ÉCOLE NORMALE SUPÉRIEURE DE LYON

Concours d'admission session 2026

Filière universitaire : Second concours

COMPOSITION D'INFORMATIQUE

Durée : 3 heures

L'utilisation des calculatrices n'est pas autorisée pour cette épreuve. L'orthographe ainsi que la propreté de la copie seront prises en compte dans la correction.

⋆ ⋆ ⋆

Fonctions booléennes évasives et complexes simpliciaux

Ce sujet comporte 4 parties, qui **ne sont pas** indépendantes. Il est toutefois possible d'admettre le résultat de toute question pour répondre aux questions suivantes.

La **cardinalité** d'un ensemble fini A est notée $\#A$.

Le **temps d'exécution** d'un algorithme sur une entrée donnée est le nombre d'opérations élémentaires (addition, multiplication, affectation, test, etc.) nécessaires à l'exécution de l'algorithme. La **complexité** d'un algorithme exprime, en fonction de certains paramètres, le temps d'exécution dans le pire cas. Lorsque ces paramètres sont $n_1, \ldots, n_k$, on dira qu'un algorithme est de complexité $O(f(n_1, \ldots, n_k))$ s'il existe des constantes C et $N_1, \ldots, N_k$ telles que le temps d'exécution soit au plus $C \times f(n_1, \ldots, n_k)$ pour tout $n_1 \geq N_1, \ldots, n_k \geq N_k$.

Le langage de référence est Python. Les algorithmes devront être écrits en pseudo-code proche de Python, en utilisant les structures de contrôle habituelles (p. ex. la boucle `for i in range(n)` parcourt les entiers `i` de 0 à $\mathtt{n}-1$ avec complexité $O(\mathtt{n})$) et les structures de données suivantes.

Listes. Les listes sont délimitées par des crochets : $\mathtt{[v_0, \ldots, v_{n-1}]}$ désigne une liste dont les éléments sont les $\mathtt{v}_i$ pour $0 \leq i < n$. La liste vide est `[]`. Si `x` est une liste de longueur n :
- L'expression `len(x)` renvoie la longueur n de `x`.
- L'expression `x[i]` renvoie le `i`-ème élément de `x`, pour $0 \leq \mathtt{i} < n$.
- L'instruction `x[i] = v` affecte la valeur `v` au `i`-ème élément de `x`, pour $0 \leq \mathtt{i} < n$.
- L'instruction `x.append(v)` attache l'élément `v` à la fin de `x`.

Ces opérations ont toutes complexité $O(1)$. De plus, l'expression `x.copy()` renvoie une copie de `x` avec complexité $O(n)$.

Files `deque`. Les files `deque` sont munies des opérations ci-dessus sur les listes, avec les mêmes complexités. On a de plus les deux opérations suivantes, de complexité $O(1)$:
- L'expression `deque()` renvoie la file vide.
- L'instruction `x.popleft()` supprime et renvoie le premier élément de la file `x`.

Tuples. Les tuples sont des listes **immuables** délimitées par des parenthèses : $\mathtt{(v_0, \ldots, v_{n-1})}$ désigne un tuple dont les éléments sont les $\mathtt{v}_i$ pour $0 \leq i < n$. Le tuple vide est `()`. Les expressions `len(x)` et `x[i]` sont disponibles sur un tuple `x`, avec complexité $O(1)$. Mais les opérations `x[i] = v`, `x.append(v)` et `x.copy()` **ne sont pas** disponibles. De plus :
- L'expression `tuple(x)` renvoie un tuple ayant les mêmes éléments que la liste `x`.
- L'expression `x+y` renvoie la concaténation des tuples `x` et `y`.

Ces deux opérations ont complexité $O(m)$, où m est la longueur du résultat.

Dictionnaires. Un dictionnaire `d` de la forme $\mathtt{\{k_0: v_0, \ldots, k_{n-1}: v_{n-1}\}}$ associe la valeur $\mathtt{v}_i$ à la clé $\mathtt{k}_i$ (le dictionnaire vide est `{}`). Les clés doivent être des objets immuables (p. ex. entiers, tuples d'objets immuables), et non des objets mutables (p. ex. listes, files `deque`). La boucle `for k in d` parcourt les clés de `d`, avec complexité $O(n)$. De plus :
- Le test `(k in d)` renvoie `True` si `k` est une clé de `d`, et renvoie `False` sinon.
- L'instruction `d[k] = v` associe la valeur `v` à `k` dans `d`, et fait de `k` une clé de `d` si elle ne l'était pas.
- L'expression `d[k]` renvoie la dernière valeur associée à `k` si `k` est une clé de `d`, et provoque une erreur sinon.

Ces trois opérations ont complexité $O(1)$.

1 Fonctions booléennes

Dans toute la suite, n désigne un entier > 0.

On note $\mathbb{B} = \{\texttt{True}, \texttt{False}\}$ l'ensemble des **booléens** et $\mathbb{B}^n$ l'ensemble des listes de booléens de longueur n. Une **fonction booléenne sur** n est une fonction qui prend en entrée une liste $\texttt{x} \in \mathbb{B}^n$ et qui renvoie un booléen. On écrit $\texttt{f}\colon \mathbb{B}^n \to \mathbb{B}$ pour indiquer que `f` est une fonction booléenne sur n.

Exemple 1.1. *Les algorithmes suivants définissent des fonctions booléennes* $\texttt{ou}, \texttt{et}\colon \mathbb{B}^n \to \mathbb{B}$ *pour tout* $n > 0$ *:*

```
def ou(x):
    for b in x:
        if b:
            return True
    return False
```

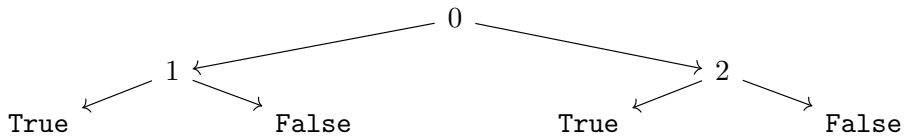
```
def et(x):
    for b in x:
        if not b:
            return False
    return True
```

Question 1. *Écrire un algorithme* `consecutifs(x)`*, où* $\texttt{x} \in \mathbb{B}^5$*, qui renvoie* `True` *s'il existe un* $\texttt{v} \in \{0, 1, 2\}$ *tel que* `x[v]` $=$ `x[v+1]` $=$ `x[v+2]` $=$ `True`*, et qui renvoie* `False` *sinon.*

1.1 Arbres de décision

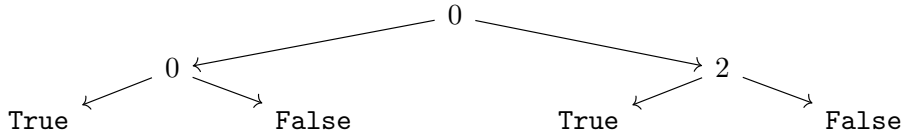
Soit $n > 0$. Un **n-arbre** est un arbre binaire dont les nœuds sont de deux sortes :
- des **feuilles**, étiquetées par des booléeens (et n'ayant pas d'enfants) ;
- des **nœuds internes**, étiquetés par des entiers $v \in \{0, \dots, n-1\}$, et ayant exactement deux enfants : un **enfant gauche** et un **enfant droit**.

Exemple 1.2. *Voici un exemple de* 3*-arbre :*



Un **arbre de décision sur** n est un n-arbre `t` tel que pour chaque chemin c de la racine de `t` à une feuille, les étiquettes de nœuds internes le long de c sont deux-à-deux distinctes.

Exemple 1.3. *L'arbre de l'Exemple 1.2 est un arbre de décision sur* 3*. Mais l'arbre ci-dessous n'est pas un arbre de décision :*



Chaque arbre de décision `t` sur n induit une fonction booléenne $f_{\texttt{t}}\colon \mathbb{B}^n \to \mathbb{B}$. La définition est par induction sur la structure de `t` :
- Si `t` est une feuille d'étiquette `b`, alors $f_{\texttt{t}}$ est la fonction constante de valeur $\texttt{b} \in \mathbb{B}$.
- Sinon, `t` est un nœud interne. Notons `v` l'étiquette de `t`, et notons respectivement `left` et `right` l'enfant gauche et l'enfant droit de `t`. Alors $f_{\texttt{t}}$ est la fonction

$$\texttt{x} \in \mathbb{B}^n \quad \longmapsto \quad f_{\texttt{t}}(\texttt{x}) = \begin{cases} f_{\texttt{left}}(\texttt{x}) & \text{si } \texttt{x[v]} = \texttt{True} \\ f_{\texttt{right}}(\texttt{x}) & \text{si } \texttt{x[v]} = \texttt{False} \end{cases}$$

où les fonctions booléennes $f_{\texttt{left}}, f_{\texttt{right}} : \mathbb{B}^n \to \mathbb{B}$ sont obtenues récursivement à partir des arbres de décision `left` et `right`, respectivement.

Exemple 1.4. *La fonction booléenne* `cond`$: \mathbb{B}^3 \to \mathbb{B}$ *induite par l'arbre de l'Exemple 1.2 est*

$$\texttt{x} \in \mathbb{B}^3 \quad \longmapsto \quad \texttt{cond(x)} = \begin{cases} \texttt{x[1]} & \textit{si } \texttt{x[0]} = \texttt{True} \\ \texttt{x[2]} & \textit{si } \texttt{x[0]} = \texttt{False} \end{cases}$$

Une fonction booléenne $\texttt{f} : \mathbb{B}^n \to \mathbb{B}$ est **représentée** par un arbre de décision `t` sur n si on a $\texttt{f(x)} = f_{\texttt{t}}(\texttt{x})$ pour tout $\texttt{x} \in \mathbb{B}^n$. On dit alors que `t` est un **arbre de décision pour** $\texttt{f} : \mathbb{B}^n \to \mathbb{B}$**.**

Question 2. *Donner un arbre de décision pour la fonction* `ou`$: \mathbb{B}^3 \to \mathbb{B}$*, ainsi qu'un arbre de décision pour la fonction* `et`$: \mathbb{B}^3 \to \mathbb{B}$ *(voir l'Exemple 1.1).*

1.2 Implémentation des arbres de décision

On suppose fournie une structure de données pour les n-arbres. Les seules opérations disponibles sur cette structure de données sont les fonctions suivantes, toutes de complexité $O(1)$:

- Une fonction `Leaf(b)` qui renvoie un arbre dont la racine est une feuille étiquetée par `b`.
- Une fonction `Node(v, left, right)` qui renvoie un arbre dont la racine est un nœud interne étiqueté par l'entier `v`, et qui a pour enfant gauche l'arbre `left` et pour enfant droit l'arbre `right`.
- Une fonction `isLeaf(t)` qui renvoie `True` si l'arbre `t` est feuille, et renvoie `False` sinon.
- Une fonction `getValue(t)` qui renvoie l'étiquette de la feuille `t`. Cette fonction provoque une erreur si `t` n'est pas une feuille.
- Des fonctions `getLabel(t)`, `getLeft(t)` et `getRight(t)`, qui renvoient respectivement l'étiquette, l'enfant gauche et l'enfant droit du nœud interne `t`. Ces fonctions provoquent une erreur si `t` n'est pas un nœud interne.

Exemple 1.5. *L'arbre de l'Exemple 1.2 est construit par*

```
Node(0, Node(1, Leaf(True), Leaf(False)), Node(2, Leaf(True), Leaf(False)))
```

Question 3. *Écrire un algorithme* `evaltree(t, n, x)`*, où* `t` *est un arbre de décision sur* $\texttt{n} > 0$ *et* $\texttt{x} \in \mathbb{B}^{\texttt{n}}$*, et qui renvoie* $f_{\texttt{t}}(\texttt{x})$*. La complexité doit être* $O(\texttt{n})$*.*

1.3 Fonctions booléennes évasives

La **hauteur** d'un arbre de décision est le nombre maximal de nœuds internes sur un chemin de la racine à une feuille. Un arbre de décision sur n est donc de hauteur $\leq n$.

Soit $n > 0$. Une fonction booléenne $\texttt{f} : \mathbb{B}^n \to \mathbb{B}$ est **évasive** si tous ses arbres de décision sont de hauteur n.

Exemple 1.6. *La fonction booléenne* `cond`$: \mathbb{B}^3 \to \mathbb{B}$ *de l'Exemple 1.4 n'est pas évasive. Les fonctions* `ou`, `et`$: \mathbb{B}^n \to \mathbb{B}$ *de l'Exemple 1.1 sont évasives pour tout* $n > 0$*.*

Question 4. *Donner un arbre de décision de hauteur* < 5 *pour la fonction* `consecutifs`$: \mathbb{B}^5 \to \mathbb{B}$ *de la Question 1.*

La suite de ce sujet concerne des algorithmes pour assurer qu'une fonction booléenne donnée est évasive.

2 Une approche énumérative

Dans cette Partie 2, on va tester si une fonction booléenne donnée est évasive en énumérant des arbres de décision.

2.1 Arbres de décision complets

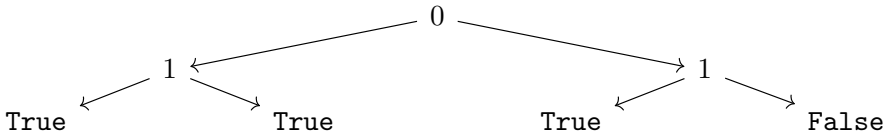
Un arbre de décision sur n est **complet** si tous les chemins de la racine à une feuille ont exactement n nœuds internes.

Question 5. *Donner un arbre de décision complet pour la fonction booléenne* `cond`$\colon \mathbb{B}^3 \to \mathbb{B}$ *de l'Exemple 1.4.*

2.2 Arbres de décision concis

Un arbre de décision est **redondant** s'il a un nœud interne dont les deux enfants sont des feuilles de même étiquette. Un arbre de décision est **concis** s'il n'est pas redondant.

Exemple 2.1. *L'arbre de décision de l'Exemple 1.2 est concis. L'arbre suivant est redondant :*



Question 6. *Écrire un algorithme* `condense(t, n)`*, où* `t` *est un arbre de décision sur* `n` > 0*, et qui renvoie un arbre de décision concis représentant la même fonction booléenne sur* `n` *que* `t`*. La complexité doit être* $O(\#\texttt{t})$*, où* $\#$`t` *est le nombre total de nœuds (nœuds internes et feuilles) de l'arbre* `t`*.*

2.3 Une énumération d'arbres de décision

Une manière de vérifier si une fonction booléenne sur n est évasive est d'énumérer tous ses arbres de décision complets, pour ensuite tester si l'algorithme `condense` de la Question 6 permet d'obtenir un arbre de hauteur $< n$.

Une difficulté est que les étiquettes des nœuds internes n'apparaissent pas forcément dans le même ordre sur tous les chemins (voir l'Exemple 1.2).

On suggère de s'aider de la construction suivante. Fixons $n > 0$. Étant donné un ensemble $V \subseteq \{0, \dots, n-1\}$, on définit l'ensemble d'arbres Arbres(V) par récurrence sur $\#V$:

- Si $V = \emptyset$, alors Arbres(V) = {`Leaf(True)`, `Leaf(False)`}.
- Sinon, Arbres(V) est l'ensemble des `Node(v, left, right)` pour `v` $\in V$ et `left`, `right` $\in$ Arbres($V \setminus$ {`v`}).

Alors Arbres($\{0, \dots, n-1\}$) est l'ensemble des arbres de décision complets sur n.

Question 7. *Écrire un algorithme* `enumtrees(f, n)`*, où* `f` *est une fonction booléenne sur* `n` > 0*, et qui renvoie une liste contenant tous les arbres de décision complets pour* `f`*.*

3 Fonctions booléennes positives et complexes simpliciaux

Soit $n > 0$. Une fonction booléenne `f`$\colon \mathbb{B}^n \to \mathbb{B}$ est **positive** si elle est croissante pour l'ordre `False` $\leq$ `True`, c'est-à-dire si pour tous `x`, `y` $\in \mathbb{B}^n$, on a `f(x)` $\leq$ `f(y)` dès lors que `x[v]` $\leq$ `y[v]` pour tout `v` $\in \{0, \dots, n-1\}$.

Exemple 3.1. *Les fonctions* `ou`, `et` *et* `consecutifs` *(Exemple 1.1 et Question 1) sont positives. La fonction* `cond` *de l'Exemple 1.4 n'est pas positive.*

Dans cette Partie 3, on va énoncer une condition suffisante pour qu'une fonction booléenne positive soit évasive. Cette condition utilise la notion de complexe simplicial.

3.1 Complexes simpliciaux

Soit $n > 0$. Un **complexe simplicial** sur n est un ensemble K de sous-ensembles de $\{0, \dots, n-1\}$ tel que si $\sigma \in K$ et $\tau \subseteq \sigma$, alors $\tau \in K$. Les **faces** de K sont les ensembles $\sigma \subseteq \{0, \dots, n-1\}$ tels que $\sigma \in K$.

Attention : Un complexe simplicial peut être vide ($K = \emptyset$). Le complexe vide $K = \emptyset$ est différent du complexe $K = \{\emptyset\}$. Notons que $\emptyset \in K$ si $K \neq \emptyset$.

Une fonction booléenne positive $\texttt{f}\colon \mathbb{B}^n \to \mathbb{B}$ induit un complexe simplicial $K^n_{\texttt{f}}$ sur n :

$$K^n_{\texttt{f}} = \{\gamma_{\texttt{x}} \mid \texttt{x} \in \mathbb{B}^n \text{ et } \texttt{f(x)} = \texttt{True}\}$$

où $\gamma_{\texttt{x}}$ est l'ensemble des $\texttt{v} \in \{0, \dots, n-1\}$ tels que $\texttt{x[v]} = \texttt{False}$.

Exemple 3.2. *Pour $n > 0$, on a $K^n_{\texttt{et}} = \{\emptyset\}$ et $K^n_{\texttt{false}} = \emptyset$, où* $\texttt{false}\colon \mathbb{B}^n \to \mathbb{B}$ *est la fonction constante de valeur* `False`*. D'autre part,*

$$K^5_{\texttt{consecutifs}} = \{\emptyset, \{0\}, \{1\}, \{3\}, \{4\}, \{0,1\}, \{3,4\}, \{0,4\}\}$$

Question 8. *Donner le complexe simplicial $K^3_{\texttt{ou}}$.*

3.2 Implémentation des complexes simpliciaux

Soit K un complexe simplicial. Une face $\{\texttt{v}_1, \dots, \texttt{v}_\ell\}$ de K avec $\texttt{v}_1 < \cdots < \texttt{v}_\ell$ peut être vue comme une suite $\texttt{v}_1, \dots, \texttt{v}_\ell$. Lorsque $K \neq \emptyset$, l'ensemble de ces suites peut être vu comme un arbre pour l'ordre préfixe : la racine est la suite vide, et chaque suite $\texttt{v}_1, \dots, \texttt{v}_\ell$ a pour enfants les suites $\texttt{v}_1, \dots, \texttt{v}_\ell, \texttt{v}$ telles que $\{\texttt{v}_1, \dots, \texttt{v}_\ell, \texttt{v}\} \in K$ et $\texttt{v}_1 < \cdots < \texttt{v}_\ell < \texttt{v}$. Ces arbres vont être représentés par des dictionnaires imbriqués.

La **représentation** de K est définie par récurrence sur $\#K$:

— La représentation du complexe simplicial vide $K = \emptyset$ est `None`.

— Si K n'est pas vide, sa représentation est le dictionnaire dont les clés sont exactement les `v` tels que $\{\texttt{v}\} \in K$, et qui associe à la clé `v` la représentation du complexe simplicial $\{\gamma \setminus \{\texttt{v}\} \mid \gamma \in K \text{ et } \texttt{v} = \min(\gamma)\}$.

Exemple 3.3. *Le complexe $\{\emptyset\}$ est représenté par* `{}`*. Le complexe $K^5_{\texttt{consecutifs}}$ de l'Exemple 3.2 est représenté par*

```
{0: {1: {}, 4: {}}, 1: {}, 3: {4: {}}, 4: {}}
```

Question 9. *Écrire un algorithme* `makecmpl(f, n)`*, où* `f` *est une fonction booléenne sur* $\texttt{n} > 0$*, et qui renvoie la représentation du complexe $K^{\texttt{n}}_{\texttt{f}}$. La complexité doit être $O(2^{\texttt{n}})$, en supposant que la complexité de* `f` *est $O(1)$.*

Il est utile pour la suite de disposer d'un algorithme qui prend en entrée une représentation d'un complexe simplicial et qui renvoie la liste de ses faces. Nous adopterons la **convention** suivante : les faces sont représentées par des tuples d'entiers $(\texttt{v}_1, \dots, \texttt{v}_\ell)$ avec $\texttt{v}_1 < \cdots < \texttt{v}_\ell$.

Question 10. *Écrire un algorithme* `makefaces(cmpl)`*, où* `cmpl` *représente un complexe simplicial K, et qui renvoie la liste des faces de K, ordonnée par faces de taille croissante. La complexité doit être $O(n \times \#K)$ si K est un complexe sur n. On pourra s'inspirer d'un parcours « en largeur d'abord », et utiliser les files du module* `deque`*.*

3.3 Caractéristique d'Euler réduite

Soit K un complexe simplicial sur $n > 0$. La **caractéristique d'Euler réduite** de K est

$$\widetilde{\chi}(K) = \sum_{i=0}^{n} (-1)^{i+1} C_i(K)$$

où $C_i(K) = \#\{\sigma \in K \mid \#\sigma = i\}$.

Exemple 3.4. *Pour tout $n > 0$, on a $\widetilde{\chi}(K^n_{\texttt{et}}) = -1$ et $\widetilde{\chi}(K^n_{\texttt{false}}) = 0$ (voir l'Exemple 3.2).*

Question 11. *Donner la valeur de $\widetilde{\chi}(K^5_{\texttt{consecutifs}})$ (voir l'Exemple 3.2).*

Le résultat suivant fournit une condition suffisante pour qu'une fonction booléenne positive soit évasive.

Théorème (Kahn-Saks-Sturtevant)**.** *Soit $\texttt{f} : \mathbb{B}^n \to \mathbb{B}$ une fonction booléenne positive. Si $\texttt{f}$ n'est pas évasive, alors $\widetilde{\chi}(K^n_{\texttt{f}}) = 0$.*

4 Couplages de Morse

L'objet principal de cette Partie 4 est de démontrer le Théorème de Kahn-Saks-Sturtevant. Soit K un complexe simplicial. Un **couplage** sur K est un ensemble $M \subseteq K \times K$ tel que

— pour tout couple $(\tau, \sigma) \in M$, on a $\tau \subseteq \sigma$ et $\#\tau + 1 = \#\sigma$, et

— pour chaque face $\gamma \in K$, il existe au plus un couple $(\tau, \sigma) \in M$ tel que $\gamma \in \{\tau, \sigma\}$.

Un couplage M sur K est **parfait** si pour toute face $\gamma \in K$, il existe un couple $(\tau, \sigma) \in M$ tel que $\gamma \in \{\tau, \sigma\}$.

Le **diagramme de Hasse** d'un couplage M sur K est le graphe orienté $G(K, M)$ dont les sommets sont les faces de K, et dont les arcs sont définis comme suit. Soient $\tau, \sigma \in K$ telles que $\tau \subseteq \sigma$ et $\#\tau + 1 = \#\sigma$.

— Si $(\tau, \sigma) \in M$, alors $G(K, M)$ a un arc $\tau \to \sigma$.

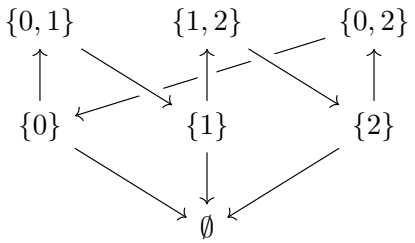
— Sinon, $G(K, M)$ a un arc $\sigma \to \tau$.

Ainsi, l'ensemble d'arcs de $G(K, M)$ est

$$M \cup \{(\sigma, \tau) \in K \times K \mid \tau \subseteq \sigma \text{ et } \#\tau + 1 = \#\sigma \text{ et } (\tau, \sigma) \notin M\}$$

Un arc $\alpha \to \beta$ de $G(K, M)$ est dit **montant** si $(\alpha, \beta) \in M$. Sinon, on dit que $\alpha \to \beta$ est un arc **descendant**.

Exemple 4.1. *Considérons le complexe $K = \{\emptyset, \{0\}, \{1\}, \{2\}, \{0,1\}, \{1,2\}, \{0,2\}\}$ et le couplage $M = \{(\{0\}, \{0,1\}), (\{1\}, \{1,2\}), (\{2\}, \{0,2\})\}$. Alors $G(K, M)$ est le graphe*



Un **couplage de Morse** sur K est un couplage M sur K tel que le diagramme de Hasse $G(K, M)$ est un graphe acyclique.

Exemple 4.2. *Le couplage de l'Exemple 4.1 n'est pas un couplage de Morse.*

Question 12. *Dessiner le graphe $G(K^5_{\texttt{consecutifs}}, M)$ pour*

$$M = \{(\{1\}, \{0,1\}), (\{0\}, \{0,4\}), (\{4\}, \{3,4\}), (\emptyset, \{3\})\}$$

4.1 Implémentation des diagrammes de Hasse

On représente un diagramme de Hasse $G(K, M)$ par un **dictionnaire d'adjacence**, c'est-à-dire un dictionnaire dont les clés sont les faces de K (en suivant la convention du §3.2), et qui associe à chaque face α de K la liste des faces $\beta \in K$ telles que (α, β) est un arc de $G(K, M)$. Les éléments de chacune de ces listes doivent être deux-à-deux distincts mais peuvent être dans un ordre quelconque.

Un couplage $M = \{(\tau_1, \sigma_1), \ldots, (\tau_\ell, \sigma_\ell)\}$ est représenté par une liste

$$\texttt{[(tau}_1\texttt{, sigma}_1\texttt{), ..., (tau}_\ell\texttt{, sigma}_\ell\texttt{)]}$$

où $\texttt{tau}_i$ et $\texttt{sigma}_i$ sont les tuples représentant τ_i et σ_i, respectivement.

Question 13. *Écrire un algorithme* `makehasse(cmpl, m)`*, où* `cmpl` *représente un complexe simplicial K et* `m` *représente un couplage M sur K, et qui renvoie un dictionnaire d'adjacence pour $G(K, M)$. La complexité doit être $O(n \times \#K)$ si K est un complexe sur n.*

Question 14. *Écrire un algorithme* `isacyclic(adj)`*, où* `adj` *est un dictionnaire d'adjacence d'un graphe orienté G, qui renvoie* `True` *si le graphe G est acyclique, et qui renvoie* `False` *sinon. La complexité doit être $O(\#V + \#E)$, où V et E sont respectivement les ensembles de sommets et d'arcs de G.*

4.2 Effondrements

Soit K un complexe simplicial et soient deux faces $\tau, \sigma \in K$ telles que $\tau \subseteq \sigma$ et $\#\tau + 1 = \#\sigma$. Supposons que σ est maximale dans K pour l'inclusion, et que τ n'est incluse dans aucune autre face de K. On dit alors que le complexe simplicial $K \setminus \{\tau, \sigma\}$ est un **effondrement élémentaire** de K et on note $K \searrow (K \setminus \{\tau, \sigma\})$.

On dit que K est **effondrable** s'il existe une séquence d'effondrements élémentaires

$$K = K_0 \searrow K_1 \searrow \cdots \searrow K_m = \emptyset$$

avec $m \geq 0$ (de sorte que le complexe vide $\emptyset$ est effondrable).

Question 15. *Soit K un complexe simplicial. Montrer que s'il existe un couplage de Morse parfait sur K, alors K est effondrable.*

4.3 Démonstration du Théorème de Kahn-Saks-Sturtevant

Soit `t` un arbre de décision complet sur $n > 0$. Un chemin c de la racine de `t` à une feuille détermine exactement une suite

$$v_0, d_0, \ldots, v_{n-2}, d_{n-2}, v_{n-1}, d_{n-1}$$

où $v_0, \ldots, v_{n-1}$ est la suite des étiquettes de nœuds internes le long de c (v_0 est l'étiquette de la racine) et où $d_0, \ldots, d_{n-1}$ sont des booléens tels que :

— $d_i = \texttt{True}$ si le chemin c suit l'enfant gauche de v_i,

— $d_i = \texttt{False}$ si le chemin c suit l'enfant droit de v_i.

Étant donné un tel chemin c, on définit $\tau_c, \sigma_c \subseteq \{0, \ldots, n-1\}$ par :

$$\begin{aligned} \tau_c &= \{v_i \mid 0 \leq i \leq n-2 \text{ et } d_i = \texttt{False}\} \\ \sigma_c &= \tau_c \cup \{v_{n-1}\} \end{aligned}$$

Notons que τ_c et σ_c ne dépendent pas de d_{n-1}. D'autre part, on a toujours $\tau_c \subseteq \sigma_c$ et $\#\sigma_c = \#\tau_c + 1$. Soit maintenant

$$M_{\mathtt{t}} = \{(\tau_c, \sigma_c) \mid c \text{ chemin de la racine de } \mathtt{t} \text{ à une feuille}\}$$

On note Δ^n le complexe simplicial sur n dont les faces sont **tous** les sous-ensembles de $\{0, \dots, n-1\}$. Il est clair (et admis) que $M_{\mathtt{t}}$ est un couplage sur Δ^n. De plus, comme $\mathtt{t}$ est un arbre de décision complet, pour tout $\gamma \in \Delta^n$ il existe dans $\mathtt{t}$ un chemin c tel que $\gamma \in \{\tau_c, \sigma_c\}$.

Question 16. *Montrer que $M_{\mathtt{t}}$ est un couplage de Morse parfait sur Δ^n.*

Question 17. *Soit $\mathtt{f} : \mathbb{B}^n \to \mathbb{B}$ une fonction booléenne positive, avec $n > 0$. Montrer que si $\mathtt{f}$ n'est pas évasive, alors $\widetilde{\chi}(K^n_{\mathtt{f}}) = 0$.*

Remarque. Une fonction booléenne $f : \mathbb{B}^n \to \mathbb{B}$ est **faiblement symétrique** si pour tous $i, j \in \{0, \dots, n-1\}$, il existe une permutation π de $\{0, \dots, n-1\}$ telle que $\pi(i) = j$ et $f(x) = f(\pi(x))$ pour tout $x \in \mathbb{B}^n$. La question suivante est ouverte :

— Est-ce que toute fonction positive, faiblement symétrique et non-triviale est évasive ?

Une réponse positive à cette question apporterait une réponse positive à la célèbre conjecture d'Aanderaa-Karp-Rosenberg sur les propriétés monotones de graphes.

* *
*