# IFT800 - Série d'exercices #4

## Algorithmes de branchement

*Exercice* 1. Dans le problème du CLUSTER-VERTEX-EDITING, on veut savoir s'il est possible de retirer au plus $k$ **sommets** d'un graphe pour que chaque composante connexe soit une clique. Donnez un algorithme $O(3^k \cdot n^3)$ pour ce problème.

*Défi.* Essayez de faire mieux que ça. C'est-à-dire, un algo $O(a^k n^c)$ où $a < 3$. Ça se fait, mais les solutions que je connais semblent assez difficiles.

---

**Solution.**

On se rappelle que pour que chaque composante connexe soit une clique, il faut éliminer chaque $P_3$. Notre stratégie de branchement sera la suivante: trouver un $P_3$ formé par des sommets $a, b, c$. S'il n'y a pas de $P_3$, on a terminé. S'il y a un tel $P_3$, brancher sur trois possibilités : retirer $a$, retirer $b$, retirer $c$. Chaque cas réduit $k$ de 1. Ceci créé un arbre de récursion où chaque noeud a 3 enfants et dont la profondeur est $k$. On a donc un algorithme $O(3^k n^3)$, le $n^3$ venant du temps requis pour trouver un $P_3$.

Pour ce qui est d'obtenir un meilleur algorithme, je l'ai fait en classe.

---

*Exercice* 2. Considérez la version de VERTEX-COVER avec poids sur les sommets, où on paramétrise par le poids total de la solution.

VERTEX-COVER avec poids
**Entrée :** graphe $G = (V, E)$ et fonction $f : V \to \mathbb{N}$ sur les sommets
**Paramètre :** $k$, le poids total d'une solution
**Sortie :** un ensemble $X \subseteq V$ qui couvre chaque arête de $G$ tel que $\sum_{x \in X} f(x) \leq k$, ou *null* si non-existant

Soit $w = \min_{x \in V} f(x)$ le poids minimum. Notez que tous les poids sont des entiers non-négatifs. En supposant que $w \geq 1$, donnez un algorithme en temps $O(2^{k/w} \cdot n^c)$ pour ce problème.

---

**Solution.**

On utilise le même algorithme de base que celui vu en classe. C'est-à-dire, on choisit une arête $uv \in E$. On branche sur retirer $u$ en réduisant $k$ de $f(u)$, et on branche sur retirer $v$ en réduisant $k$ de $f(v)$. On arrête si $k < 0$ ou si $|E| = 0$.

Ceci créé un arbre de récursion où chaque noeud a 2 enfants. Puisque l'algorithme arrête si $k < 0$ et que chaque récursion réduit $k$ de $w = \min_{x \in V} f(x)$, la profondeur est bornée par $k/w$. Le nombre de noeuds de l'arbre est donc borné par $O(2^{k/w}(n + m))$, où le $n + m$ vient du temps pour retirer $u$ ou $v$.

---

*Exercice* 3. Considérez l'algorithme bidon suivant.

**fonction** *bidon*$(I, k)$
Si $k < 0$, return $false$
Si $I$ satisfait une propriété $X$, return $true$
Créer quatre sous-instances $I_1, I_2, I_3, I_4$ d'une façon non-spécifiée
$S_1 = bidon(I_1, k - 1)$
$S_2 = bidon(I_2, k - 3)$
$S_3 = bidon(I_3, k - 3)$
$S_4 = bidon(I_4, k - 4)$
Si un de $S_1, S_2, S_3, S_4$ n'est pas *null*, le retourner, sinon retourner *null*

En notation $O$, combien de noeuds l'arbre de récursion de cet algorithme a-t-il?

---

**Solution.**

On peut exprimer le nombre de noeuds par la récurrence

$$t(k) = t(k-1) + 2t(k-3) + t(k-4)$$

Le polynôme caractéristique est

$$c^k - c^{k-1} - 2c^{k-3} - c^{k-4} = 0$$

qui devient

$$c^4 - c^3 - 2c - 1 = 0$$

avec wolfram, on obtient la plus grande racine réelle $c \simeq 1.794$. Le nombre de noeuds est donc $O(1.794^k)$.

---

*Exercice* 4. Nous allons améliorer la complexité de l'algorithme de branchement de VERTEX-COVER vu en classe.

1. Soit $G$ un graphe dans lequel chaque sommet a $0, 1$ ou $2$ voisins. Il est possible de montrer que dans $G$, chaque composante connexe est un chemin ou un cycle. Montrez que VERTEX-COVER peut se résoudre en temps polynomial sur un tel graphe.

2. Donnez un algorithme en temps $O(1.47^k \cdot n^c)$ pour VERTEX-COVER paramétré par $k$, la taille de la solution.

---

**Solution.**

a) Supposons que chaque composante est un chemin ou un cycle. On peut trouver une couverture optimale pour chaque composante $C$ séparément.

Si $C$ a un seul sommet (un cas particulier d'un chemin), il n'y a pas d'arête à couvrir. Si $C$ est un chemin avec 2 sommets ou plus, soit $u$ un sommet de degré 1 sur $C$. On peut supposer que le voisin $v$ de $u$ est dans une couverture optimale $X$, car si $v$ n'y est pas, alors $u$ doit être dans $X$. On peut alors

échanger $u$ et $v$ dans $X$ et avoir une couverture de la même taile. Donc, étant donné le chemin $C$, on ajoute $v$, un voisin d'un sommet de degré 1, puis on retire $v$ et ses arêtes. On répète sur $C$ jusqu'à ce qu'on ait couvert toutes les arêtes de $C$.

Supposons finalement que $C$ est un cycle. Puisque tous les sommets de $C$ sont identiques, pour tout $v \in C$, il existe une solution optimale qui contient $v$. C'est-à-dire, le premier sommet qu'on décide d'inclure à une couverture par sommets sur un cycle est arbitraire. On ajoute donc à $X$ n'importe quel sommet $v$ de $C$, puis on retire $v$ et ses arêtes. Ensuite, $C$ est un chemin qu'on peut résoudre avec la procédure ci-haut.

b) On sait comment gérer les graphes de degré maximum 2 ou moins. On peut donc supposer qu'il y a un sommet de degré 3 et l'utiliser pour brancher.

```
fonction vc3(G = (V, E), k, X)
    si k < 0 alors
        return null
    fin
    si G n'a pas d'arête alors
        return X
    fin
    si G a un degré maximum 2 ou moins alors
        Résoudre G optimalement avec la procédure ci-haut
        Soit X' le couverture optimale sur G
        si |X'| > k alors
            return null
        sinon
            return X ∪ X'
        fin
    fin

    Soit u un sommet de degré au moins 3
    Obtenir G_u en retirant u et ses arêtes de G
    X_1 = vc3(G_u, k − 1, X ∪ {u})

    Obtenir G* en retirant {u} ∪ N(u) et leurs arêtes de G
    X_2 = vc3(G*, k − |N(u)|, X ∪ N(u))

    si un de X_1 ou X_2 n'est pas null alors
        le retourner
    sinon
        return null
    fin
```

Ce qui a changé, c'est que le $u$ sur lequel on branche a au moins 3 voisins. Donc, le deuxième appel récursif réduit $k$ d'au moins 3. Le nombre de noeuds de l'arbre de récursion est borné par

$$t(k) = t(k-1) + t(k-3)$$

Comme on a vu, ceci est $O(1.47^k)$. La complexité totale est donc $O(1.47^k(n+m))$.

*Exercice* 5. Considérez le problème MAX-INDSET, que nous avons déjà rencontré en approximation. La version paramétrée est la suivante.

MAX-INDSET
**Entrée :** graphe $G = (V, E)$
**Paramètre :** $k$, le taille d'un ensemble indépendant
**Sortie :** un ensemble indépendant $X \subseteq V$ de taille $k$, ou null si non-existant

Ce problème est $W[1]$-complet. Par contre, si le degré de chaque sommet est borné, on peut développer un algorithme FPT.

Supposons que chaque sommet de $G$ a au plus 4 voisins. Montrez qu'il existe un algorithme en temps $O(5^k \cdot n^c)$ pour ce problème.

Si vous préférez, vous pouvez montrer que MAX-INDSET pour se résoudre en temps $O((d+1)^k \cdot n^c)$, où $d$ est le degré maximum du graphe.

*Suggestion :* utilisez le fait que si $X$ est un ensemble indépendant maximum, alors pour tout sommet $v \in V(G)$, on a $v \in X$, ou bien un voisin de $v$ est dans $X$. Idéalement, argumentez aussi ce fait.

---

**Solution.**

Soit $X$ un ensemble indépendant maximum et soit $v \in V(G)$. Si $v \notin X$ et que aucun voisin de $v$ n'est dans $X$, on pourrait alors ajouter $v$ à $X$. On peut donc supposer que pour tout $v \in V(G)$, au moins un de $N(v) \cup \{v\}$ est dans $X$. Ceci est vrai indépendamment des degrés.

Pour notre algorithme, on choisit un $v$ arbitraire et on branche sur $d+1$ scénarios, chacun correspondant à l'inclusion d'un élément $w$ de $N(v) \cup \{v\}$. On retire $w$ et ses voisins et on arrête s'il n'y a plus de sommets, on si on a atteint une solution de taille $k$.

La raison pour laquelle ceci fonctionne est que si on prend une solution optimale $X$, un des cas de branchement incluera un sommet qui se trouve dans $X$ (par l'argument énoncé ci-haut).

```
fonction indsetFPT(G = (V, E), k, X)
    si k < 0 alors
        return null
    fin
    si |X| ≥ k alors
        return X
    fin
    si G n'a pas de sommets alors
        return null
    fin
    Soit u ∈ V(G)
    pour chaque w ∈ N(u) ∪ {u} faire
        Soit G' obtenu de G en enlevant w et ses voisins
        X' = indsetFPT(G', k, X ∪ {w})
        si |X'| ≠ null alors
            return X'
    fin
    return null
```

Notez qu'on ne décrémente pas $k$ aux appels récursifs. Par contre, on augmente $|X|$. Puisqu'on arrête lorsque $|X| \geq k$, la profondeur de l'arbre de récursion est bornée par $k$. Chaque sommet de l'arbre a au plus $d+1$ enfants, et donc la complexité est bornée par $O((d+1)^k \cdot dn)$, où $dn$ vient du temps pour retirer un sommet et ses voisins, opération que l'on répète $O(d)$ fois dans la boucle.

---

*Exercice* 6. Un graphe est *planaire* si on peut le dessiner en deux dimensions sans que deux arêtes ne se croisent. Cette propriété n'est pas fondamentale pour l'instant. On a plutôt deux propriétés importantes sur les graphes planaires :

- si on enlève un sommet d'un graphe planaire, on obtient un autre graphe planaire;
- un graphe planaire a toujours un sommet qui a 5 voisins ou moins.

Montrez que le problème MAX-INDSET de l'exercice précédent peut se résoudre en temps $O(6^k \cdot n^c)$ sur els graphes planaires.

---

**Solution.**

L'idée est la même que l'exercice précédent, à l'exception qu'au lieu de choisir un sommet $u$ arbitraire, on choisit $u$ tel que $deg(u) \leq 5$. On branche sur l'inclusion de chaque $w \in N(u) \cup \{u\}$ en retirant $w$ et ses voisins. Les propriétés des graphes planaires garantissent que l'on trouvera toujours un sommet $u$ de degré au plus 5, même après avoir branché et retiré un sous-ensemble de sommets.

En bref, il suffit de prendre l'algorithme de l'exercice précédent et de changer la ligne

"Soit $u \in V(G)$"

par la ligne

"Soit $u \in V(G)$ tel que $|N(u)| \leq 5$"

Le reste est identique. On note que chaque noeud de l'arbre de récursion aura au plus 6 enfants et une profondeur au plus $k$, ce qui donne un algorithme $O(6^k n)$.

---

*Exercice* 7. Dans le problème MAX-SAT, on a en entrée un ensemble de clauses $C_1, \ldots, C_m$ sur variables $x_1, \ldots, x_n$. On veut savoir s'il existe une assignation des $x_i$ qui satisfait au moins $k$ clauses.

Donnez un algorithme FPT en le paramètre $k$ pour ce problème. Un algorithme $O(2^k \cdot (n+m))$ n'est pas extrêmement difficile. Avec un peu plus de travail, on peut atteindre $O(1.618^k \cdot (n+m))$.

---

**Solution.**

Soit $x_i$ une variable. Si toutes les occurrences de $x_i$ sont positives, on peut automatiquement mettre $x_i = True$, enlever les clauses satisfaites et réduire $k$. Il en est de même si toutes les occurrences de $x_i$ sont négatives.

On peut donc supposer que pour chaque variable $x_i$, il y a au moins une occurrences positive de $x_i$ dans une clause $C_j$, et une occurrence négative dans une clause $C_h$. On branche sur $x_i = True$ et $x_i = False$, et chaque cas peut décrémenter $k$ d'au moins 1 car on ajoute une clause satisfaite.

**fonction** *maxsatFPT*($C = \{C_1, \ldots, C_m\}, k, A$)
  //$C$ est l'ensemble des clauses, $A$ est l'assignation en cours
  **tant que** $\exists$ $x_i$ *telle que toutes les occurrences sont du même type (positif ou négatif)* **faire**
    Assigner $x_i$ dans $A$ selon son type d'occurrence
    Retirer de $C$ les clauses $C_{x_i}$ satisfaites par $x_i$
    $k = k - |C_{x_i}|$
  **fin**
  **si** $k \leq 0$ **alors**
    return $A$  `// Il est trivial de satisfaire au moins` $k$ `clauses si` $k \leq 0$
  **fin**
  **si** $k > 0$ *et* $m = 0$ **alors**
    return *null*
  **fin**
  Soit $x_i$ une variable non-assignée
  Soient $C^+$ les clauses satisfaites si $x_i = true$
  Soit $A^+$ obtenu de $A$ avec $x_i = true$
  $A_1 = maxsatFPT(C \setminus C^+, k - |C^+|, A^+)$

  Soient $C^-$ les clauses satisfaites si $x_i = false$
  Soit $A^-$ obtenu de $A$ avec $x_i = false$
  $A_2 = maxsatFPT(C \setminus C^-, k - |C^-|, A^-)$

  Si $A_1$ ou $A_2$ n'est pas *null*, le retourner avec la valeur de $x_i$ appropriée, sinon retourner *null*

L'arbre de récursion a une profondeur au plus $k$ et chaque noeud a 2 enfants. L'arbre a donc $O(2^k)$ noeuds et la complexité total est bornée par $O(2^k \cdot n \cdot m)$.

Il n'est pas immédiat que cet algorithme fonctionne. Il faut le démontrer — prenons le temps de le faire!

*Theorem* 1. L'algorithme est correct.

*Proof.* Il faut montrer que si l'algorithme retourne une assignation, elle satisfait $k$ clauses, et si l'algorithme retourne *null*, il n'y avait pas de solution. Notons d'abord qu'il est clair que la boucle au départ ne peut pas enfreindre cette condition.

On procède par induction sur la hauteur de l'arbre de récursion.

Comme cas de base, si l'arbre de récursion est de hauteur 1, il n'y a pas d'appel récurfis. Ceci est possible si $k \leq 0$, dans quel cas toute assignation $A$ satisfait trivialement au moins $k$ clauses et donc le retour est correct. Une autre possibilité est $k > 0$ et $m = 0$. Dans ce cas, il est correct de retourner $null$ car il est impossible de satisfaire $k > 0$ clauses parmi 0.

Considérons un appel dans l'arbre dont la hauteur est plus grande que 1. Donc l'appel fait des récursions. On suppose par induction que les deux appels récursif retournent une solution correcte. Soit $x_i$ la variable choisie. Supposons qu'il est impossible de satisfaire $k$ clauses de $C$. Alors peu importe si on assigne $x_i$ avec $true$ ou $false$, il sera impossible de satisfaire $k - |C^+|$ clauses si $x_i = true$, ou de satisfaire $k - |C^-|$ clauses si $x_i = false$. Par induction, les deux appels récursifs retourneront correctement $null$ et notre algorithme retournera correctement $null$.

Supposons plutôt qu'il est possible de satisfaire $k$ clauses de $C$ par une assignation $A^*$. Supposons que $x_i = true$ dans $A^*$. Puisque $A^*$ satisfait au moins $k$ clauses, il faut que $A^*$ puisque satisfaire au moins $k - |C^+|$ clauses parmi $C \setminus C^+$. Par induction, la récursion retournera correctement une assignation qui satisfait au moins $k - |C^+|$ clauses de $C \setminus C^+$, car elle existe. L'assignation retournée par l'algorithme va donc satisfaire au moins $k - |C^+| + |C^+| = k$ clauses. Si $x_i = false$ dans $A^*$, la preuve est identique. □

Pour obtenir $O(1.618^k n^c)$, il faut trouver $x_i$ qui a au moins 3 occurrences dans des clauses, et un des appels réduira $k$ de 2 ou plus. Il faut argumenter que si un tel $x_i$ n'existe pas, on peut résoudre en temps polynomial.

---

# Kernelisation

*Exercice* 8. Dans le problème SET-SPLITTING, on a en entrée des ensembles $S = \{S_1, \ldots, S_n\}$ sur un univers $U$. On veut attribuer une couleur $f : U \to \{R, B\}$ à chaque élément. Notre but est qu'il y ait au moins $k$ ensembles contenant au moins un élément de chaque couleur (donc, s'arranger pour que $k$ ensembles contiennent un élément $B$ et un élément $R$ avec notre coloriage).

Donnez un noyau pour SET-SPLITTING.

---

**Solution.**

On va d'abord supposer que $S$ ne contient aucun ensemble de taille 1, car il ne peuvent pas contribuer au nombre d'ensembles sur deux couleurs.

On argumente ensuite que s'il y a $n$ ensembles, on peut toujours trouver une coloration avec au moins $m/2$ ensemble avec deux couleurs. Pour le voir, on colorie chaque élément aléatoirement. Si on prend un ensemble $S_i$ de taille $t$, il a seulement deux façons qu'il soit d'une seule couleur, sur $2^t$ façons de le colorier. Il y a donc une probabilité $2/2^t$ que $S_i$ soit monochromatique. Puisque $t \geq 2$, cette probabilité est au plus $1/2$. Donc, on a au moins $1/2$ chance que $S_i$ soit sur deux couleurs. En espérance, on aura alors au moins $m/2$ ensembles sur deux couleurs. Si on l'exige, on peut trouver une telle coloration par dérandomisation. Il y a aussi d'autres façons de trouver une telle coloration (par exemple une coloration gloutonne).

Ceci étant dit, si $k \leq m/2$, on peut retourner une coloration avec le nombre souhaité d'ensembles. Si $k > m/2$, alors $m < 2k$ et donc le nombre d'ensembles est au plus $2k$.

Ceci est presque suffisant, mais il faut aussi borner la taille de l'univers, qui est $|\bigcup_{S_i \in S} S_i|$. Ma façon est d'utiliser la règle suivante.

**Règle.** S'il existe $S_i \in S$ tel que $|S_i| > 2k$, alors retirer $S_i$ et réduire $k$ de 1.

La raison est que si $S_i$ est assez grand, on pourra toujours trouver un de ses éléments pour le forcer à être sur deux couleurs. La preuve de sanité de la règle va comme suit. Si $S$ admet une coloration avec $k$ ensembles deux couleurs, il est évident que la même coloration est telle que $S \setminus \{S_i\}$ a au moins $k-1$ ensembles deux couleurs. Conversément, supposons que $S \setminus \{S_i\}$ admet une coloration avec au moins $k-1$ ensembles deux couleurs. Soient $S'_1, \ldots, S'_{k-1}$ ces ensembles. Alors chaque $S'_j$ contient deux éléments $a_j, b_j$ de couleur différente. On peut garder la même coloration pour $a_1, b_1, \ldots, a_{k-1}, b_{k-1}$, et ce coloriage partiel

d'au plus $2(k-1)$ éléments est suffisant pour qu'ils restent deux couleurs. Puisque $|S_i| > 2k$, l'ensemble $S_i$ contient au moins deux élément $c, d$ qui ne sont pas parmi les $a_j, b_j$. On colorie $c$ et $d$ différemment et on obtient la solution voulue pour $S$.

En supposant que la règle est appliquée et que $m < 2k$, on a $2k$ ensembles de taille au plus $2k$, et donc un noyau sur univers de taille au plus $4k^2$.

---

*Exercice* 9. Donnez un noyau pour CLUSTER-EDITING, où le paramètre est le nombre d'arêtes à modifier (ajouter et supprimer) pour obtenir un graphe où chaque composante connexe est une clique.

*Indice.* Considérez trois règles de réduction: 1) si $v \in V$ n'est dans aucun $P_3$, supprimer $v$; 2) si $uv \in E$ est dans $k+1$ $P_3$, supprimer $uv$; 3) si $uv \notin E$ est dans $k+1$ $P_3$, ajouter $uv$.

Montrez qu'après avoir appliqué ces règles, le nombre de sommets est borné par une fonction de $k$. Pour bien faire les choses, vous devriez argumenter que ces règles sont saines.

---

**Solution.**

Nous allons brièvement argumenter que les trois règles citées ci-haut sont saines.

**Règle 1.** Si $v$ n'est dans aucun $P_3$, supprimer $v$ ($(G,k) \to (G-v,k)$).

Supposons que $v$ n'est dans aucun $P_3$. Il est facile de voir que $N(v)$ doit être une clique. De plus, pour tout $w \in N(v)$, le sommet $w$ ne peut pas avoir de voisin en dehors de $N(v)$. Ceci est parce que si $w$ avait un voisin $z \notin N(v)$, alors $v, w, z$ serait un $P_3$. On déduit que si $v$ n'a aucun $P_3$, la composante connexe qui contient $v$ est une clique. Il est donc clair qu'une solution de touchera pas à cette clique. On pourrait le retirer complètement en préservant l'équivalence et, en particulier, on peut supprimer $v$.

Pourquoi ne pas poser la règle comme "si une composante est une clique, la retirer"? Seulement parce que la règle 1 ci-haut facilite l'argument plus tard.

**Règle 2.** si $uv \in E$ est dans $k+1$ $P_3$, supprimer $uv$.

Soient $(u, v, w_1), \ldots, (u, v, w_{k+1})$ les $P_3$ contenant $uv$. S'il existe une solution pour $G$, elle doit supprimer $uv$, car sinon chaque $(u, v, w_i)$ demandera une modification différente. On peut déduire que si $(G, k)$ admet une soution, il faut que $(G-uv, k-1)$ admette une solution. Conversément, si $G-uv$ admet

une solution qui modifie un ensemble $A$ de $k-1$ arêtes, alors $A \cup \{$"supprimer $uv\}$ modifie $k$ arêtes de $G$ et résulte en le même graphe.

**Règle 3.** si $uv \notin E$ est dans $k+1$ $P_3$, ajouter $uv$.

Ceci est similaire au cas précédent. Soient $(u, w_1, v), \ldots, (u, w_{k+1}, v)$ les $P_3$ impliquant la non-arête $u, v$. Toute solution de $G$ doit ajouter $uv$, et donc si $G$ a une solution avec $k$ arêtes arêtes modifiées, alors $G + uv$ a une solution avec $k-1$ arêtes modifiées. Conversément, une solution sur $G + uv$ avec $k-1$ arêtes modifiées est une solution pour $G$ avec $k$ arêtes modifiées, après l'ajout de $uv$.

**Taille de noyau.** Supposons qu'on a appliqué les règles de réduction ci-haut sur $G$ jusqu'à ce qu'on ne puisse plus. On argumente que si $G$ admet une solution, alors $G$ avec $O(k^2)$ sommets et $O(k^4)$ arêtes.

Par la règle 2, chaque arête participe à au plus $k$ $P_3$. Donc, supprimer une arête peut éliminer au plus $k$ $P_3$. Par le règle 3, chaque non-arête peut éliminer au plus $k$ $P_3$ en l'ajoutant. Bref, une modification élimine au plus $k$ $P_3$, et il s'ensuit que dans $G$, le nombre de $P_3$ est au plus $k^2$.

Par la règle 1, chaque sommet participe à au moins un $P_3$. Donc, si on prend l'union des sommets de tous les $P_3$, on a tous les sommets. Puisque chaque $P_3$ contient trois sommets, l'union de ces $k^2$ $P_3$ donne que $G$ a au plus $3k^2 \in O(k^2)$ sommets. Le nombre d'arêtes est ensuite borné par $O((k^2)^2) = O(k^4)$.

---

*Exercice* 10. Donnez un noyau pour le problème 3-HITTING-SET.

*Indice.* D'abord, montrez que si deux éléments $x$ et $y$ appraissent dans beaucoup de triplets ensemble, alors on peut simplement enlever un de ces triplets. Ma définition de beaucoup est $k+2$, mais d'autres peuvent fonctionner.

Ensuite, montrez qu'après avoir fait les éliminations ci-haut, s'il y a un élément $x$ qui est dans plus de $f(k)$ triplets, il faut que $x$ soit dans la solution. On peut donc retirer $x$ et réduire $k$ de 1. Ici, ma définition de $f(k)$ est $k \cdot (k+1) + 1$.

Finalement, montrez qu'après application de ces règles, chaque $x$ peut couvrir une quantité $f(k)$ de triplets, et donc qu'il doit y avoir au plus $k \cdot f(k)$ triplets.

---

**Solution.**

Soient $S = \{S_1, \ldots, S_m\}$ les triplets et soit $U$ l'univers. Une première règle consiste à éliminer la redondance en regardant les paires $x, y$ qui apparaissent souvent ensemble.

**Règle 1.** S'il existe $x, y \in U$ tel qu'il existe $k+2$ ensembles distincts $\{x, y, b_1\}, \ldots, \{x, y, b_{k+2}\}$, alors supprimer $\{x, y, b_{k+2}\}$. On ne réduit pas $k$.

L'idée est que s'il y a $k+2$ triplets qui ont $x$ et $y$, nous sommes obligés d'inclure $x$ ou $y$ dans une solution. Si on supprime $\{x, y, b_{k+2}\}$, ceci reste vrai et la nature d'une solution de taille $k$ ou moins reste inchangée. Voici la façon rigoureuse d'argumenter qu'on préserve l'équivalence.

Soit $S = S' \setminus \{\{x, y, b_{k+2}\}\}$. Suposons qu'il existe un hitting set $X$ de $S$ de taille $k$. Puisque $S' \subset S$, il est clair que $X$ est un hitting set de taille $k$ de $S'$. Conversément, soit $X'$ un hitting set de $S'$. On a soit $x \in X'$, ou $y \in X'$, à cause des $k+1$ triplets $\{x, y, b_i\}$ (si on n'inclut ni $x$ ni $y$, il faut inclure tous les $b_i$ et il y en a trop). Dans ce cas, $X'$ est aussi un hitting set pour $S$ car $\{x, y, b_{k+2}\}$ est couvert par $x$ ou $y$.

On suppose qu'on applique la règle 1 jusqu'à épuisement. On sait maintenant que chaque paire $x, y$ apparaît dans au plus $k+1$ triplets. Ceci nous permet de borner le nombre d'occurrence d'un seul élément.

**Règle 2.** Supposons que la Règle 1 a été appliquée. S'il existe $x \in U$ tel que $x$ est présent dans $k \cdot (k+1) + 1$ triplets, alors inclure $x$ dans la solution. C'est-à-dire, retirer tous les triplets contenant $x$ et réduire $k$ de 1.

On argumente qu'un tel $x$ doit être dans la solution. Supposons que non, et qu'il y a un hitting set $X$ de taille $k$ et que $x \notin X$. Soient $B$ les triplets qui contiennent $x$. Alors la solution couvre $B$ mais pas avec $x$. Or, par la règle 1, chaque $b_i$ apparaît au plus $k+1$ fois avec $x$. Donc, chaque $b_i$ couvre au plus $k+1$ triplets de $B$. S'il y a plus de $k(k+1)$ triplets qui contiennent $x$, on déduit que $k$ éléments autres que $x$ ne suffisent pas à couvrir $B$. Donc, $x$ doit être dans la solution.

Puisque $x$ est forcé dans toute solution, il n'est pas difficile de voir qu'inclure $x$ et réduire $k$ de 1 donne une instance équivalente.

**Taille de noyau.** Supposons que la Règle 1 et la Règle 2 ont été appliquées. Dans ce cas, chaque $x \in U$ peut couvrir au maximum $k(k+1)$ triplets. Donc, s'il y a un hitting set de taille $k$, le nombre de triplets doit être au plus $k \cdot k(k+1) \in O(k^3)$. Puisque chaque triplet a 3 éléments, le nombre d'éléments de l'univers est $O(3k^3) = O(k^3)$.