1) = X(i)\}$.

On a $\forall i \leq k$, $\operatorname{card} X(i+1) \geq 1 + \operatorname{card} X(i)$, d'où $\operatorname{card} X(k+1) \geq k+1$, et ainsi $k \leq n$.

On a alors $X(k) = X(k+1)$, et par récurrence immédiate, $\forall i \geq k$, $X(i) = X(k)$.

Question 8

Supposons $s \in R_1$. Alors il existe $i \in \mathbb{N}$ tel que $s \in X(i)$. Donc s est gagnant pour le joueur 1.

Supposons désormais $s \in R_2$.

Supposons par l'absurde que s est gagnant pour le joueur 1 : il existe une stratégie conduisant à un sommet de T. Or, toute partie gagnante pour le joueur 1 admet au plus n étapes avant d'atteindre un sommet de T. En effet, sinon, elle passerait deux fois par un même sommet, d'où l'existence possible d'un cycle et d'une partie infinie non gagnante. Donc $s \in X(n)$, d'où une contradiction.

Question 9

a)
```
def attracteur(G,S1,T):
    n = len(G)
    visites = [False] * n
    X = T.copy()
    for i in range(n):
        for s in range(n):
```

```
            if not visites[s]:
                b1 = False
                b2 = True
                for t in G[s]:
                    b1 = b1 or visites[t]
                    b2 = b2 and visites[t]
                if (S1[s] and b1) or ((not S1[s]) and b2):
                    visites[s] = True
                    X.append(s)
    return X
```

b) `tabAttract[t]` donne le nombre de successeurs appartenant à X (lequel est construit progressivement en passant par la pile qui permet de mettre à jour le tableau de marquage et le tableau `tabAttract`).

Un sommet non encore visité est ajouté à la pile

- soit s'il appartient à S_1 et qu'un de ses successeurs appartient à X

- soit s'il appartient à S_2 et que tous ses successeurs appartiennent à X.

```
def attracteur(G, S1, T):
    n = len(G)
    visites = [False] * n
    X = []
    GT = transpose(G)
    tabAttract = [0] * n
    def traite(s):
        if not visites[s] :
            visites[s] = True
            X.append(s)
            for t in GT[s] :
                tabAttract[t] += 1
                if S1[s] or (not S1[s] and tabAttract[t] == len(G[s]) :
                    traite(t)
    for s in T : traite(s)
```

Question 10

a) La solution contient une boucle de taille n dans laquelle on parcourt exactement une fois chaque liste d'adjacence. La complexité est donc $O(n(n+m))$.

b) La solution correspond à la complexité d'un parcours de graphe (on ne parcourt la liste d'adjacence d'un sommet qu'une seule fois), ce qui donne une complexité $O(n+m)$.