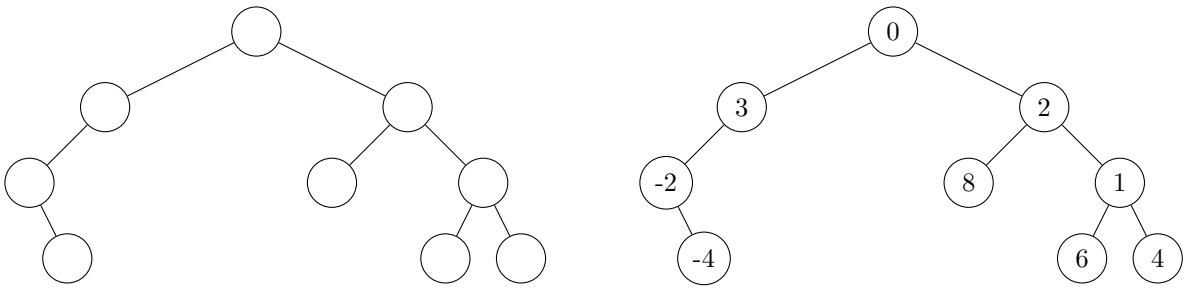
Exercice 1

Minima locaux dans des arbres (CCINP 2023)

Dans cet exercice, on considère des arbres binaires étiquetés par des entiers relatifs deux à deux distincts. Un nœud est un minimum local d'un arbre si son étiquette est plus petite que celle de son éventuel père et celles de ses éventuels fils. Considérons par exemple l'étiquetage (b) de l'arbre binaire non étiqueté (a) :



(a) (b)

1. Déterminer le ou les minima locaux de l'arbre (b).
2. Donner une définition inductive permettant de définir les arbres binaires ainsi que la définition de la hauteur d'un arbre. Quelle est la hauteur de l'arbre (b) ?
3. Montrer que tout arbre non vide possède un minimum local.
4. Proposer un algorithme permettant de trouver un minimum local d'un arbre non vide et déterminer sa complexité.

On considère un arbre binaire non étiqueté que l'on souhaite étiqueter par des entiers relatifs distincts deux à deux de manière à maximiser le nombre de minima locaux de cet arbre.

5. Proposer sans justifier un étiquetage de l'arbre (a) qui maximise le nombre de minima locaux.
6. Proposer un algorithme qui, étant donné un arbre binaire non étiqueté en entrée, permet de calculer le nombre maximal de minima locaux qu'il est possible d'obtenir pour cet arbre. Déterminer la complexité de votre algorithme.
7. Montrer que, pour un arbre de taille $n \in \mathbb{N}$, le nombre maximal de minima locaux est majoré par $\left\lfloor \frac{2n+1}{3} \right\rfloor$. *On pourra remarquer que les nœuds non minimaux couvrent l'ensemble des arêtes de l'arbre.*

Corrigé

1. Les nœuds d'étiquettes -4, 0 et 1 sont les trois minima locaux.

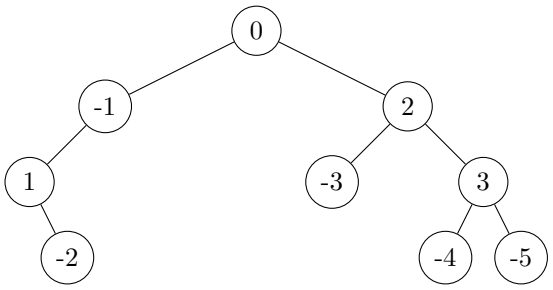
   Jury : On attend simplement du candidat qu'il propose les minima trouvés sans justification. Une réponse orale suffit. En cas d'erreur, le candidat est invité à expliquer son raisonnement.
2. Un arbre est soit un arbre vide soit un nœud formé d'une étiquette et de deux sous-arbres. La hauteur est la profondeur maximale d'une feuille, c'est-à-dire la longueur maximale d'un chemin de la racine à une feuille. La hauteur de l'arbre (b) est 3.

   Jury : Dans le cas où le candidat propose une définition inductive des arbres avec une feuille pour cas de base, il est guidé vers une définition dont le cas de base est l'arbre vide de sorte à rendre compte du fait que les arbres considérés ne sont pas binaires stricts.
3. L'arbre possède un nombre fini non vide d'étiquettes et donc une étiquette de valeur minimale, qui est un minimum global et donc local.

   Jury : Plusieurs stratégies sont ici acceptées (preuve par induction ou existence d'un minimum global qui est local par exemple).

4. Si la racine de l'arbre est un minimum local on a trouvé notre minimum local. Sinon, un des deux fils est non vide, avec une étiquette à sa racine plus petite que celle de la racine de l'arbre. Un appel récursif permet d'obtenir un minimum local de ce sous-arbre, qui est également un minimum local de l'arbre (que ce soit la racine du sous-arbre ou un descendant strict). La complexité est linéaire en la hauteur de l'arbre.

   Jury : Si le candidat propose une solution linéaire en la taille de l'arbre, il est guidé vers une solution linéaire en la hauteur.
5. On propose l'étiquetage à la figure (c) dans lequels les 5 minima locaux sont étiquetés par des entiers strictement négatifs.



(c)

   Proposer un étiquetage correct suffit à obtenir tous les points.
6. On propose une approche récursive qui pour un arbre a en entrée calcule $m(a)$ le nombre maximal de nœuds qui peuvent être des minima locaux dans un étiquetage de a, ainsi que, en même temps, la quantité $m_-(a)$ correspondant à cette même quantité mais en supposant de plus que la racine n'est pas un minimum local. Pour un arbre vide, ces deux valeurs valent 0. Pour un arbre a de fils gauche f_g et de fils droit f_d, on peut obtenir par appels récursifs les quantités $m(f_g)$, $m_-(f_g)$, $m(f_d)$ et $m_-(f_d)$. On a alors $m_-(a) = m(f_g) + m(f_d)$ et $m(a) = \max\{m_-(a), 1 + m_-(f_g) + m_-(f_d)\}$. La complexité est linéaire en la taille de l'arbre, chaque nœud est visité exactement une fois avec un nombre d'opérations constant.

   Jury : Le jury attend une complexité linéaire. Des indications peuvent être apportées pour aiguiller le candidat vers une telle solution.
7. Le résultat est vrai pour $n = 0$ et on peut donc supposer que $n \geqslant 1$. Considérons un étiquetage et notons X l'ensemble des nœuds qui sont des minima locaux et Y ceux qui ne le sont pas. On remarque que deux nœuds adjacents ne peuvent pas être tous les deux des minima locaux, puisque toutes les étiquettes sont deux à deux distinctes. Ainsi toute arête de l'arbre est extrémité d'au moins un nœud de Y et l'ensemble Y couvre donc toutes les arêtes. Comme chaque nœud de Y est incident à au plus 3 arêtes et qu'il y a exactement $n-1$ arêtes dans l'arbre, il faut au moins $\frac{n-1}{3}$ nœuds pour couvrir toutes les arêtes, c'est-à-dire $|Y| \geqslant \frac{n-1}{3}$. On a donc $|X| = n - |Y| \leqslant n - \frac{n-1}{3} = \frac{2n+1}{3}$ et donc $|X| \leqslant \left\lfloor \frac{2n+1}{3} \right\rfloor$.

   Jury : Très peu de candidats ont pu aborder cette question.

Exercice 2

On considère l'alphabet $\Sigma = \{a, b, c\}$ et L l'ensemble des mots constitués d'un nombre strictement positif de a, suivis d'un c, suivi d'un mélange quelconque de a et b.

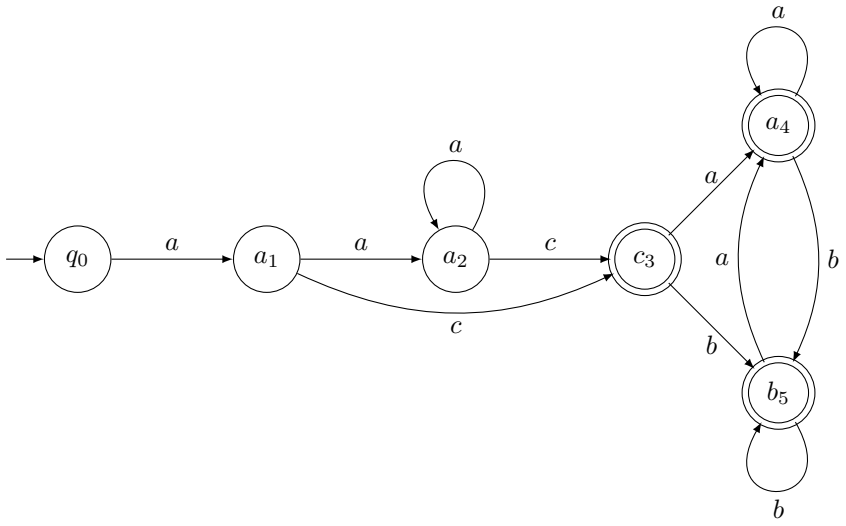
1. Proposer une expression régulière r décrivant le langage L.
2. Construire l'automate de Glushkov associé à l'expression régulière r.
3. L'automate précédent est-il déterministe ? La réponse à cette question est-elle toujours la même si on considère une autre expression régulière (pas nécessairement associée au langage L) ? Justifier.

4. Rappeler l'énoncé du lemme de pompage (appelé aussi lemme de l'étoile).
5. Montrer que le langage $L' = \{uu \mid u \in \{a,b\}^*\}$ n'est pas rationnel.
6. Le langage $L'' = \{uuv \mid u, v \in \{a,b\}^*\}$ est-il rationnel ? Justifier. Même question pour le langage $L''' = \{uvuw \mid u \in \{a,b\}^*, v, w \in \{c\}^*\}$

Corrigé

1. On propose $r = aa^\star c(a|b)^\star$.
2. On commence par linéariser r en $r' = a_1 a_2^\star c_3 (a_4|b_5)^\star$. On calcule ensuite :
   - $P(r') = \{a_1\}$,
   - $S(r') = \{c_3, a_4, b_5\}$,
   - $F(r') = \{a_1a_2, a_1c_3, a_2a_2, a_2c_3, c_3a_4, c_3b_5, a_4a_4, a_4a_5, b_5a_4, b_5b_5\}$.

   On obtient donc l'automate :



3. L'automate précédent est bien déterministe. Ce n'est pas nécessairement le cas pour tout automate de Glushkov. Par exemple, si l'expression avait été $aa^\star(c|\varepsilon)(a|b)^\star$, on aurait eu un automate similaire, où a_2 aurait été final, et avec des transitions de a_1 et a_2 vers a_4 et b_5, étiquetées par a (vers a_4) et b (vers b_5). L'automate ainsi obtenu n'aurait pas été déterministe.
4. Soit $L \in \mathrm{Rat}(\Sigma)$. Alors il existe un entier $n \in \mathbb{N}^*$ tel que tout mot $u \in L$ tel que $|u| \geqslant n$ admet une décomposition de la forme xyz telle que :
   - $y \neq \varepsilon$;
   - $|xy| \leqslant n$;
   - $\forall k \in \mathbb{N}, xy^k z \in L$ ou, de manière équivalente, $xy^\star z \subseteq L$.
5. Supposons que L' est rationnel. Considérons n la longueur de pompage donnée par le lemme, et posons $u = a^n b a^n b$. Par le lemme, on peut écrire $u = xyz$ vérifiant les trois conditions du lemme. Les deux premières impliquent que $xy \in a^*$ et $y = a^k$, $k > 0$. Dès lors, $xz = a^{n-k}ba^n b$ ne peut pas s'écrire comme vv avec $v \in \{a,b\}^*$. En effet, si k est impair, alors xz est de longueur impaire. Si k est pair, alors $k \geqslant 2$ et on aurait $v = a^{n-k}ba^{k/2} = a^{n-k/2}b$ ce qui est absurde. On conclut par l'absurde que L' n'est pas rationnel.
6. On remarque que $L'' = \{a,b\}^*$ est bien un langage rationnel. En effet, tout mot $v \in \{a,b\}^*$ peut s'écrire $v = \varepsilon\varepsilon v$ et $\varepsilon \in \{a,b\}^*$.

   On remarque que $L''' \cap \{a,b\}^* = L'$. Si L''' était rationnel, alors L' le serait aussi, car les langages rationnels sont clos par intersection. On conclut que L''' n'est pas rationnel.

Exercice 3: L'exercice suivant est à traiter dans le langage C

Dans l'ensemble de l'exercice, on considère un tableau d'entiers $t = [t_0, t_1, \ldots, t_{n-1}]$.

1. Écrire une fonction `int somme(int* t, int i)` telle que `somme(t, i)` calcule la somme partielle $\sum_{k=0}^{i} t_k$ des valeurs du tableau t entre les indices 0 et i inclus.

Un tableau t d'éléments de $[\![0, n-1]\!]$ est dit *autoréférent* si pour tout $0 \leqslant i < n$, t_i est exactement le nombre d'occurrences de i dans t, c'est-à-dire que :

$$\forall i \in [\![0, n-1]\!], t_i = |\{k \in [\![0, n-1]\!] \mid t_k = i\}|$$

Ainsi, par exemple, pour $n = 4$, le tableau suivant est autoréférent :

i	0	1	2	3
t_i	1	2	1	0

En effet, la valeur 0 existe en une occurrence, la valeur 1 en deux occurrences, la valeur 2 en une occurrence et la valeur 3 n'apparaît pas dans t.

2. Justifier qu'il n'existe pas de tableau autoréférent pour $n \in [\![1, 3]\!]$. Déterminer un autre tableau autoréférent pour $n = 4$.
3. Écrire une fonction `bool est_auto(int* t, int n)` qui prend en argument un tableau et sa taille et vérifie s'il est autoréférent. On demande une complexité temporelle en $\mathcal{O}(n)$.

On propose d'utiliser une méthode de retour sur trace (*backtracking*) pour trouver tous les tableaux autoréférents pour un $n > 0$ donné. On se donne pour cela une fonction `void gen_auto(int n)` qui affiche tous les tableaux autoréférents d'une taille n donnée en argument :

```
void remplir_a_partir_de(int* t, int i, int n){
    if (i == n){
        if (est_auto(t, n)){ afficher_tab(t, n); }
        return;
    }
    for (int k=0; k<n; k++){
        t[i] = k;
        // Élagage
        remplir_a_partir_de(t, i + 1, n);
    }
}

void gen_auto(int n){
    int* t = malloc(n * sizeof(int));
    remplir_a_partir_de(t, 0, n);
    free(t);
}
```

Pour accélérer la recherche, il faut élaguer l'arbre (repérer le plus rapidement possible qu'on se trouve dans une branche ne pouvant pas donner de solution). Pour chaque stratégie proposée dans les questions qui suivent, on détaillera comment la mettre en œuvre dans le code précédent.

4. Que peut-on dire de la somme des éléments d'un tableau autoréférent ? En déduire une stratégie d'élagage pour accélérer la recherche.
5. Que peut-on dire si, juste après avoir affecté la case t_i, il y a déjà strictement plus d'occurrences d'une valeur $0 \leqslant k \leqslant i$ que la valeur de t_k ? En déduire une stratégie d'élagage supplémentaire.
6. Après avoir affecté la case t_i, combien de cases reste-t-il à remplir ? Combien de ces cases seront complétées par une valeur non nulle ? À quelle condition est-on alors certain que la somme dépassera la valeur maximale possible à la fin ? En déduire une stratégie d'élagage supplémentaire.

7. Montrer qu'il existe un tableau autoréférent pour tout $n \geqslant 7$. À titre d'exemple, pour $n = 8$, le tableau suivant est autoréférent :

i	0	1	2	3	4	5	6	7
t_i	4	2	1	0	1	0	0	0

On peut également montrer qu'il y a unicité, mais ce n'est pas demandé ici.

Corrigé

1. Pas de difficulté pour cette question :

```
int somme(int* t, int i){
    // On suppose ici que i < n, la taille du tableau t.
    int s = 0;
    for (int k=0; k<=i; k++){
        s += t[k];
    }
    return s;
}
```

2. On distingue les cas, en supposant à chaque fois que t est un tableau autoréférent :
   - pour $n = 1$, si $t_0 = 0$, alors on devrait avoir $t_0 > 0$ (absurde) ; sinon, si $t_0 > 0$, alors il n'y a pas de 0 dans le tableau, donc $t_0 = 0$ (absurde à nouveau) ;
   - pour $n = 2$, comme précédemment, $t_0 > 0$. On en déduit $t_1 = 0$ (sinon il n'y a pas de zéro dans le tableau), soit $t_0 = 1$, ce qui est absurde, parce qu'alors $t_1 = 1$.
   - pour $n = 3$, à nouveau $t_0 > 0$. On peut alors avoir $t_0 = 1$ ou $t_0 = 2$. Si $t_0 = 2$, alors $t_1 = t_2 = 0$ (sinon il n'y a pas assez de zéros dans le tableau), mais alors $t_2 = 1$, ce qui est absurde. Sinon, $t_0 = 1$, et alors $t_1 > 0$, donc $t_2 = 0$. Mais alors, si $t_1 = 1$, il y a deux 1 dans le tableau (absurde), et si $t_1 = 2$, on aurait $t_2 \neq 0$ (absurde).

   Comme autre tableau autoréférent, on propose le tableau suivant :

i	0	1	2	3
t_i	2	0	2	0

3. On crée un nouveau tableau pour compter le nombre d'occurrences de chaque élément, puis on vérifie que ce tableau est égal au tableau donné en entrée. Chaque boucle est de taille n et fait des opérations en temps constant, d'où la complexité.

```
bool est_auto(int* t, int n){
    int* occ = malloc(sizeof(int) * n);
    for (int i=0; i<n; i++){
        occ[i] = 0;
    }
    for (int i=0; i<n; i++){
        if (t[i] < 0 || t[i] >= n){
            free(occ);
            return false;
        }
        occ[t[i]]++;
    }
    for (int i=0; i<n; i++){
        if (t[i] != occ[i]){
            free(occ);
            return false;
        }
    }
    free(occ);
    return true;
}
```

4. La somme des éléments d'un tableau autoréférent correspond à la somme du nombre d'occurrences d'éléments du tableau, c'est-à-dire exactement n. Ainsi, si la somme dépasse n strictement jusqu'à un indice donné, on peut arrêter l'exploration et élaguer la branche. Il suffit de rajouter :

```
        if (somme(t, i) > n){ return; }
```

5. S'il y a plus d'occurrences de $0 \leqslant k \leqslant i$ que t_k, il est impossible de compléter le tableau correctement. On peut donc compter en parallèle le nombre d'occurrences de chaque élément dans le tableau (jusqu'à l'indice i), et arrêter le calcul si le cas précédent se produit. Pour l'implémentation, on rajoute en argument un tableau d'occurrences qu'on initialise à 0, et qu'on modifiera au fur et à mesure. On en profite également pour rajouter une variable de somme, pour rendre le test rajouté précédemment faisable en temps constant. La fonction `gen_auto` devient :

```
void gen_auto(int n){
    int* t = malloc(n * sizeof(int));
    int* occ = malloc(sizeof(int) * n);
    int som = 0;
    for (int i=0; i<n; i++){
        occ[i] = 0;
    }
    remplir_a_partir_de(t, occ, &som, 0, n);
    free(occ);
    free(t);
}
```

Dès lors, on modifie `remplir_a_partir_de` avec :

```c
void remplir_a_partir_de(int* t, int* occ, int* som, int i, int n){
    if (i == n){
        if (est_auto(t, n)) {
            afficher_tab(t, n);
        }
        return;
    }
    for (int k=0; k<n; k++){
        t[i] = k;
        occ[k]++;
        *som += k;
        if (*som > n){
            occ[k]--;
            *som -= k;
            return;
        }
        if (verif_occ(t, i)){
            remplir_a_partir_de(t, occ, som, i + 1, n);
        }
        occ[k]--;
        *som -= k;
    }
}
```

On note qu'avant de passer à l'itération suivante de la boucle ou d'arrêter la fonction, on s'assure que l'état de `occ` et `*som` est inchangé.

6. Après avoir affecté t_i, il reste $n-i-1$ cases à remplir. En notant n_0 le nombre de cases affectées à 0 parmi les indices $0, \ldots, i$, on en déduit qu'il reste $t_0 - n_0$ cases parmi les restantes qui doivent valoir 0, donc $n + n_0 - i - 1 - t_0$ qui sont $\geqslant 1$. Si on garde en mémoire n_0, on peut arrêter les calculs si $s + n_0 - i - 1 - t_0 > 0$. On exprime cela comme :

```c
        if (*som > n || *som + occ[0] - i - 1 - t[0] > 0){
            occ[k]--;
            *som -= k;
            return;
        }
```

7. Pour $n \geqslant 7$, en posant :
   - $t_0 = n - 4$;
   - $t_1 = 2$;
   - $t_2 = 1$;
   - $t_{n-4} = 1$;
   - $t_k = 0$ pour $k \notin \{0, 1, 2, n-4\}$.

   Alors on obtient un tableau autoréférent.

Exercice 4: Théorie de Sprague-Grundy

Un **jeu** est un triplet (S, A, s_0) où S est un ensemble fini d'**états**, $A \subseteq S^2$ est un ensemble de **transitions** et $s_0 \in S$ est un **état initial**, tels que le graphe (S, A) est acyclique.

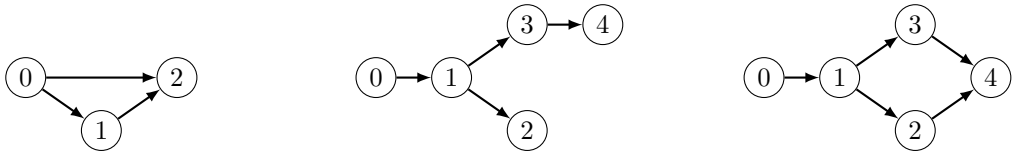
Une **stratégie** est une fonction partielle $\varphi : S \rightharpoonup S$ telle que pour tout état s où elle est définie, $(s, \varphi(s)) \in A$. On peut faire jouer deux stratégies φ_0 et φ_1 l'une contre l'autre en les faisant jouer alternativement. On définit ainsi la séquence d'états $s_1 = \varphi_0(s_0)$, $s_2 = \varphi_1(s_1)$, $s_3 = \varphi_0(s_2)$, …, $s_k = \varphi_{(k-1) \bmod 2}(s_{k-1})$.

1. Montrer que la séquence (s_k) est finie, c'est-à-dire qu'il existe un entier n_f minimal tel que $\varphi_{n_f \bmod 2}(s_{n_f})$ n'est pas défini.

Pour deux stratégies φ_0 et φ_1 jouées par les joueurs 0 et 1, le joueur perdant est le premier joueur pour lequel la stratégie n'est plus définie. Formellement, si n_f est pair, alors le joueur 0 perd et le joueur 1 gagne. Inversement, si n_f est impair, alors le joueur 1 perd et le joueur 0 gagne.

Une **stratégie gagnante** pour le joueur $i \in \{0, 1\}$ est une stratégie qui garantit que le joueur i gagne quand il joue suivant cette stratégie, quelle que soit la stratégie jouée par l'autre joueur.

2. Parmi les jeux suivants (où l'état initial est le sommet 0), quels sont ceux qui ont une stratégie gagnante pour le joueur 0 ? Pour le joueur 1 ?



3. On définit le jeu suivant : il y a un tas de n jetons entre les deux joueurs. Chacun son tour, un joueur peut prendre $1, 2, 3$ ou 4 jetons. Le joueur qui prend le dernier jeton gagne. Expliquer comment cette description informelle du jeu peut se formaliser avec la notion formelle de jeu définie plus haut. Sous quelle condition sur n existe-t-il une stratégie gagnante pour le joueur 0 ? Pour le joueur 1 ?

Pour un jeu (S, A, s_0), on définit, $s \in S$, sa **valeur de Grundy** $G(s) \in \mathbb{N}$ par :

$$G(s) = \min(\mathbb{N} \setminus \{G(t) \mid (s, t) \in A\})$$

4. Justifier que cette relation définit bien G de manière unique.
5. Donner une condition nécessaire et suffisante sur G pour que le joueur 0 ait une stratégie gagnante. Qu'en est-il du joueur 1 ?
6. Donner le pseudo-code d'un algorithme permettant de déterminer, étant donné un jeu, quel(s) joueur(s) a(ont) une stratégie gagnante. Quelle est sa complexité en temps et en espace ?

Dans le **jeu de Nim**, il y a n tas de $t_0, \ldots, t_{n-1}$ jetons chacun entre les deux joueurs. Chacun son tour, un joueur choisit un tas et en retire autant de jetons qu'il le souhaite. Le joueur qui prend le dernier jeton gagne.

7. Montrer qu'il existe une stratégie gagnante pour le joueur 0 si et seulement si $t_0 \oplus \ldots \oplus t_{n-1} \neq 0$, où $\oplus$ représente le ou-exclusif bit à bit (ou l'addition modulo 2, bit à bit).

Pour $J_1 = (S_1, A_1, s_1)$ et $J_2 = (S_2, A_2, s_2)$, leur **somme** $J_1 + J_2 = (S, A, s_0)$ est définie par :

- $S = S_1 \times S_2$;
- $A = \{((s, t), (s, t')) \mid s \in S_1, (t, t') \in A_2\} \cup \{((s, t), (s', t)) \mid (s, s') \in A_1, t \in S_2\}$;
- $s_0 = (s_1, s_2)$.

8. Soit (s, t) un état de $J_1 + J_2$. Montrer que $G(s, t) = G_1(s) \oplus G_2(t)$, où G, G_1 et G_2 désignent les valeurs de Grundy dans $J_1 + J_2$, J_1 et J_2 respectivement.
9. Quelle est la valeur de Grundy des états du jeu de Nim ? Commenter.

Corrigé

1. Si (s_k) n'est pas finie, alors par le principe des tiroirs, il existe $i < j$ tels que $s_i = s_j$. Cela signifie que le graphe contient un cycle par définition des stratégies. C'est contradictoire avec l'hypothèse. On en déduit par ailleurs que $n_f \leqslant |S|$.
2. On commence par signaler qu'il ne peut pas exister de stratégie gagnante simultanément pour les

deux joueurs pour un même graphe.

- pour le premier graphe, $\varphi_0 : 0 \mapsto 2$ est une stratégie gagnante pour le joueur 0 ;
- pour le deuxième graphe, $\varphi_1 : 1 \mapsto 2$ est une stratégie gagnante pour le joueur 1 ;
- pour le troisième graphe, $\varphi_0 : 0 \mapsto 1$, $2 \mapsto 4$, $3 \mapsto 4$ est une stratégie gagnante pour le joueur 0.

3. On peut modéliser ce jeu par $J = (S, A, s_0)$ où :
   - $S = [\![0, n]\!]$;
   - $s_0 = n$;
   - $A = \{(s, s-i) \mid s \in S, i \in [\![1, 4]\!], s - i \geqslant 0\}$.

   Pour déterminer une stratégie gagnante, le mieux est de commencer par étudier de petites valeurs de n :
   - si $n \leqslant 4$, il est clair que 0 a une stratégie gagnante (prendre tous les jetons en un seul coup) ;
   - si $n = 5$, c'est 1 qui a une stratégie gagnante, car quel que soit le coup de 0, on se ramène au cas précédent en inversant les joueurs ;
   - si $n \in [\![6, 9]\!]$, 0 a une stratégie gagnante, en prenant un nombre de jetons pour ramener le total à 5, donc au cas précédent en inversant les joueurs ;
   - ...

   On en déduit (et cela peut se montrer par récurrence) que 1 a une stratégie gagnante si n est un multiple de 5, sinon c'est 0 qui a une stratégie gagnante.
4. Soit $s \in S$. Montrons par récurrence sur k, la longueur maximale d'un chemin dans (S, A) de sommet initial s, que $G(s)$ est défini de manière unique :
   - si $k = 0$, alors s est un puits (il n'a pas de voisin), donc $G(s) = 0$ est bien défini de manière unique ;
   - supposons le résultat établi jusqu'à $k \in \mathbb{N}$ fixé. Soit s ayant une longueur maximale de chemin sortant égale à $k+1$. Alors chaque voisin t de s a une longueur maximale de chemin sortant $\leqslant k$, donc $G(t)$ est défini de manière unique, donc $G(s)$ est défini de manière unique (minimum d'un ensemble non vide).

   On conclut par récurrence finie, car le graphe est acyclique.
5. Le joueur 0 a une stratégie gagnante si et seulement si $G(s_0) \neq 0$. Montrons ce résultat par récurrence sur k, la longueur maximale d'un chemin sortant de s_0 :
   - si $k = 0$, alors s_0 est un puits et $G(s_0) = 0$, et 0 n'a effectivement pas de stratégie gagnante ;
   - supposons le résultat établi jusqu'à $k \in \mathbb{N}$ fixé. Supposons que la longueur maximale d'un chemin sortant de s_0 soit $k+1$ et distinguons :
     * si $G(s_0) = 0$, alors chaque voisin t de s_0 vérifie $G(t) \neq 0$ (sinon $G(s_0)$ ne pourrait pas valoir 0). Par hypothèse de récurrence, il existe une stratégie gagnante pour le premier joueur depuis t. Ainsi, quel que soit le coup de 0, 1 a une stratégie gagnante, donc 0 n'a pas de stratégie gagnante depuis s_0.
     * si $G(s_0) \neq 0$, alors il existe un voisin t de s_0 tel que $G(t) = 0$ (sinon on aurait $G(s_0) = 0$). Par hypothèse de récurrence, il n'existe pas de stratégie gagnante pour le premier joueur depuis t. Ainsi, il existe une stratégie gagnante depuis s_0 pour le joueur 0, vérifiant $\varphi_0(s_0) = t$.

   On conclut par récurrence. De même, on en déduit que 1 a une stratégie gagnante si et seulement si $G(s_0) = 0$.
6. Il s'agit d'implémenter un parcours de graphe depuis s_0. L'algorithme ressemble également à l'algorithme glouton de coloration de graphe (selon l'ordre inverse d'un ordre topologique).

```
Entrée : Jeu J = (S, A, s_0), n = |S|.
Début algorithme
    G ← tableau de taille n contenant des −1.
    Fonction Calcul_G(s)
        Si G[s] = −1 Alors
            m ← 0.
            Pour t voisin de s Faire
                Calcul_G (t)
                m ← m + 1.
            T ← tableau de taille m + 1 contenant des Faux.
            Pour t voisin de s Faire
                T[G[t]] ← Vrai.
            Pour i de 0 à m Faire
                Si T[i] Alors
                    G[s] ← i.
                    Sortir de la boucle.
    Calcul_G (s_0).
    Si G[s_0] ≠ 0 Alors
        Renvoyer 0.
    Sinon
        Renvoyer 1.
```

À noter, la création de T et les deux boucles qui suivent permettent de trouver le plus petit entier naturel absent des nombres de Grundy des voisins en temps linéaire. Cela permet d'avoir une complexité temporelle totale en $\mathcal{O}(|S| + |A|)$ et une complexité spatiale en $\mathcal{O}(|S|)$.

7. Montrons ce résultat par récurrence sur le nombre total k de jetons restant. Appelons **score** d'une configuration la valeur $t_0 \oplus \ldots \oplus t_{n-1}$.
   - si $k = 0$, alors tous les $t_i = 0$, donc le score est nul et 0 n'a pas de stratégie gagnante ;
   - supposons le résultat établi pour $k \in \mathbb{N}$ fixé et soit un jeu de Nim à $k+1$ jetons. Distinguons :
     * si le score σ est nul, alors quel que soit i, si 0 prend $k_i > 0$ jetons dans le tas i, il reste $k + 1 - k_i \leqslant k$ jetons, et en faisant le ou-exclusif entre l'ancien score σ et le nouveau score σ', on obtient $\sigma \oplus \sigma' = t_i \oplus (t_i - k_i) \neq 0$. Comme $\sigma \oplus \sigma' = 0 \oplus \sigma' = \sigma'$, on en déduit que $\sigma' \neq 0$. Par hypothèse de récurrence, il existe une stratégie gagnante pour le joueur suivant ;
     * si le score σ est non nul, soit d la position du bit de poids fort dans σ. Il existe $i \in [\![0, n-1]\!]$ tel que le d-ème bit de t_i est différent de 0. En en déduit que $\sigma \oplus t_i < t_i$. Posons alors $k_i = t_i - \sigma \oplus t_i$. En prenant k_i jetons dans le tas i, alors le nouveau score σ' vaut $\sigma' = \sigma \oplus t_i \oplus (t_i - k_i) = \sigma \oplus t_i \oplus (\sigma \oplus t_i) = 0$. On en déduit par hypothèse de récurrence qu'il n'y a pas de stratégie gagnante pour le joueur suivant.

   On conclut par récurrence.

8. Pour $s \in S_1$, notons $\ell_1(s)$ la longueur maximal d'un chemin partant de s (et de même $\ell_2(t)$ pour $t \in S_2$).

   Montrons l'égalité par récurrence sur $\ell_1(s) + \ell_2(t)$:
   - si $\ell_1(s) + \ell_2(t) = 0$, alors ni s, ni t n'a de voisin, donc $G(s,t) = 0 = G_1(s) \oplus G_2(t)$;
   - supposons le résultat établi jusqu'à $\ell_1(s) + \ell_2(t) = k \in \mathbb{N}$ fixé et soit $(s,t) \in S$ tels que $\ell_1(s) + \ell_2(t) = k + 1$.
     * Soit $x < G_1(s) \oplus G_2(t)$. Montrons qu'il existe (s', t') un voisin de (s,t) tel que $G(s',t') = x$. Posons $y = G_1(s) \oplus G_2(t) \oplus x$. On a donc $y \neq 0$. Soit d la position du bit de poids fort de y. Sachant que $x < G_1(s) \oplus G_2(t)$, le d-ème bit dans $G_1(s) \oplus G_2(t)$ vaut 1 (et 0 dans x). Sans perte de généralité, le d-ème bit de $G_1(s)$ vaut donc 1. On en déduit que $G_1(s) \oplus y < G_1(s)$, donc il existe par définition un voisin s' de s tel que

$G_1(s) = G_1(s) \oplus y$. Dès lors, (s', t) est un voisin de (s, t), et par hypothèse de récurrence, $G(s', t) = G_1(s') \oplus G_2(t) = G_1(s) \oplus G_2(t) \oplus y = x$.

* Soit (s', t') un voisin de (s, t). Montrons que $G(s', t') \neq G_1(s) \oplus G_2(t)$. Sans perte de généralité, $t' = t$ et on a $G_1(s') \neq G_1(s)$. On en déduit que $G(s', t') = G_1(s') \oplus G_2(t') \neq G_1(s) \oplus G_2(t)$.

On en déduit que $G(s, t) = G_1(s) \oplus G_2(t)$.

On conclut par récurrence.

9. Par récurrence immédiate, la valeur de Grundy d'un jeu de Nim à un seul tas est son nombre de jetons. De plus, un jeu de Nim à plusieurs tas est la somme des jeux de Nim pour chacun des tas, donc la valeur de Grundy est $t_0 \oplus \ldots \oplus t_{n-1}$, ce qui est cohérent avec les questions 5 et 7.

Exercice 5: Clôture par sur-mots et sous-mots

Définition

Soit Σ un alphabet et $u, v \in \Sigma^*$, $u = a_0 \ldots a_{m-1}$ et $v = b_0 \ldots b_{n-1}$. On dit que u est un **sous-mot** de v, noté $u \preccurlyeq v$, s'il existe une fonction strictement croissante $\varphi : [\![0, m-1]\!] \to [\![0, n-1]\!]$ telle que $\forall i \in [\![0, m-1]\!]$, $a_i = b_{\varphi(i)}$. On dit alors que v est un **sur-mot** de u.

Étant donné un langage L, on note $\widehat{L}$ le langage des sur-mots de mots de L, c'est-à-dire

$$\widehat{L} = \{v \in \Sigma^* | \exists u \in L, u \preccurlyeq v\}$$

1. Soient $L_0 = ab^*a$ et $L_1 = (ab)^*$. Déterminer des expressions régulières pour $\widehat{L_0}$ et $\widehat{L_1}$.
2. Montrer que pour tout langage L, on a $\widehat{\widehat{L}} = \widehat{L}$.
3. Existe-t-il des langage L' pour lesquels il n'existe aucun langage L tel que $\widehat{L} = L'$?
4. Montrer que si L est un langage rationnel, alors $\widehat{L}$ est également rationnel.

On admet le :

Théorème

Pour toute suite $(u_n) \in (\Sigma^*)^{\mathbb{N}}$, il existe $i < j$ tels que $u_i \preccurlyeq u_j$.

5. Montrer que pour tout langage L, il existe un langage fini $F \subseteq L$ tel que $\widehat{F} = \widehat{L}$.

Définition

Un langage L est dit **clos par sur-mots** si, pour $u \in L$ et $v \in \Sigma^*$ tels que $u \preccurlyeq v$, on a $v \in L$.

On définit de même un langage **clos par sous-mots**.

6. Montrer que tout langage clos par sur-mots est rationnel.
7. Montrer que tout langage clos par sous-mots est rationnel.
8. Montrer le théorème précédemment admis.

Corrigé

1. Pour L_0, la seule contrainte pour être un sur-mot est que le mot contienne 2 a, soit $\widehat{L_0} = \Sigma^* a \Sigma^* a \Sigma^*$. Pour L_1, tout mot est un sur-mot du mot vide qui est dans L_1, donc $\widehat{L_1} = \Sigma^*$.

2. L'inclusion $\supset$ est directe, car tout mot est un sur-mot de lui-même. L'inclusion $\subseteq$ découle de la transitivité de $\preccurlyeq$: si $u \preccurlyeq v \preccurlyeq w$, alors $u \preccurlyeq w$, donc $w \in \widehat{\widehat{L}} \Rightarrow w \in \widehat{L}$.
3. Le langage L_0 convient comme contre-exemple, car si $L_0 = \widehat{L}$, alors $\widehat{L_0} = \widehat{L} = L_0$, ce qui est faux.
4. On montre cette propriété par induction :
   - $\widehat{\emptyset} = \emptyset$;
   - si $a \in \Sigma$, $\widehat{a} = \Sigma^* a \Sigma^*$;
   - si L_1, L_2 sont des langages rationnels, alors :
     * $\widehat{L_1 \cup L_2} = \widehat{L_1} \cup \widehat{L_2}$ (double inclusion assez simple) ;
     * $\widehat{L_1 L_2} = \widehat{L_1}\widehat{L_2}$ (on découpe le mot en deux) ;
     * $\widehat{L_1^*} = \Sigma^*$ (le mot vide est dans le langage).
5. Si L est fini, alors $F = L$ convient. Sinon, sachant que Σ^* est dénombrable, alors L l'est également. Soit donc $(u_n)_{n\in\mathbb{N}}$ une énumération des mots de L. Posons alors $F = \{u_n \in L | \forall m < n, u_m \not\preccurlyeq u_n\}$. F est bien fini, car dans le cas contraire, il existerait $i < j$, tels que $u_i, u_j \in F$ et $u_i \preccurlyeq u_j$ (par le théorème), ce qui contredirait la construction de F. Par construction également, il semble clair que $F \subseteq L$.

   Montrons alors que $\widehat{L} = \widehat{F}$. L'inclusion $\supset$ est évidente. Réciproquement, soit $v \in \widehat{L}$ tel que $u_n \preccurlyeq v$. Deux cas se présentent : si $u_n \in F$, alors $v \in \widehat{F}$, sinon, $u_n \notin F$, donc il existe $m < n$ tel que $u_m \preccurlyeq u_n$. Quitte à réitérer le processus, on peut supposer $u_m \in F$, et donc par transitivité de $\preccurlyeq$, on a bien $v \in \widehat{F}$.
6. En utilisant les deux questions précédentes, sachant qu'un langage fini est toujours rationnel, on en déduit que $\widehat{L}$ est toujours rationnel (que L soit rationnel ou non). On remarque enfin que si un langage L est clos par sur-mots, alors $\widehat{L} = L$ pour conclure.
7. Soit L un langage clos par sous-mots et $\overline{L}$ son complémentaire. Montrons que $\overline{L}$ est clos par sur-mot. En effet, soit $u \in \overline{L}$ et $v \in \Sigma^*$ tels que $u \preccurlyeq v$. Alors si $v \notin \overline{L}$, on a $v \in L$, et donc u est un sous-mot d'un mot de L, donc est censé être dans L (car L est stable par sous-mots). C'est absurde, donc $v \in \overline{L}$, donc $\overline{L}$ est clos par sur-mots, donc rationnel, donc son complémentaire l'est également.
8. Supposons qu'il existe une suite $(u_n)_{n\in\mathbb{N}}$ qui ne vérifie pas la condition (qu'on appellera une **mauvaise suite** pour le reste de la question). Soit alors v_0 de taille minimale telle qu'il existe une mauvaise suite commençant par v_0. Soit v_1 de taille minimale telle qu'il existe une mauvaise suite commençant par v_0, v_1. On construit de même tous les termes de la (mauvaise) suite $(v_n)_{n\in\mathbb{N}}$. Par construction, pour toute mauvaise suite $(w_n)_{n\in\mathbb{N}}$, si i est le plus petit indice pour lequel $v_i \neq w_i$, alors $|v_i| \leqslant |w_i|$.

   L'alphabet Σ étant fini, soit $a \in \Sigma$ tel qu'il existe une infinité de mots de (v_n) commençant par a et soit $p \in \mathbb{N}$ le plus petit entier tel que v_p commence par a. On pose alors $(w_n)_{n\in\mathbb{N}}$ comme la suite formée de v_0, … v_{p-1} et de la suite extraite de (v_n) des mots commençant par a auxquels on a retiré la première lettre.

   Montrons que la suite (w_n) ainsi formée est une mauvaise suite. En effet, soient $i < j$. Si $j < p$, alors on a bien $w_i = v_i \not\preccurlyeq v_j = w_j$. Si $p \leqslant i$, alors $aw_i = v_i \not\preccurlyeq v_j = aw_j$, donc $w_i \not\preccurlyeq w_j$. Si $i < p \leqslant j$, alors $w_i = v_i \not\preccurlyeq v_j = aw_j$, donc $w_i \not\preccurlyeq w_j$, car w_i ne commence pas par a.

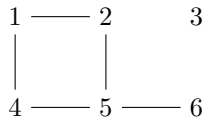
   De plus, cette suite contredit la minimalité de (v_n), car $|v_p| > |w_p|$ et p est bien le premier indice où les deux suites diffèrent. On conclut par l'absurde.

Exercice 6: Acyclicité dans les hypergraphes

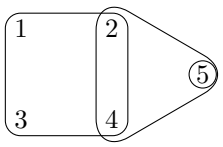
Définition

Un **graphe non orienté** est un couple $G = (S, A)$ où $A \subseteq \mathcal{P}_2(S)$ est l'ensemble des arêtes. On définit un **hypergraphe** comme un couple $H = (S, M)$ où $M \subseteq \mathcal{P}(S) \setminus \{\emptyset\}$ est un ensemble d'**hyperarêtes**, c'est-à-dire de sous-ensembles de S de cardinal au moins 1. Ainsi, un graphe peut être vu comme un hypergraphe dont toutes les hyperarêtes sont de cardinal 2. Pour $X \subseteq S$, on définit également le **projeté** sur X par $H[X] = (X, \{m \cap X | m \in M\} \setminus \{\emptyset\})$.

1. On donne le graphe G_0 et l'hypergraphe H_0 ci-dessous. Dessiner $G_0[\{2,3,4,5\}]$ et $H_0[\{1,2,5\}]$.



G_0



H_0

Définition

On définit le **réduit** $R(H)$ d'un hypergraphe H comme étant l'hypergraphe H dans lequel on a supprimé les hyperarêtes incluses dans une autre hyperarête. Par exemple, dans le réduit de H_0, on a supprimé le singleton $\{5\}$ comme hyperarête.

On dit qu'un hypergraphe $H = (S, M)$ a un **cycle induit** s'il existe $X = \{s_1, \ldots, s_n\} \subseteq S$ tel que $n \geqslant 3$ et si $R(H[X]) = (X, M_X)$, alors $M_X = \{\{s_i, s_{i+1}\} | i \in [\![1; n-1]\!]\} \cup \{s_n, s_1\}$.

2. Déterminer si G_0 et H_0 possèdent des cycles induits.
3. Montrer qu'un graphe G possède un cycle induit si et seulement s'il possède un cycle au sens usuel.
4. Exhiber un hypergraphe $H = (S, M \cup \{m\})$ qui ne possède pas de cycle induit, mais tel que (V, M) possède un cycle induit. Commenter.

Définition

Pour un hypergraphe $H = (S, M)$, on définit le **graphe d'incidence** de H par $I(H) = (S \cup M, A_I)$ où $A_I = \bigcup_{m \in M} \bigcup_{s \in m} \{\{s, m\}\}$. On dit qu'un hypergraphe est **Berge-cyclique** si son graphe d'incidence $I(H)$ contient un cycle (au sens usuel).

5. Montrer qu'un graphe G contient un cycle (au sens usuel) si et seulement s'il est Berge-cyclique.

   Est-il possible qu'un hypergraphe possède un cycle induit sans être Berge-cyclique ? Est-il possible qu'un hypergraphe soit Berge-cyclique sans posséder de cycle induit ?

Définition

Pour $H = (S, M)$, une **feuille** $s \in S$ est un sommet tel qu'il existe une hyperarêtes m_s qui contient toutes les hyperarêtes contenant s. Pour une feuille s, on note $H[-s] = H[S \setminus \{s\}]$.

6. Soit H un hypergraphe et s une feuille de H. Montrer que H possède un cycle induit si et seulement si $H[-s]$ possède un cycle induit.
7. Montrer qu'il existe un hypergraphe H Berge-cyclique qui possède une feuille s tel que $H[-S]$ ne soit pas Berge-cyclique.
8. Montrer que si un hypergraphe est non vide et non Berge-cyclique, alors il possède une feuille.

Corrigé

1. Question de prise en main. Il ne faut pas hésiter et dessiner ce qui est intuitif.
2. G_0 possède un cycle induit, en prenant $(1, 2, 5, 4)$ comme séquence et $X = \{1, 2, 4, 5\}$. H_0 ne possède pas de cycle induit, car on ne peut pas choisir dans la séquence trois sommets de la même multi-arête. Choisir $\geqslant 3$ sommets pour former un cycle induit revient donc (sans perte de généralité) à choisir $X = (1, 2, 5)$ qui ne forme pas de cycle dans son réduit projeté.
3. Le sens direct est évident, car si un réduit projeté possède des multi-arêtes, le graphe possède les mêmes arêtes. Pour le sens réciproque, il faut par contre considérer le plus petit cycle usuel dans le graphe. Cela nous assure qu'il n'existe pas d'arête qui « traverse » le cycle.
4. L'hypergraphe $H = \{\{1, 2, 3\}, \{\{1, 2\}, \{2, 3\}, \{1, 3\}, \{1, 2, 3\}\}$ convient en supprimant la multi-arête $\{1, 2, 3\}$. Cela change par rapport aux graphes usuels, dans lequel supprimer des arêtes ne peut pas créer des cycles.
5. Pour la première question, il suffit (pour les deux sens) de voir que la construction du graphe d'incidence consiste à rajouter un sommet intermédiaire sur chaque arête. La cyclicité est donc conservée.

   Un hypergraphe H qui possède un cycle induit est toujours Berge-cyclique, car si dans un réduit projeté il existe une multi-arête $\{v_1, v_2\}$, alors cela signifie que dans H, v_1 et v_2 sont dans une même multi-arête m. Il existe donc un chemin $v_1 - m - v_2$ dans le graphe d'incidence. On construit de cette façon un cycle dans le graphe d'incidence.

   Pour la dernière question, le contre-exemple de la question précédente convient (car il est Berge-cyclique).
6. Il s'agit ici de montrer qu'une feuille ne peut pas faire partie d'un cycle induit. En effet, si dans un cycle induit on voit les multi-arêtes $v_1 - v - v_2$, v étant une feuille, il existe nécessairement une multi-arête contenant v, v_1 et v_2, donc on ne peut pas choisir v, v_1 et v_2 pour construire le cycle induit.
7. Le contre-exemple de la question 4 convient, car en enlevant un sommet, on ne peut plus construire de cycle.
8. Il s'agit ici de raisonner par contraposée : si un hypergraphe ne possède pas de feuille, alors on choisit un sommet quelconque v_0 et on peut le relier, dans le graphe d'incidence, à un sommet v_1 de la même multi-arête. Dès lors, il existe un autre sommet $v_2 \notin \{v_0, v_1\}$ tel que v_1 et v_2 sont dans la même multi-arête (sinon, v_1 serait une feuille). On procède ainsi jusqu'à retomber sur un sommet déjà visité pour créer un cycle.