Tout problème de 3-coloration de graphe est soluble en temps polynomial

OLIVET KARL

Décembre 2023

1 Introduction

En théorie des graphes, la coloration de graphe consiste à attribuer une couleur à chacun de ses sommets de manière que deux sommets reliés par une arête soient de couleur différente. On appelle nombre chromatique le nombre minimal de couleurs.

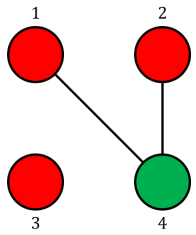


Figure 1: Possibliter de coloration chromatique d'un graphe de quatre noeuds et dont l'ensemble des arêtes est $\{(1,4),(2,4)\}$

Ainsi le problème de la trois-coloration de graphe consiste à déterminer s'il est possible de colorer les sommets d'un graphe avec au plus trois couleurs de manière à respecter la règle de non-adjacence de couleurs identiques.

Un graphe est 3-coloriable si le nombre chromatique est inférieur ou égales à 3.

2 Définition

Soit G un graphe avec V l'ensemble des noeuds et E l'ensemble des arêtes.

On note :

- $n = card(V)$ (nombre de noeuds)
- $E \subseteq \{a, b \in V \mid a \neq b\} = E_{complet}$ (définition de E)
- $\bar{E} = E_{complet} \setminus E$ (définition du complementaire de E)

Si l'on définit b comme une application d'ensemble telle que pour un $k \in [\![1, n]\!]$ elle nous associe l'ensemble des k-noeuds qui ne sont pas relier entre eux alors :

$$b_1 = \{\beta \in V\}$$

et

$$b : \begin{array}{c} [\![2, n]\!] \to V^n \\ k \mapsto \left\{\beta_1, ..., \beta_k \in V \mid (i, j) \in [\![1, k]\!]^2, i \neq j, (\beta_i, \beta_i) \in \bar{E}\right\} \end{array}$$

Puis l'on définit A comme une application d'ensemble telle que pour un $q \in [\![1, n]\!]$ elle nous associe l'ensemble dont les éléments sont des q groupe de noeuds ,telle que dans c'est groupe les noeuds ne sont pas relié entre eux, dont la réunion nous donne l'ensemble V et dont l'intersection est vide alors:

En notant :

$$B = \bigcup_{j=1}^{n} b_k$$

Alors:

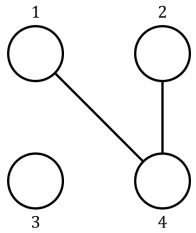
$$A : \begin{array}{c} [\![1, n]\!] \to I \subseteq B^n \\ q \mapsto \left\{a_1, ..., a_q \in B \mid \left(\bigcup_{j=1}^{n} a_j = V\right) \wedge (i, j \in [\![1, q]\!], \quad a_i \neq a_j)\right\} \end{array}$$

On peut alors définir le nombre chromatique $\chi(G) \in [\![1, n]\!]$ par la relation suivante :

$$A_{\chi(G)} \neq \varnothing \text{ et } A_{\chi(G)-1} = \varnothing$$

3 Exemple

Soit le graphe de l'introduction :



$V = \{1, 2, 3, 4\}$ $E = \{(1, 4), (2, 4)\}$

$E_{complet} = \{(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)\}$ $\bar{E} = \{(1, 2), (1, 3), (2, 3), (3, 4)\}$

$b_1 = \{(1), (2), (3), (4)\}$ $b_2 = \{(1, 2), (1, 3), (2, 3), (3, 4)\}$

$b_3 = \{(1, 2, 3)\}$ $b_4 = \varnothing$

$B = \{(1), (2), (3), (4), (1, 2), (1, 3), (2, 3), (3, 4), (1, 2, 3)\}$

$A_1 = \varnothing$ $A_2 = \{[(1, 2, 3), (4)], [(1, 2), (3, 4)]\}$

$A_3 = \{[(1, 2), (3), (4)], [(1, 3), (2), (4)], [(2, 3), (1), (4)]\}$ $A_4 = \{[(1), (2), (3), (4)]\}$

Ainsi ce graphe est 3-coloriable est sont nombre chromatique est de 2.

4 Analyse

Si l'on souhaite savoir si un graphe est 3-coloriable alors il suffit que $A_3 \neq \varnothing$.

Il faut qu'il existe $\alpha_x \cup \beta_y \cup \gamma_z \in B$ deux à deux distinct telle que :

$$\alpha_x \cup \beta_y \cup \gamma_z = V \text{ avec } \alpha_x \in b_x, \beta_y \in b_y, \gamma_z \in b_z \ (1)$$

4.1 Calcul du nombre de possibliter de solution à (1)

Une première condition sympose :

$$card(\beta_x \cup \beta_y \cup \beta_z) = n$$

c'est-à-dire

$$card(\beta_x) + card(\beta_y) + card(\beta_z) = n$$

car $\beta_x, \beta_y, \beta_z$ sont deux à deux distinct.

Sachant que $card(\beta_x), card(\beta_y), card(\beta_z) \in [\![1, n-2]\!]$.

On peut alors reformuler le problème :

Soit $x, y, z \in [\![1, n-2]\!]$ telle que $x + y + z = n$.

On cherche maintenant combien de solutions à cette équation.

Mais cette équation est un cas particulier de ce problème :

Soit $p \in \mathbb{N}^*$ et $n \in \mathbb{N}$, on note Γ_p^n le nombre de n-uplets $(x_1, ..., x_p) \in \mathbb{N}^p$ tels que $x_1 + ... + x_p = n$.

1. Démontrons que, pour tout $p \in \mathbb{N}^*$, pour tout $n \in \mathbb{N}$.

$$\Gamma_{p+1}^n = \Gamma_p^0 + \Gamma_p^1 + ... + \Gamma_p^n$$

2. Puis on en déduira que, pour tout $p \in \mathbb{N}^*$, pour tout $n \in \mathbb{N}$.

$$\Gamma_p^n = \binom{p+n-1}{n}$$

Démonstration :

1) Le nombre de $p+1$-uplets $(x_1, ..., x_{p+1})$ tels que $x_1 + ... + x_{p+1} = n$ et $x_{p+1} = k$ vaut exactement Γ_p^{n-k}, le nombre de p-uplets $x_1, ..., x_p$ tels que $x_1 + ... + x_p = n - k$.
On somme pour toutes les valeurs de k possibles, c'est-à-dire de 0 à n. On a donc

$$\Gamma_p^{n-0} + ... + \Gamma_p^{n-n} = \Gamma_{p+1}^n$$

ce qui était bien le résultat voulue.

2) Soit $\Gamma_p^n = \binom{p+n-1}{n}$ avec $p \in \mathbb{N}^*$ et $n \in \mathbb{N}$.

On va procéder par récurrence sur p.

Initialisation : Prenoms $p = 1$

$\Gamma_1^n = 1$ car x_1 doit être égal à n et $\binom{1+n-1}{n} = 1$.

Γ_1^n est vraie

Hérédité : Supposons que la propriété est prouvé jusqu'au rang p et prouvons-le au rang $p+1$. On a

$$\Gamma_{p+1}^n = \binom{p-1}{0} + \binom{p}{1} + ... + \binom{p+n-1}{n}$$

Il s'agit donc de prouver que :

$$\binom{p+n}{n} = \binom{p-1}{0} + \binom{p}{1} + ... + \binom{p+n-1}{n}$$

Or d'après la formule du triangle de Pascal :

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} \forall n \in \mathbb{N} \mid 1 \leq k \leq n-1$$

Donc $\binom{p+n}{n} = \binom{p-1}{0} + \binom{p}{1} + ... + \binom{p+n-1}{n}$ se prouve par application sucessives de la formule du triangle de Pascal. Plus précisement, on peut le démontrer par récurrence sur n. Si $n = 0$ ou 1, elle est vraie. Si elle est vraie jusqu'au rang $n-1$, alors

$$\binom{p-1}{0} + \binom{p}{1} + ... + \binom{p+n-2}{n-1} + \binom{p+n-1}{n} = \binom{p+n-1}{n-1} + \binom{p+n-1}{n} = \binom{p+n}{n}$$

Le prédicat étant vrai au rang 1 et héréditaire, il est alors vrai pour tout $n \in \mathbb{N}$.

□

Applicons maintenant le résultat optenue à notre équation de départ qui était la suivante :

Soit $x, y, z \in \mathbb{N}^*$ et $x + y + z = n$
Si l'on procède au changement de variable suivant :

$$X = x - 1, Y = y - 1, Z = z - 1$$

alors on optient l'équation suivante a résoudre :

Soit $X, Y, Z \in \mathbb{N}, X + Y + Z = n - 3$.

Cette équation a Γ_3^{n-3} solutions soit $\binom{n-1}{n-3}$ solutions.

Or $\binom{n-1}{n-3} = \frac{(n-1)!}{(n-3)!(2)!} = \frac{(n-1)(n-2)}{2}$

Donc l'équation

$$card(\beta_x) + card(\beta_y) + card(\beta_z) = n$$

a $\frac{(n-1)(n-2)}{2}$ solutions.

4.2 Nouvelle expression de l'application d'ensemble b

Comme la sous partie 4.1 a pu nous le montrer il faudra au plus vérifié $\frac{(n-1)(n-2)}{2}$ triplet (même si celui si pourrais être d'avantage reduit du fait de la commutativité de l'opperation union; mais cette encadrement nous est suffisant car il est polynomiale)

Mais un problème vient a nous on ne connait pas une meuilleur majoration du $card\,\langle b\rangle$ que par son ensemble image qui est par définition $[\![1,n]\!]^n$ est dont le cardinal est egale à n^n ce qui n'est pas polynomiale.

Pour trouver une majoration polynomiale du $card\,\langle b\rangle$ il va falloir une nouvelle expression de b.

Soit :

$$b_1 = \{\beta \in V\}$$

et

$$b : \begin{array}{c} [\![2,n]\!] \to V^n \\ k \mapsto \left\{\beta_1, ..., \beta_k \in V \mid (i,j) \in [\![1,k]\!]^2, i \neq j, (\beta_i, \beta_i) \in \bar{E}\right\} \end{array}$$

Si l'on introduit $k_{max} \in [\![1,n]\!]$ telle que $b_{k_{max}} \neq \varnothing$ et $b_{k_{max}+1} = \varnothing$ alors sachant le $card(b_1) = n$ et tous les $j \in [\![k_{max}+1, n]\!], b_j = \varnothing$. On va donc s'intéréser à b restreint à $[\![2, k_{max}]\!]$.
Si on introduit P telle que

$$P : \begin{array}{c} [\![1,n]\!] \times [\![1,n]\!]^n \to \mathscr{P}([\![1,n]\!]) \\ k, A \mapsto \{a \in \mathscr{P}(A) \mid card(a) = k\} \end{array}$$

Alors : $\forall k \in [\![2, k_{max}]\!]$

$$b_k = \Phi \cup P_k\,\langle b_{k+1}\rangle \text{ avec } \Phi \subseteq [\![1,n]\!]^n$$

Cherchons a quoi peut correspond Φ. Par définition de b :

$$\Phi = \{\beta_1, ..., \beta_k \in V \mid \forall i \neq j, (\beta_i, \beta_j) \in E_{max} \setminus [E \cup P_k\,\langle b_{k+1}\rangle]\}$$

Or cela est équivalent à trouver le "$b_{k_{max}}$" du graphe :

$$G_{(k)} : \begin{matrix} V_{(k)} = V \\ E_{(k)} = E \cup P_k \langle b_{k+1} \rangle \end{matrix}$$

Si l'on note $\Phi_{(k)}$ le "$b_{k_{max}}$" du graphe $G_{(k)}$ alors :

$$b_k = \Phi_{(k)} \cup P_k \langle b_{k+1} \rangle$$

Exemple : Si on se réfère à "3 Exemple" alors :

$$b_2 = \Phi_{(2)} \cup P_2 \langle b_3 \rangle \text{ avec } P_2 \langle b_3 \rangle = \{(1,2), (1,3), (2,3)\}$$

On a donc

$$G_{(2)} : \begin{matrix} V_{(2)} = V \\ E_{(2)} = E \cup P_2 \langle b_3 \rangle \end{matrix}$$

et

$$\Phi_{(2)} = \{(3,4)\}$$

Conclusion :

$\forall k \in [\![2, k_{max}]\!]$

$$b_k = \Phi_{(k)} \cup P_k \langle b_{k+1} \rangle$$

Or $b_2 = \bar{E}$ alors de cette expression on peut en déduire :

$\forall k \in [\![3, k_{max}]\!]$

$$b_k = \{ A_1 \cup A_2 \subset V^k \mid A_1 \in b_{k-1}, A_2 \in b_{k-1}, A_1 \neq A_2, A_1 = \{a_1, \ldots, a_{k-1}\}, A_2 = \{\alpha_1, \ldots, \alpha_{k-1}\}, \\ v \in [\![1, k-2]\!], w \in [\![1, k-2]\!], a_v = \alpha_w, \{a_{k-1}, \alpha_{k-1}\} \in \bar{E} \}$$

Exemple:

On a montrer dans l'exemple au dessus :

$$b_2 = \Phi_{(2)} \cup P \langle b_3 \rangle \text{ avec } \left| \begin{matrix} \Phi_{(2)} = (3,4) \\ P \langle b_3 \rangle = \{(1,2), (1,3), (2,3)\} \end{matrix} \right.$$

Or $b_3 = (1,2,3)$ et il n'y a pas $(2,3,4)$ dans b_3 car $(2,4) \notin \bar{E}$ mais $(2,4) \in E$.

4.3 Calcul d'une majoration polynomiale du cardinal de b restrain à $[\![2, k_{max}]\!]$

Dans la partie précédente nous avons déterminer une nouvelle expressions de b restrain à $[\![2, k_{max}]\!]$ car notre première aproximation du cardinal de b n'étais pas polynomiale quand b était restrain à $[\![2, k_{max}]\!]$.

Maintenant tachons de trouver une nouvelle aproximation du cardinal de b restrain à $[\![2, k_{max}]\!]$ qui serait polynomiale grace à la nouvelle expression de b déterminer dans la sous partie 4.2.

Démontrons par récurrence la propriété suivante soit $card(b_k) = O(n^2)$ avec $k \in [\![2, k_{max}]\!]$

Raisonons par récurrence sur k

Initialisation : Prenoms $k = 2$

$$b_2 = \bar{E} \text{ or } card(\bar{E}) = O(n^2)$$

La propriété est vraie pour $k = 2$

Hérédité : Supposons que la propriété est prouvé jusqu'au rang $k-1$ et prouvons-le au rang k.

Soit :

$$card(b_{k-1}) = card(\Phi_{(k-1)} \cup P_k \langle b_k \rangle) = card(\Phi_{(k-1)}) + card(P_k \langle b_k \rangle)$$

car $\Phi_{(k-1)} \cap P_k \langle b_k \rangle = \varnothing$

cela implique

$$card(\Phi_{(k-1)}) = O(n^2) \text{ et } card(P_k \langle b_k \rangle) = O(n^2)$$

Sachant

$$\begin{aligned} card(P_k \langle b_k \rangle) &= card\{a \in \mathscr{P} \langle b_k \rangle \mid card(a) = k\} \\ &= card\{a \subseteq [\![1, n]\!] \mid (l = card(b_k), \beta_1, .., \beta_l = b_k, i \in [\![1, l]\!], a \subseteq \beta_i) \wedge (card(a) = k)\} \end{aligned}$$

et $card(P_k \langle b_k \rangle) = O(n^2)$ cela implique

$$card(P_k \langle b_k \rangle) = card\left\{a \subseteq [\![1, n]\!] \mid \left(l = O(n^2) = card(b_k), \beta_1, .., \beta_l = b_k, i \in [\![1, l]\!], a \subseteq \beta_i\right) \wedge (card(a) = k)\right\}$$

donc

$$card(P_k \langle b_k \rangle) = O(n^2) \Rightarrow card(b_k) = O(n^2)$$

Le prédicat étant vrai au rang 2 et héréditaire, il est alors vrai pour tout $k \in [\![2, k_{max}]\!]$.

Conclusion : Ainsi on vient de démontrer que le card(b) sur sont domaine de définition est un $O(n^2)$.

5 Synthèse

Pour savoir si un graphe $G(V, E)$ est trois-coloriable il suffit :

$\exists \alpha_x \in b_x, \exists \beta_y \in b_y, \exists \gamma_z \in b_z$ deux à deux distinct telle que $\alpha_x \cup \beta_y \cup \gamma_z = V$

avec :

$b_1 = \{\beta \in V\}$

$b_2 = \bar{E}$

$\forall k \in [\![3, n]\!]$

$$b_k = \{A_1 \cup A_2 \subset V^k \mid A_1 \in b_{k-1}, A_2 \in b_{k-1}, A_1 \neq A_2, A_1 = \{a_1, \ldots, a_{k-1}\}, A_2 = \{\alpha_1, \ldots, \alpha_{k-1}\},$$
$$v \in [\![1, k-2]\!], w \in [\![1, k-2]\!], a_v = \alpha_w, \{a_{k-1}, \alpha_{k-1}\} \in \bar{E}\ \}$$

6 Algorithmique

6.1 Déduction d'un algorithme de complexité polynomiale pour trouver l'ensemble des valeurs de b restrein à $[\![1, k_{max}]\!]$

Nous avons démontrer :

$b_1 = \{\beta \in V\}$

$b_2 = \bar{E}$

$\forall k \in [\![3, n]\!]$

$$b_k = \{A_1 \cup A_2 \subset V^k \mid A_1 \in b_{k-1}, A_2 \in b_{k-1}, A_1 \neq A_2, A_1 = \{a_1, \ldots, a_{k-1}\}, A_2 = \{\alpha_1, \ldots, \alpha_{k-1}\},$$
$$v \in [\![1, k-2]\!], w \in [\![1, k-2]\!], a_v = \alpha_w, \{a_{k-1}, \alpha_{k-1}\} \in \bar{E}\ \}$$

Pour définir cette algorithme il nous faudra d'abord définir 2 fonction qui sont les suivantes:

Un algorithme permettant de compare deux listes d'éléments et retourne les éléments communs ainsi que leurs positions respectives dans chaque liste que l'on nommera "Compare". Et d'un autre qui verifie que le couple (a_{k-1}, α_{k-1}) n'appartient pas à E que l'on nommera "Appartenir"

Algorithme 2 : Compare

Entrées : deux listes d'entier de même longueur sans doublons, liste1 et liste2

Initialisation :
Une liste de listes appelée "positions_communes" avec trois sous-listes vides.
Une variable k initialisée à 0

Traitement :
Pour chaque élément "élément1" et son index "i" dans la liste1 :
    Si "élément1" est présent dans la liste2 :
        Trouver la position de "élément1" dans liste2 et stocker le résultat sous le nom "p_2".
        Ajouter "élément1" à la première sous-liste de "positions_communes".
        Ajouter l'index "i" à la deuxième sous-liste de "positions_communes".
        Ajouter "p_2" à la troisième sous-liste de "positions_communes".
        Incrémenter de 1 la variable k.
Si k est égal à la longueur de liste1 moins 1 :
    Retourner la liste "positions_communes".
Sinon, retourner Aucun.

Complexité de l'Algorithme 2 : Compare

Entrées : deux listes d'entier de même longueur sans doublons, liste1 et liste2

Initialisation :
O(1)
O(1)

Traitement :
O(n) car la liste est de longueur $k \leq n$:
    O(n) car la liste est de longueur $k \leq n$:
        O(n) car la liste est de longueur $k \leq n$
        O(1)
        O(1)
        O(1)
        O(1)
O(1)
    O(1)
O(1)

La complexité de cette algorithme est donc $O(n^3)$

Algorithme 3 : Appartenir
Entrées : Une liste composer de trois sous liste nommé "Comp" Trois liste d'entier de m : list1 ,list2 Une liste de couple d'entier deux à deux distcincts nommé "E"
Initialisation : Création de la liste a_{k-1} contenant le premier élément de list1 dont l'indice n'appartient pas à la deuxième sous-liste de Comp. Création de la liste α_{k-1} contenant le premier élément de list2 dont l'indice n'appartient pas à la deuxième sous-liste de Comp.
Traitement : Si la liste $[a_{k-1}, \alpha_{k-1}]$ appartient à la liste E :   Retourner Aucun. Sinon, retourner les entiers a_{k-1} et α_{k-1}.

Complexité de l'Algorithme 3 : Appartenir
Entrées : Une liste composer de trois sous liste nommé "Comp" Trois liste d'entier de m : list1 ,list2 Une liste de couple d'entier deux à deux distcincts nommé "E"
Initialisation : O(n) car la liste est de longueur $k \leq n$ O(n) car la liste est de longueur $k \leq n$
Traitement : O(longueur(E))=$O(n^2)$:   O(1) O(1)
La complexité de cette algorithme est donc $O(n^2)$

Ces deux fonctions étant réunit on peut en déduire un algorithme qui donnera B l'ensemble des b_k que l'on nommera "L'ensemble B"

Algorithme 4 : L'ensemble B

Entrées :
Un entier : n
Une liste de couple : E

Initialisation :
Création de la liste E_{bar} contenant toutes les paires d'indices [i, j] pour i de 0 à n-2 et j de i+1 à n-1 qui ne sont pas présentes dans E.
Création de la liste B initialement vide.
Pour chaque entier i de 0 à $n-1$, créer une liste contenant l'élément i et ajouter cette liste à B. b_0
Ajout de la liste E_{bar} à B en tant que deuxième élément. b_1
Initialisation de la variable k à 1.
Initialisation de la variable c à la longueur de b_1.

Traitement :
Tant que c est supérieur à 0 :
- Création d'une liste vide b_{bis}.
- Création d'une liste vide b.
- Obtention de la liste a à partir de la sous-liste de B correspondant à l'indice k.
- Affichage de la liste a.
- Pour chaque entier i de 0 à c-2 :
  - Pour chaque entier j de i+1 à c-1 :
    - Obtention de la liste A_i à partir de la sous-liste de a correspondant à l'indice i.
    - Obtention de la liste A_j à partir de la sous-liste de a correspondant à l'indice j.
    - Appel de la fonction Compare avec les listes A_i et A_j et stockage du résultat dans e_c_p.
    - Si e_c_p est égale à Aucun :
      - Continuer à la prochaine itération.
    - Sinon :
      - Appel de la fonction Appartenir avec les listes e_c_p, A_i, A_j, et E et stockage du résultat dans verif.
      - Si verif est égale à Aucun :
        - Continuer à la prochaine itération.
      - Sinon :
        - Ajout de la liste résultante de la concaténation triée du premier élement de e_c_p et de la liste composer du premier élément de verif et du second à b_{bis}.
- Pour chaque élément dans b_{bis}:
  - Si l'élément n'est pas déjà dans b, l'ajouter à b.
- Mise à jour de la variable c avec la longueur de la liste b.
- Incrémentation de un k.
- Ajouter à la liste B la liste b

Retourner toutes les sous-listes de B sauf la dernière.

Complexité de l'Algorithme 4 : L'ensemble B

Entrées :
Un entier : n
Une liste de couple : E

Initialisation :
$O(n^2)$
O(1)
O(n)
O(1)
O(1)
O(1)

Traitement :
O(n), la boucle s'exécute au plus n-1 fois
- O(1)
- O(1)
- O(1)
- O(1)
- O(c)=O(n)
  - O(c)=O(n)
    - O(1)
    - O(1)
    - $O(n^3)$
    - Si e_c_p est égale à Aucun : O(1)
      - Sinon :
        - $O(n^2)$
        - Si verif est égale à Aucun : O(1)
          - Sinon :
            - O(longeur(e_c_p))=O(n)
  - O(longeur(b_{bis}))=$O(n^3)$ (car on ajoute e_c_p au maximum $O(n^2)$):
    - O(longeur(b))=$O(n^2)$ déterminer dans la partie 4.3:
      - O(1)
  - O(1)
  - O(1)
  - O(1)

O(1)

La complexité de l'algortihme est un $O(n^6)$

6.2 Déduction d'un algorithme de complexité polynomiale pour trouver l'ensemble des triplets qui vérifient (1)

Pour trouver l'ensemble des triplets distincts qui vérifient l'équation (1) on peut en déduire l'algortihme que l'on nomera Triplet suivant :

Algorithme 5 : Triplet

Entrées :
n un entier

Initialisation :
une liste vide appelée "triplets"

Traitement :

```
Pour chaque entier "x" allant de 1 à n :
    | Pour chaque entier "y" allant de 1 à n-x :
    |     | Calculer z=n-x-y
    |     | Si z strictement positif :
    |     |     | Crée un "triplet" en ordre croissant (x,y,z)
    |     |     | Si le "triplet" n'est pas déjà dans la liste triplets :
    |     |     |     | Ajouter le "triplet" à la liste "triplets"
Retourner la liste "triplets"
```

Complexité de l'Algorithme 5 : Triplet

Entrées :
n un entier

Initialisation :
O(1)

Traitement :

```
O(n):
    | O(n) :
    |     | O(1)
    |     | O(1) :
    |     |     | O(1)
    |     |     | O((n-1)(n-2)/2)=O(n^2) (calculer dans la partie 4.1) :
    |     |     |     | O(1)
O(n^2)
```

La complexité de l'algorithme est un $O(n^4)$

6.3 Déduction d'un algorithme de complexité polynomiale pour une savoir si un graphe G(V,E) est 3-coloriable

Algorithme 6 : Trois_coloration

Entrées :
un entier n
une liste E

Initialisation :
Définir l'ensemble V comme une liste d'entiers de 1 à n.
Appel de la fonction B avec les paramètres n et E et stockage du résultat dans E_B.
Appel de la fonction Triplet avec le paramètre n et stockage du résultat dans T.
Pour chaque triplet t dans la liste T :

Traitement :
Pour chaque ensemble β_x dans la liste E_B correspondant
à l'élément du premier élement de t :
    Pour chaque ensemble β_y dans la liste E_B correspondant
    à l'élément du second élement de t :
        Pour chaque ensemble β_z dans la liste E_B correspondant
        à l'élément du tierse élement de t :
            Si la concaténation triée de β_x, β_yet β_z est égale
            à l'ensemble V :
                Retourner le triplet $(\beta_x, \beta_y, \beta_z)$.
Retourner Aucun.

Complexité de l'Algorithme 6 : Trois_coloration

Entrées :
un entier n
une liste E

Initialisation :
O(n)
Complexité dépend de la fonction B(n, E)=O(n^6)
Complexité dépend de la fonction Triplet(n)=O(n^4)

Traitement :
O(len(T))= O(n^2) :
    O(n) car on prend une liste de b_k:
        O(n) car on prend une liste de b_k
            O(n) car on prend une liste de b_k
                O(n^2) il existe de nombre methode pour concaténer
                et trier une liste qui on une complexité en O(n^2).
                    O(1)
O(1)

La complexité de l'algorithme est un O(n^7)

On vient donc d'exiber un algorithme polynomiale qui permet de savoir si un graphe est trois coloriable.

7 Conclusion

En conclusion, la démonstration présentée souligne de manière convaincante que le problème de 3-coloration est résoluble en temps polynomial. Ce résultat a des implications significatives, notamment la résolubilité en temps polynomial de tous les problèmes NP-complets. En se basant sur le travail de Clarence Kineider (https://perso.eleves.ens-rennes.fr/people/clarence.kineider/DevsInfo/3Col.pdf), qui a démontré que le problème de coloration est NP-complet, et en appliquant le théorème de Cook, selon lequel la résolubilité en temps polynomial de tous les problèmes NP-complets équivaut à P=NP, on peut conclure que la résolution efficace du problème de 3-coloration a des répercussions importantes sur la classe de complexité P=NP. Cette démonstration offre une perspective optimiste sur la possibilité de résoudre des problèmes difficiles en temps polynomial, ce qui aurait des implications majeures dans le domaine de la complexité algorithmique.