Exercice n° 1

1. Combien existe-t-il de cycles différents passant par toutes les villes une et une seule
fois ?

Étant donné un graphe complet G, il existe (n)! cycles hamiltoniens possibles (ou (n-1)! si on considère que le point de départ est sans importance.

Remarque :

En utilisant la programmation dynamique, on peut résoudre le problème en (O ($n^2 2^n$) '(algorithme de Held – Karp)

Algorithm 2 : Dynamic Programming algorithm for the original TSP

Data: A set of locations V, an arbitrary location $v \in V$ and cost function c
Result: A shortest tour that visits all locations in V

```
1  Initialize D_TSP with values ∞ ;
2  Initialize a table P to retain predecessor locations ;
3  Initialize v as an arbitrary location in V ;
4  foreach w ∈ V do
5      D_TSP({w}, w) ← c(v, w) ;
6      P({w}, w) ← v ;
7  for i = 2, ..., |V| do
8      for S ⊆ V where |S| = i do
9          foreach w ∈ S do
10             foreach u ∈ S do
11                 z ← D_TSP(S \ {w}, u) + c(u, w) ;
12                 if z < D_TSP(S, w) then
13                     D_TSP(S, w) ← z ;
14                     P(S, w) ← u ;
15 return path obtained by backtracking over locations in P starting at P(V, v) ;
```

Donc l'algorithme peut être exponentiel.

2. Montrer que TSP ∈ NP.

Pour cela, on montre qu'il existe un algorithme non déterministe qui résoud TSP

Algo non déterministe :

Génération :

On génère toute les instances possibles (= tous les cycles = tous les chemin qui passent par tous les sommets)

Vérification :

La vérification d'un chemin se fait en temps linéaire : on parcourt le chemin en marquant chaque sommet qu'on rencontre et en sommant les poids. On vérifie à la fois que le poids total ne dépasse par k et que chaque sommet est visité une et une seule fois.

3. Á l'aide d'une réduction depuis le problème HAMILTONIEN , montrer que TSP est NP-dur, c'est-à-dire que tout les problèmes de NP se réduisent à lui.

Méthode générale

Pour faire une réduction depuis le problème hamiltonien, on procède de la manière suivante :
On montre qu'à partir de chaque instance G du problème Hamiltonien, on peut construire de manière polynomiale une instance (G',w, k) du problème TSP telle que G est hamiltonien si et seulement si (G',w, k) à une réponse positive.

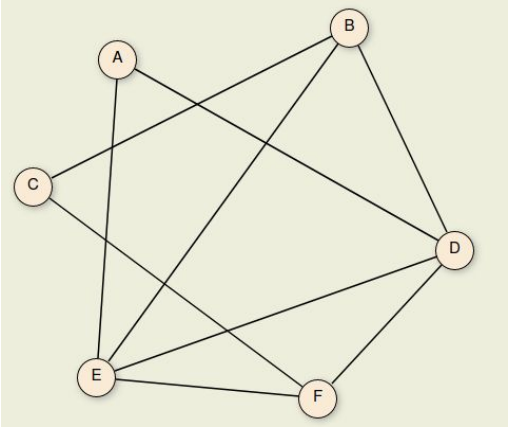
Construction proposée

Étant donné une instance G du problème Hamiltonien, on considère l'instance (G' , w, k) de TSP avec :

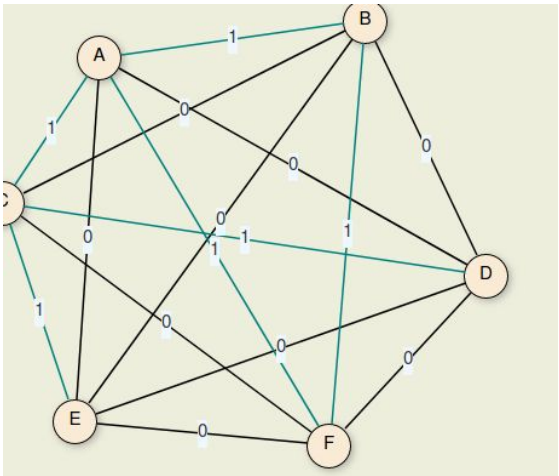
- G' le complété de G,
- w ((u, v)) = 0 si (u, v) ∈ G, et w ((u, v)) = 1 sinon.
- On choisit enfin k = 0

On se convainc facilement que la construction de l'instance (G' , w, k) se fait bien en temps polynomiale à partir de l'instance G

Exemple :
Instance G



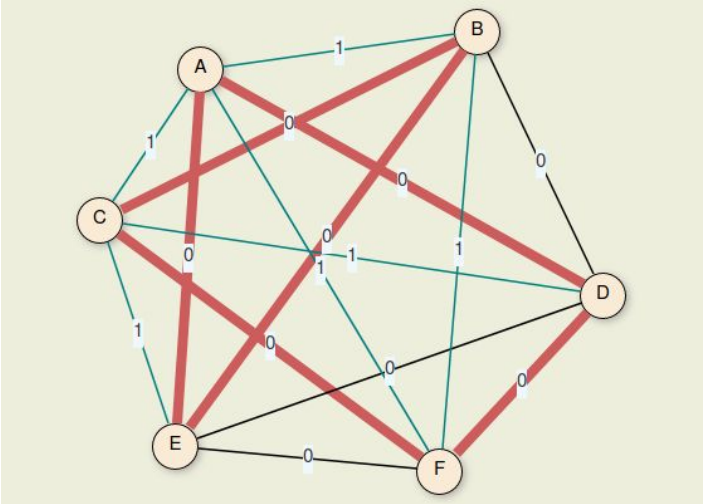
Instance (G', w, k)



Démonstration de l'équivalence :
On montre que G est un graphe hamiltonien si et seulement si (G,'w) vérifie TSP avec un poid de 0:

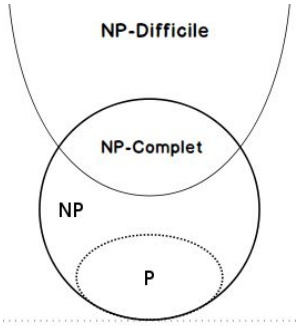
En effet :

1. si G est hamiltonien, alors on prend le même cycle dans G' et le poid vaut 0 donc on vérifie TSP avec k = 0
2. si G' vérifie TSP avec 0, alors on prend le même cycle dans G et alors G est bien hamiltonien (les arêtes prises dans G' sont bien présente dans G).



Conclusion : on a fait une réduction : Hamiltonien se réduit à TSP

or Hamiltonien est NP complet
et donc TSP est NP - dur



$P \neq NP$

Complément : On a montré que TSP appartient à NP et qu'il est NP dur. Donc on peut en déduire qu'il est NP complet.

Exercice n°2

Rappels :

Les formules de la logique propositionnelle sont construites à partir de variables propositionnelles et des connecteurs booléens "et" ($\wedge$), "ou" ($\vee$, "non" ($\neg$). Une formule est **satisfaisable** (on dit aussi **satisfiable**) s'il existe une assignation des variables propositionnelles qui rend la formule logiquement vraie. Par exemple :

- La formule $(p \wedge q) \vee \neg p$ est satisfaisable car si p prend la valeur faux, la formule est évaluée à vrai ;
- La formule $(p \wedge \neg p)\}$ n'est pas satisfaisable car aucune valeur de p ne peut rendre la formule vraie.

Un littéral ℓ est une variable propositionnelle v (littéral positif) ou la négation d'une variable propositionnelle $\neg v$

Clause Normale Conjonctive
$F = (v_1 \vee v_2) \wedge (\neg v_1 \vee v_3) \wedge (\neg v_2 \vee \neg v_1)$

$(v_1 \vee v_2)$, $(\neg v_1 \vee v_3)$ et $(\neg v, \vee \neg v,)$ sont des clauses avec deux littéraux par clause. Leur conjonction f est une forme normale conjonctive.

On s'intéresse à la réduction polynomial de 2-SAT vers la recherche de chemin dans un graphe orienté. On rappelle que 2-SAT est la satisfiabilité d'une Forme Normale Conjonctive F comportant au maximum 2 littéraux par clause. On considère la transformation suivante d'une instance F de 2-SAT en un graphe orienté G appelé le graphe d'implication de F.

- **Pour chaque variable propositionnelle** x_i **, G possède deux sommets étiquetés** x_i **et** $\overline{x_i}$ **.**
- **Pour chaque clause** $l_i \vee l_j$ **, on crée une arête du sommet** $\overline{l_i}$ **vers** l_j **("si** l_i **est faux alors** l_j **doit être vrai") et une arête du sommet** $\overline{l_j}$ **vers** l_i **("si** l_j **est faux alors** l_i **doit être ,vrai").**

L'idée est de remarquer qu'une clause de taille 2 peut toujours s'écrire comme une implication logique. Par exemple la clause $(x_1 \vee x_2)$ dans une formule peut s'écrire $(\neg x_1 \rightarrow x_2)$ ou encore $(\neg x_2 \rightarrow x_1)$
C'est pourquoi on met un arc du sommet $\neg x_1$ au sommet x_2 et un arc du sommet $\neg x_2$ au sommet x_1

1. Dessinez le graphe d'implication correspondant à la formule suivante :
F = ($\overline{x_1}$ ∨ x_2) ∧ ($\overline{x_2}$ ∨ x_3) ∧ ($\overline{x_3}$ ∨ x_1) .

Donnez l'ordre de grandeur de la complexité de la construction du graphe d'implications en fonction du nombre n de variables et du nombre m de clauses de l'instance
2-SAT à transformer.
2n + 2m

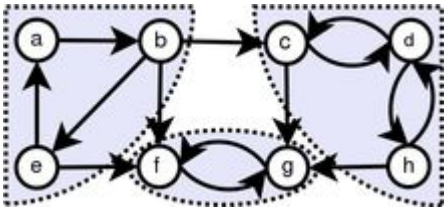
3. Soit F une instance de 2-SAT et G le graphe d'implications correspondant. Montrez que, s'il existe dans G un circuit passant par deux sommets x_i et $\overline{x_i}$, alors F est insatisfiable.
Indication : Montrer (par double implication) que les littéraux reliés par une chaîne fermée d'implications (un circuit du graphe G) ne peuvent qu'avoir la même valeur de vérité (Vrai ou Faux), dans une interprétation satisfaisant F.

Supposons qu'il existe un chemin de x_i vers $\overline{x_i}$ et de $\overline{x_i}$ vers x_i dans G).
Supposons qu'on pose T(x_i) = vrai, alors on obtient T($\overline{x_i}$) = faux.
Puisqu'il existe un chemin de x_i vers $\overline{x_i}$ alors il existe un arc (x_n , x_m) dans ce chemin tel que T(x_n) = vrai et T(x_m) = faux.
Or, l'arc (x_n , x_m) correspond à la clause ($\overline{x_n}$ V x_m) dans la formule F.
Par conséquent, T($\overline{x_n}$,V x_m) = faux et l'assignation T ne satisfait pas la formule F.

On applique le même raisonnement si T(x_i) = faux en considérant le chemin de $\overline{x_i}$ vers x_i .
On ne peut donc ni choisir T(x_i) = vrai ni T(x_i) = faux donc il n'y a pas d'assignation satisfaisante possible.

4. Donner un algorithme qui permet de décider si deux sommets donnés font partie d'un même circuit.

On peut utiliser l'algorithme de Tarjan qui permet de déterminer les composantes fortement connexes d'un graphe orienté



Décomposition d'un graphe en composantes fortement connexes.

Il est en temps linéaire.

L'algorithme prend en entrée un graphe orienté et renvoie une partition des sommets du graphe correspondant à ses composantes fortement connexes.

Le principe de l'algorithme est le suivant : on lance un parcours en profondeur depuis un sommet arbitraire. Les sommets explorés sont placés sur une pile P. Un marquage spécifique permet de distinguer certains sommets : les racines des composantes fortement connexes, c'est-à-dire les premiers sommets explorés de chaque composante (ces racines dépendent de l'ordre dans lequel on fait le parcours, elles ne sont pas fixées de façon absolue sur le graphe). Lorsqu'on termine l'exploration d'un sommet racine v, on retire de la pile tous les sommets jusqu'à v inclus. L'ensemble des sommets retirés forme une composante fortement connexe du graphe. S'il reste des sommets non atteints à la fin du parcours, on recommence à partir de l'un d'entre eux.

Les sommets sont numérotés dans l'ordre où ils sont explorés. Le numéro d'un sommet est noté v.num.

Les arêtes empruntées par le parcours en profondeur forment un arbre. Dans ce contexte, on peut définir le sous-arbre associé à tout sommet v. Au cours de l'exploration de ce sous-arbre, on calcule une seconde valeur v.num Accessible. Elle est initialisée à v.num et décroît lors du parcours des successeurs de v. Lorsque le parcours de v se termine, v.num Accessible correspond au numéro du plus petit sommet situé soit dans le sous-arbre de v, soit successeur direct appartenant à P d'un sommet de ce sous-arbre.

Deux cas sont possibles :

- *v est une racine. Alors tous les sommets accessibles depuis v sont dans le sous-arbre de v (à l'exception éventuelle de ceux explorés lors d'un parcours antérieur). On a v.numAccessible = v.num ;*
- *v n'est pas une racine. Alors il existe un sommet accessible depuis v qui est dans P mais pas dans le sous-arbre de v. On a alors v.num Accessible < v.num.*

5. En déduire que 2-SAT appartient à P

Solution 1 : (sans l'algo de Tarjant) :

On suppose qu'il n'existe pas de **circuit passant par deux sommets** x_i **et** $\overline{x_i}$ et on montre que la formule F est satisfiable.
En effet, on peut construire une assignation T satisfaisante pour la formule F comme ceci :
On répète la procédure suivante jusqu'à ce qu'une valeur soit assignée à tous les sommets :

1. Choisir un sommet x_i **non assigné tel qu'il n'existe pas de chemin de** x_i **vers** $\overline{x_i}$
2. Assigner vrai aux sommets atteignables à partir de x_i
3. Assigner faux à tous les sommets qui atteignent $\overline{x_i}$

Remarque pour la procédure :

- Des valeurs sont toujours assignées à x_i et à $\overline{x_i}$ dans une même itération.
- Supposons qu'il y a des chemins de x_i vers x_j et de x_i vers $\overline{x_j}$ alors par la symétrie du graphe G, il y aurait des chemins de x_j vers $\overline{x_i}$ et de $\overline{x_j}$ vers $\overline{x_i}$ et donc un chemin de x_i **vers** $\overline{x_i}$ vers li, ce qui ne peut pas se produire car on a supposé qu'il n'y a pas de tel chemin dans le graphe.
- À la fin, lorsque tous les sommets sont assignés, on ne trouve pas de chemin allant de vrai vers faux car tous les successeurs d'un sommet assigné à vrai sont à vrai et leur complément à faux.

Alors, la procédure de construction d'une assignation T se termine puisque à chaque itération, au moins une variable est assignée.
Il reste à vérifier que T satisfait F :
Lorsque vrai est assignée à un littéral, alors vrai est assignée à chaque successeur du sommet correspondant à ce littéral. Par analogie, chaque prédécesseur d'un sommet ayant faux assignée au littéral correspondant, la valeur faux lui est aussi assignée.

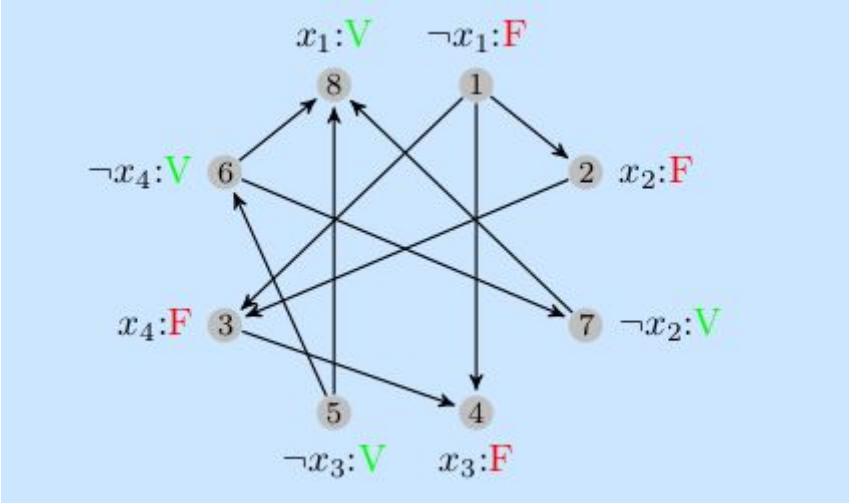
Alors, dans la formule 0, il n'y a pas de clause contenant l'implication (vrai => faux) par l'assignation T :
Si cette implication se trouve dans F alors les assignations x_i =faux et x_j =faux sont données aux littéraux de la clause x_i V x_j et donc les arêtes correspondantes ($\overline{x_i}$, x_j) et ($\overline{x_j}$, x_i) sont des chemins de vrai vers faux. Ceci n'est pas possible à cause des propriétés de T. Donc l'assignation T satisfait F.

Solution 2 : (avec l'algo de Tarjant) :

L'algorithme de Tarjan fournit un ordre topologique sur les composantes fortement connexe (c'est à dire un ordre < tel que s'il existe une arête de S i vers S j alors S i < S j). De plus, une composante fortement connexe Si = {x 1 , . . . , x k } est telle que {¬x 1 , . . . , ¬x k } est aussi une composante fortement connexe, que l'on note ¬S i . On considère alors les composantes dans l'ordre inverse, et on leur affecte des valuations : tant qu'il reste une composante non valuée Si , affecter à tous ses noeuds la valeur VRAI et à tous les noeuds

de ¬S i la valeur FAUX. Ceci assure que tous les noeuds sont finalement valués sans aucun chemin de type
V RAI ⇒ F AUX.
Par exemple :

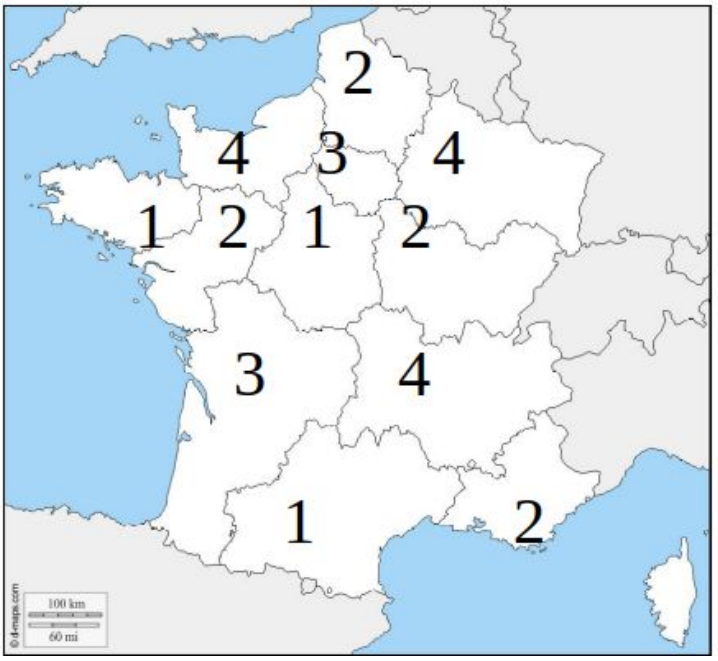
$$\phi = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_4) \wedge (x_1 \vee x_4) \wedge (x_1 \vee x_3) \wedge (\neg x_2 \vee x_4)$$



Le graphe associé est celui de la figure 1 dans lequel chaque sommet une composante fortement connexe, les numéros à l'intérieur des noeuds indiquent un ordre topologique sur les composantes trouvées par l'algorithme de Tarjan. Comme il n'existe aucune composante fortement connexe contenant x i et ¬x i , la formule est satisfiable, et on peut effectuer les valuations, dans l'ordre topologique inverse : VRAI pour x 1 , donc FAUX pour ¬x 1 , vrai pour ¬x 2 , ...

Exercice n° 3

1. Trouver un 4-coloriage de la carte des régions françaises.



2. Montrer que 2-C OULEUR ∈ P.

On montre que 2 couleur est dans P. Pour cela, il faut trouver un algorithme polynomiale qui réponde à la question :

On fait un parcours en largeur en coloriant une "couche" sur deux d'une couleur, et on vérifie si le coloriage fonctionne bien, le tout en temps linéaire.

3. Montrer qu'un graphe G est 2-coloriable si et seulement s'il n'existe pas de cycle de longueur impaire dans G.

Implication :

- Si G possède un cycle impair, il faut au moins trois couleurs pour colorier ce cycle, donc au moins trois couleurs pour G.

Détail :

Supposons que G est 2-colorable, mais qu'il possède un cycle de longueur impaire 2p+1. Soit x_1 x_2 … x_{2p+1} x, ce cycle, avec donc $x_1 = x_{2p+2}$.

Si x_1 a pour couleur C1, alors x_2 doit avoir pour couleur C2, x_3 doit avoir pour couleur C1, etc... Précisément, une récurrence élémentaire prouve que x_k doit avoir la couleur C1 pour k impair et la couleur C2 pour k pair. Il y a un problème pour x_{2p+2}. En effet, il doit avoir la couleur C2 et comme x,= x1, il a déjà la couleur C1!

Réciproque :

On peut supposer que le graphe est connexe : (On peut colorer indépendamment chaque composante connexe d'un graphe.)

On suppose donc que le graphe est connexe, et on fixe x un sommet de G. Pour k≥1, on note G_k l'ensemble des sommets de G qui sont à une distance exactement égale à k de x.

On montre que s'il y a une arête entre un sommet y dans G_{2p} et un sommet z dans $G_{2q,}$ alors le graphe possède un cycle de longueur impaire :

Voici comment construire ce cycle. On prend un chemin de longueur 2p allant de y à x. On lui juxtapose un chemin de longueur 2q allant de x à z. On termine avec l'arête de y à z. On obtient une chaîne de longueur 2p+2q+1.

On peut alors conclure : On vient de prouver que si $y \in G_{2p}$ et z∈G2q, il ne peut pas y avoir d'arête entre y et z. De la même façon, on peut prouver que si $y \in G_{2p+1}$ et $z \in G_{2q+1}$, il ne peut pas y avoir d'arête entre y et z. Ainsi, on peut 2-colorer le graphe de la façon suivante. Si y est un sommet de graphe, alors

- On lui attribue la couleur C1 si $y \in G_{2p}$, pour un certain p∈N
- On lui attribue la couleur C2 si $y \in G_{2p+1}$, pour un certain p∈N.

4. On suppose maintenant que le problème 2-SAT suivant est dans P.
"Etant donnée une formule F en forme normale conjonctive dans laquelle chaque clause a exactement 2 littéraux, F est-elle satisfiable ?"
Remontrer, à l'aide d'une réduction à 2-SAT, que 2-COULEUR est dans P.

On montre que 2-Couleur <= 2-SAT

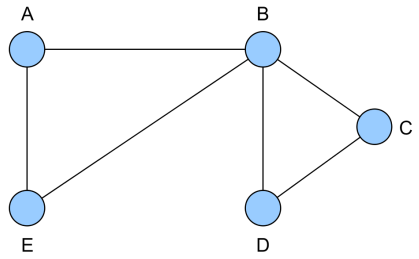
Méthode générale
Pour faire une réduction depuis le problème 2-SAT, on procède de la manière suivante :
On montre qu'à partir de chaque instance I1 du problème 2-Couleur, on peut construire de manière polynomiale une instance I2 du problème 2-SAT telle que I1 est 2 coloriable si et seulement si I2 est satisfiable.

Construction proposée

on part d'une instance de 2-Couleur et on construit une instance de 2 SAT de la manière suivante :

- Pour chaque sommet A du graphe I1 , on lui attribue une variable propositionnelle x_A
- pour chaque arête (A, B) on construit les clauses $(x_A \vee x_B) \wedge (\overline{x_A} \vee \overline{x_B})$
- On obtient alors la formule F en ajoutant toutes les clauses
- Pour chaque couleur d'un sommet A , on assigne une valeur de vérité Vrai / Faux à la variable propositionnelle x_A

La construction est bien polynomiale (nombre de variable, nombre de clause)
Exemple :
Instance I1



Instance I2

F = $(x_A \vee x_B) \wedge (\overline{x_A} \vee \overline{x_B}) \wedge (x_A \vee x_E) \wedge (\overline{x_A} \vee \overline{x_E}) \wedge (x_E \vee x_B) \wedge (\overline{x_E} \vee \overline{x_B})$
$\wedge (x_C \vee x_B) \wedge (\overline{x_C} \vee \overline{x_B}) \wedge (x_D \vee x_B) \wedge (\overline{x_D} \vee \overline{x_B}) \wedge (x_D \vee x_C) \wedge (\overline{x_D} \vee \overline{x_C})$

Démonstration de l'équivalence :

On se convainc facilement que F satisfiable si et seulement si G 2 coloriables (il faut qu'il y ai un Vrai et un Faux dans chaque clause)

Conclusion :
On a montrer que 2-Couleur <= 2-SAT et on sait de plus que -SAT est dans P.
On peut donc en conclure que 2-Couleur est dans P

5. Montrer que 3-Couleur ∈ NP.

On utilise considère l'algo non déterministe polynomial suivant :
1- Génération non déterministe :
On génère toute les colorations possibles avec 3 couleur
2. Vérification polynomiale
on vérifie pour chaque colorations générées, si la coloration est valide en temps polynomiale (parcour de graphe en largeur).

Cette algorithme non déterministe polynomiale permet de répondre à 3-Couleur. On peut donc en conclure que 3-Couleur ∈ NP.

Remarque : On voit ici que le nombre de couleurs a peu d'importance. On montre ainsi de la même manière que pour tout entier k, k-Couleur ∈ NP.

6. Montrer que 3-Couleur est NP-dur.

On montre que 3-SAT < 3 Couleur. Ainsi, comme 3-SAT est NP-complet, on pourra en déduire que 3 Couleur est NP-dur

Construction proposée

On considère une instance de I1 3 SAT et on construit de manière polynomiale une instance I2 de 3 Color telle que I1 est satisfiable si et seulement si I2 est 3-coloriable.

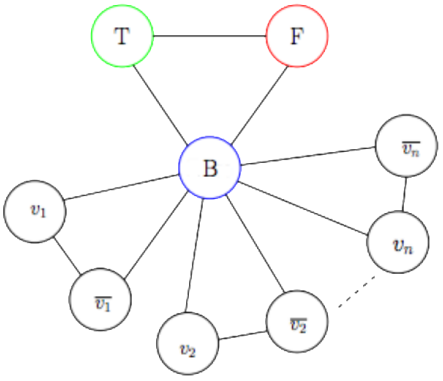
On part donc d'une formule pour construire un graphe avec une affectation de couleur en fonction de valeur de vérité des variables.

Etape 1

- On considère une formule F avec n variables $x_1, x_2, x_3, .. x_n,$ et m clauses C_1, C_2 ...C_m
- On considère {T, F, B} (True, False, Base)comme l'ensemble des 3 couleurs que nous utiliserons pour colorer (étiqueter) les sommets du graphe.

Ainsi, à une 3-coloration correspond une valuation des variables ; et réciproquement, toute valuation peut être représentée par une 3-coloration

- On crée un triangle avec 3 noeud True, False, Base
- Pour chaque variables x_i on crée deux sommets v_i et $\overline{v_i}$ connecté en triangle avec le somme B

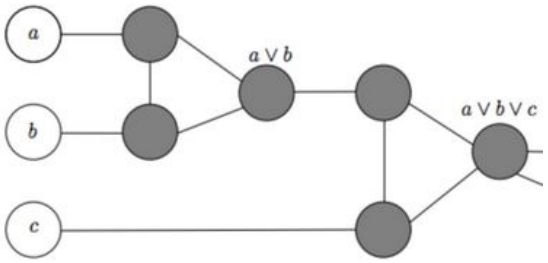


(De cette façon, si on a un 3-coloriage, les sommets correspondants à un littéral auront soit la couleur de T, soit de F, ce qui permettra de leur associer une valeur de vérité.)

Etape 2

Pour chaque clause C = (a∨b∨c), nous devons exprimer le OU de ses littéraux en utilisant nos couleurs {T, F, B}. Pour ce faire, on crée un "gadget "graphique que nous connectons aux littéraux de la clause.

Le gadget OU est construit de la manière suivante

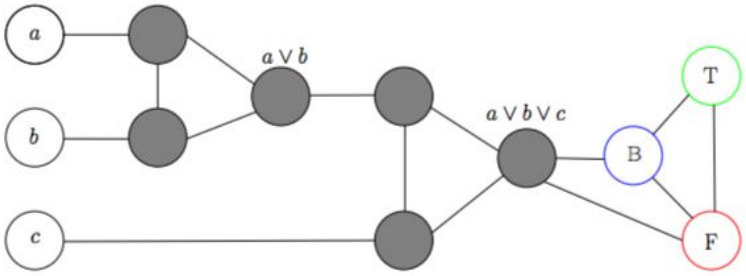


Prop1 : si les noeud a, b et c sont colorés en F, alors la sortie doit être coloré en F dans le cadre d'un 3 coloriage valide)

Prop 2 : si un des noeud a, b, c est coloré en T, alors la sortie doit être coloré en T (dans le cadre d'un 3 coloriage valide)

Etape 3

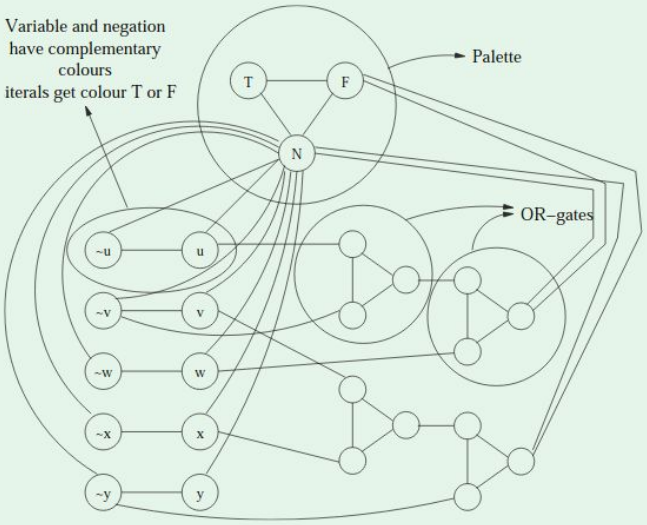
Pour chaque clause C, on connecte la sortie du gadget graphique aux deux noeuds B et T



Cela permet de capturer la satisfiabilité de chacunes des clauses .

Exemple :

$$\varphi = (u \vee \neg v \vee w) \wedge (v \vee x \vee \neg y)$$



Démonstration de l'équivalence :

Maintenant, nous prouvons que notre instance 3-SAT initiale F est satisfiable si et seulement le graphe G construit ci-dessus est 3 coloriable.

F satisfiable implique que G est 3 coloriable :

Supposons F soit satisfaisable. On considère une affectation de valeur de vérité de $x_{1,}$ $x_{2,}$ $x_{3,}$ .. $x_{n,}$ qui réalise F.
Si $x_{i,}$ est affecté True, nous colorions le sommet v_i avec T et $\overline{v_i}$ avec F
Comme F est satisfaisable, chaque clause C = (a∨b∨c) doit être satisfiable, c'est-à-dire au moins a , b, c est défini sur True.
Par la deuxième propriété du gadget OR, nous savons que le gadget correspondant à la clause C a 3 couleurs pour que le nœud de sortie soit coloré en T. Et parce que le nœud de sortie est adjacent aux sommets False et Base du triangle initial, c'est une bonne 3-coloration.

G 3 coloriable implique F satisfiable:
À l'inverse, supposons que G est 3-colorable.
Nous construisons une affectation des littéraux de C en définissant $x_{i,}$ à True si v_i est coloré en T et inversement.

Supposons maintenant que cette affectation ne soit pas satisfaisante pour F, alors cela signifie qu'il existe au moins une clause C= (a∨b∨c) qui n'était pas satisfaisable.
C'est-à-dire, tous que tous les littéraux a, b, c sont à False.
Mais si c'est le cas, le nœud de sortie du gadget OR correspondant de C doit doit être coloré à F.
Mais ce nœud de sortie est adjacent au sommet de couleur F;
Cela contredit ainsi la 3-colorabilité de G

Conclusion
Pour conclure, nous avons montré que 3-Couleur est en NP et qu'il est NP-dur en donnant une réduction de 3-SAT.
Par conséquent, 3-Couleur est NP-complet

7. Montrer que 4-Couleur est NP-Complet.

On montre que 3-Couleur se réduit à 4-Couleur : 3-Couleur <= 4-Couleur
Pour chaque instance G de 3-Couleur, on construit une instance G' de 4-couleurs de la manière suivante :
On prend G' et on ajoute un sommet que l'on relie à tous les autres. (la construction est bien polynomiale).

On a bien l'équivalence : G est 3 coloriable si et seulement si G' est 4 coloriable.
En effet :
Si G est 3 coloriable, on colorie dans G' le nouveau sommet de la 4eme couleur non utilisé et on obtient ainsi un 4 - coloriage valide .

Si G' est 4 coloriable, le sommet relié à tous les autres est d'une couleur que seul lui a. la coloration obtenu pour les autres sommets est un 3-coloriage valide pour G.

Donc on a bien 3-couleur <= 4-couleur

Conclusion : on a
- 3-couleur <= 4-couleur
- 3-couleur est NP complet
- 4-couleur **∈ NP.**

On peut donc en conclure que 4-couleur est NP complet