Correction des exercices de TD

Table des matières

1 Calculabilité

1.1 Divers

Exercice 1 - Paradoxe

Montrer que les problèmes suivants engendrent un paradoxe.

1. Le conseil municipal d'un village vote un arrêté municipal qui enjoint à son barbier (masculin) de raser tous les habitants masculins du village qui ne se rasent pas eux-même et seulement ceux-ci.
2. Un crocodile s'empare d'un bébé et dit à la mère : « si tu devines ce que je vais faire, je te rends le bébé, sinon je le dévore. ». En supposant que le crocodile tienne parole, que doit dire la mère pour que le crocodile rende l'enfant à sa mère ? Une réponse usuelle de la mère est : « Tu vas le dévorer ! »

1. Le barbier, s'il se rase, se rase lui-même, mais il est aussi rasé par le barbier. S'il ne se rase pas, il doit être rasé par le barbier, c'est à dire lui-même, donc il se rase. L'énoncé est faux (impossible).
2. Si le crocodile rend le bébé à la mère avec cette réponse, c'est qu'il comptait le dévorer. Seulement, s'il a l'intention de rendre le bébé, c'est qu'il n'a pas l'intention de le dévorer, donc il le dévorera.

Exercice 2 - Une preuve incorrecte

Nous considérons la fonction suivante donnée par l'algorithme 1 :

Algorithm 1: La fonction de Collatz

```
1 begin
2     while n ≠ 1 do
3         if n mod 2 = 0 then
4             n := n/2
5         else
6             n := 3 × n + 1
```

Actuellement nous ne savons pas si cette fonction termine $\forall n$. Êtes-vous d'accord avec la preuve suivante ? « Si le problème de l'arrêt était décidable, il suffirait de l'appliquer à ce programme pour savoir si son exécution s'arrête. Or, on ne sait pas si son exécution s'arrête. D'où la contradiction. »

Le raisonnement est faux, la preuve est donc incorrecte. En effet, « on ne sait pas si son exécution s'arrête » signifie que soit elle s'arrête, soit elle ne s'arrête pas, on répète la question. Il n'y a pas de lien logique entre la première et la seconde phrase.

1.2 Variations sur le codage

Exercice 3 - Codage de couples d'entiers

Soit $Rang : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ tel que $Rang(x, y) = \frac{(x+y)(x+y+1)}{2} + y$.

1. Donner une version récursive de la fonction $Rang$.
2. Donner la fonction inverse.
3. Calculer $Rang(4, 5)$. Donner le couple pour lequel la valeur du codage est 8.

1. $RangRec = \begin{cases} 0 & \text{si } x = 0 \text{ et } y = 0 \\ RangRec(0, x-1) + 1 & \text{si } y = 0 \\ RangRec(x+1, y-1) + 1 & \text{sinon} \end{cases}$.
2. On pose la fonction inverse $RangInv(n) : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$. On cherche d'abord $x + y$, on prend donc $x + y = max\{m \mid \frac{m(m+1)}{2} \leq n\}$. On pose $t = x + y$. Ainsi, comme $n = \frac{t(t+1)}{2} + y$, on a $y = n - \frac{t(t+1)}{2}$. De plus, comme $t = x + y$, pour retrouver x, il suffit de prendre $x = t - y$.
3. $Rang(4, 5) = \frac{(4+5)(4+5+1)}{2} + 5 = \frac{9 \times 10}{2} + 5 = 50$. Pour $n = 8$, on cherche d'abord t. On a $t = max\{1, 2, 3\} = 3$, et $\frac{t(t+1)}{2} = 6$. On a donc $y = 8 - 6 = 2$ et $x = 3 - 2 = 1$. Le couple codé par $n = 8$ est $(1, 2)$.

Exercice 4 - Codage de triplets

Soit c la fonction de codage pour les couples d'entiers vue dans l'exercice précédent.

1. Soit h la fonction de codage pour les triplets définie par $h(x, y, z) = c(c(x, y), z)$. Quel est le doublet codé par 67 ? Quel est le triplet codé par 67 ?
2. Le couple (z, t) succède au couple (x, y) si $c(z, t) = c(x, y) + 1$. Écrire la fonction successeur qui prend en paramètre un couple et retourne le couple successeur.

1. Pour $n = 67$, on cherche d'abord t. On a $t = max\{m|m \leq 11\} = 11$. En effet, $\frac{t(t+1)}{2} = \frac{11\times12}{2} = \frac{132}{2} = 66$. On a donc $y = 67 - 66 = 1$ et $x = 11 - 1 = 10$. Comme 67 code $(10, 1)$, pour avoir le triplet, on veut le couple codé par $n = 10$. De nouveau, on cherche $t = max\{1, 2, 3, 4\} = 4$. En effet, $\frac{4\times5}{2} = 10$. Donc $y = 10 - 10 = 0$ et $x = 4 - 0 = 4$. Le couple codé par $n = 10$ est $(4, 0)$ et ainsi le triplet codé par $n = 67$ est $(4, 0, 1)$.
2. La fonction c fait augmenter les couples comme une diagonale. Prenons les premiers couples :

$c(x, y)$	0	1	2	3	4	5	6	7	8	9
(x, y)	$(0, 0)$	$(1, 0)$	$(0, 1)$	$(2, 0)$	$(1, 1)$	$(0, 2)$	$(3, 0)$	$(2, 1)$	$(1, 2)$	$(0, 3)$

On remarque que le successeur de (x, y) est $(x - 1, y + 1)$ (qui est d'ailleurs l'inverse de ce qu'on avait fait à l'exercice précédent pour $RangRec$), et que si $x = 0$, le successeur est $(y + 1, 0)$. Cela nous donne la fonction suivante : $sucesseur(x, y) = \begin{cases} (y + 1, 0) & \text{si } x = 0 \\ (x - 1, y + 1) & \text{sinon} \end{cases}$

Exercice 5 - Étude d'une équation fonctionnelle dans $\mathbb{N}$

Soit f une application de $\mathbb{N}$ dans $\mathbb{N}$ telle que : $\forall(m, n) \in \mathbb{N}, f(m^2+n^2) = f(m)^2+f(n)^2$. Nous voulons montrer que f est :

— l'application nulle, donnée par : $\forall n \in \mathbb{N}, f(n) = 0$,

— l'application identité, donnée par : $\forall n \in \mathbb{N}, f(n) = n$.

Nous supposerons que a est l'entier naturel $f(1)$.

1. Montrer que $f(0) = 0$. En déduire que $\forall n \in \mathbb{N}$, on a $f(n^2) = f(n)^2$.
2. Montrer alors que $a^2 = a$, donc que a est égal à 0 ou à 1.
3. Vérifier successivement les égalités $f(2) = 2a$, $f(4) = 4a$ et $f(5) = 5a$.
4. Utiliser les valeurs $f(4)$ et $f(5)$ pour montrer que $f(3) = 3a$.
5. Utiliser les valeurs de $f(1)$ et de $f(5)$ pour montrer que $f(7) = 7a$.
6. Montrer que $f(8) = 8a$, $f(9) = 9a$, $f(10) = 10a$ et $f(6) = 6a$.
7. Observer que

$$\forall m, \text{ on a } \begin{cases} (2k)^2 + (k - 5)^2 = (2k - 4)^2 + (k + 3)^2 \\ (2k + 1)^2 + (k - 2)^2 = (2k - 1)^2 + (k + 2)^2 \end{cases}$$

Montrer que $\forall n$, on a $f(n) = an$.

8. Conclure.

1. On a $m^2 + n^2 = 0 \Rightarrow m = 0$ et $n = 0$:

$$\begin{aligned} & f(m^2 + n^2) = f(m)^2 + f(n)^2 \\ \Rightarrow\ & f(0) = f(0)^2 + f(0)^2 \\ \Rightarrow\ & f(0) = 2f(0)^2 \\ \Rightarrow\ & f(0) - 2f(0)^2 = 0 \\ \Rightarrow\ & f(0)(1 - 2f(0)) = 0 \end{aligned}$$

Il y a deux cas. Soit $f(0) = 0$, soit $1 - 2f(0) = 0 \Rightarrow f(0) = \frac{1}{2}$. Or, f est une fonction de $\mathbb{N}$ dans $\mathbb{N}$, donc le seul résultat possible ici est $f(0) = 0$. On en déduit que $f(n^2) = f(0^2 + n^2) = f(0)^2 + f(n)^2 = f(n)^2$.

2. On sait que $f(n^2) = f(n)^2$. De plus, $1^2 = 1$, donc $f(1) = f(1)^2 \Rightarrow a = a^2$. Dans les entiers, seulement 0 et 1 vérifient cette égalité.
3. $f(2) = f(1^2 + 1^2) = f(1)^2 + f(1)^2 = a^2 + a^2 = 2a^2 = 2a$.
   $f(4) = f(0^2 + 2^2) = f(0)^2 + f(2)^2 = (2a)^2 = 4a^2 = 4a$.
   $f(5) = f(1^2 + 2^2) = f(1)^2 + f(2)^2 = a^2 + 4a^2 = 5a^2 = 5a$.
4. $f(5^2) = f(25) = f(3^2 + 4^2) = f(3)^2 + f(4)^2 = f(3)^2 + 16a^2$. De plus, $f(5^2) = f(5)^2 = (5a)^2 = 25a^2$. On a donc $f(3)^2 = 25a^2 - 16a^2 = 9a^2$. Ainsi, $f(3) = \sqrt{9a^2} = 3a$.
5. $f(5^2+5^2) = f(1^2+7^2) = f(50)$. On a $f(50) = f(5)^2+f(5)^2 = 50a^2$ et $f(50) = f(1)^2+f(7)^2 = a^2+f(7)^2$. On a donc $f(7)^2 = 50a^2 - a^2 = 49a^2$. Ainsi, $f(7) = \sqrt{49a^2} = 7a$.
6. $f(8) = f(2^2 + 2^2) = f(2)^2 + f(2)^2 = 4a^2 + 4a^2 = 8a^2 = 8a$.
   $f(9) = f(3^2 + 0^2) = f(3)^2 + f(0)^2 = 9a^2 = 9a$.
   $f(10) = f(3^2 + 1^2) = f(3)^2 + f(1)^2 = 9a^2 + a^2 = 10a^2 = 10a$.
   $f(10^2) = f(8^2 + 6^2) = f(100)$. On a $f(100) = f(10^2) = f(10)^2 = 100a^2$ et $f(100) = f(8^2 + 6^2) = f(8)^2 + f(6)^2 = 64a^2 + f(6)^2$. On a donc $f(6)^2 = 100a^2 - 64a^2 = 36a^2$. Ainsi, $f(6) = \sqrt{36a^2} = 6a$.
7. On veut prouver l'hypothèse $H(n) : \forall n, f(n) = an$. On prouve ça par induction :

   Base On a prouvé précédemment tous les cas pour $n \leq 10$.

   Induction On suppose $\forall i < n, H(i)$. Montrons $H(n)$. Il y a deux cas : n pair, c'est à dire qu'il existe k tel que $n = 2k$ ou bien n impair, c'est à dire qu'il existe k tel que $n = 2k + 1$.

   — n est pair. On sait que $f((2k)^2 + (k-5)^2) = f((2k-4)^2 + (k+3)^2)$:

$$
\begin{aligned}
& f((2k)^2 + (k-5)^2) = f((2k-4)^2 + (k+3)^2) \\
& \Rightarrow f(2k)^2 + f(k-5)^2 = f(2k-4)^2 + f(k+3)^2 \\
& \Rightarrow f(2k)^2 = f(2k-4)^2 + f(k+3)^2 - f(k-5)^2 \\
& \Rightarrow f(2k)^2 = a^2(2k-4)^2 + a^2(k+3)^2 - a^2(k-5)^2 \\
& \Rightarrow f(2k)^2 = a^2(4k^2 - 16k + 16) + a^2(k^2 + 6k + 9) - a^2(k^2 - 10k + 25) \\
& \Rightarrow f(2k)^2 = a^2(4k^2) \\
& \Rightarrow f(2k) = \sqrt{4a^2k^2} \\
& \Rightarrow f(2k) = a2k
\end{aligned}
$$

   — n est impair. On sait que $f((2k+1)^2 + (k-2)^2) = f((2k-1)^2 + (k-2)^2)$:

$$
\begin{aligned}
& f((2k+1)^2 + (k-2)^2) = f((2k-1)^2 + (k+2)^2) \\
& \Rightarrow f(2k+1)^2 + f(k-2)^2 = f(2k-1)^2 + f(k+2)^2 \\
& \Rightarrow f(2k+1)^2 = f(2k-1)^2 + f(k+2)^2 - f(k-2) \\
& \Rightarrow f(2k+1)^2 = a^2(2k-1)^2 + a^2(k+2)^2 - a^2(k-2)^2 \\
& \Rightarrow f(2k+1)^2 = a^2(4k^2 - 4k + 1) + a^2(k^2 + 4k + 4) - a^2(k^2 - 4k + 4) \\
& \Rightarrow f(2k+1)^2 = a^2(4k^2 + 4k + 1) \\
& \Rightarrow f(2k+1) = \sqrt{a^2(4k^2 + 4k + 1)} \\
& \Rightarrow f(2k+1) = a(2k+1)
\end{aligned}
$$

   Conclusion On a prouvé que $\forall i \leq 10, H(i)$ et $\forall i < n, H(i) \Rightarrow H(n)$, on a donc bien $\forall n, f(n) = an$.
8. On a prouvé que $\forall n, f(n) = an$. De plus, on sait que $a = 0$ ou $a = 1$. Ainsi, il y a deux cas :
   — $\forall n, f(n) = 0 \times n = 0$ (application nulle),
   — $\forall n, f(n) = a \times n = n$ (application identité).

   On a prouvé que les deux seules applications de $\mathbb{N} \to \mathbb{N}$ telles que $\forall (m, n) \in \mathbb{N}, f(m^2 + n^2) = f(m)^2 + f(n)^2$ sont l'application nulle et l'application identité.

Exercice 6 - Codage rationnels

Proposer un codage pour les nombres rationnels.

On peut proposer un codage naïf : toute fraction rationnelle $\frac{a}{b}$ se réduit en fraction $\frac{p}{q}$ avec p, q premiers. Pour coder les rationnels, on pourrait prendre $c(p, q)$. Cependant, ce codage est très inefficace, car il y a énormément de couples non premiers entre eux.

Une méthode moins naïve de faire serait de poser la fonction σ qui prend $(p + q)$ et ordonne par p (numérateur) croissant lors de l'égalité :

$\sigma(p/q)$	1	2	3	4	5	6
$\frac{p}{q}$	$\frac{0}{1}$	$\frac{1}{1}$	$\frac{1}{2}$	$\frac{2}{1}$	$\frac{1}{3}$	$\frac{3}{1}$

On remarque qu'on saute $\frac{2}{2}$, car on peut réduire la fraction a $\frac{1}{1}$.

Exercice 7 - Codage des listes d'entiers

Pour coder les listes d'entiers, peut-on :

1. Faire la somme des entiers de la liste, et à somme égale prendre l'ordre lexicographique ?
2. Faire comme pour les mots : prendre les listes les plus courtes d'abord et à égalité de longueur l'ordre lexicographique ?

1. Non, car dans ce cas, on n'aurait que les listes qui contiennent des 0 : $(0), (0, 0), (0, 0, 0), \ldots$.
2. Non, car dans ce cas, on n'aurait que les listes de longueur 1 : $(0), (1), (2), \ldots$.

Exercice 8 - Codage de listes d'entiers

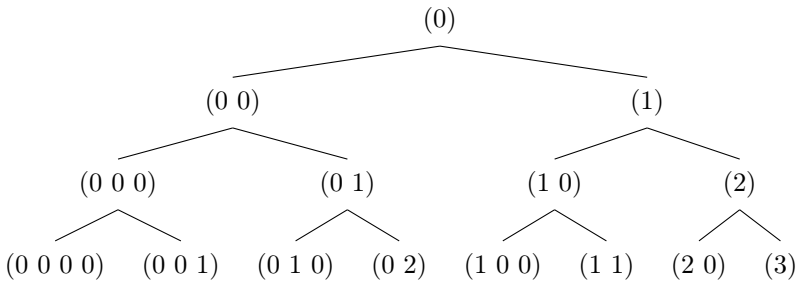
On ordonne les listes de la façon suivante :

$$\sigma(l) = \text{somme des entiers de la liste} + \text{longueur de la liste}$$

Puis à valeur de σ égale on ordonne dans l'ordre lexicographique.
On note U_k l'ensemble des listes l telles que $\sigma(l) = k$ et $u_k = |U_k|$.

1. Donner les ensemble $U_i, i = 0, \ldots, 4$.
2. Montrer que $u_k = 2^{k-1}, \forall k \geq 1$.
3. Quelle est la première liste de $U_k, \forall k \in \mathbb{N}^*$ et la dernière ?
4. Donner la fonction de codage en version itérative et récursive (resp. décodage).

1. $U_0 = \{()\}$ la liste vide.
   $U_1 = \{(0)\}$.
   $U_2 = \{(0, 0), (1)\}$.
   $U_3 = \{(0, 0, 0), (1, 0), (0, 1), (2)\}$.
   $U_4 = \{(0, 0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1), (2, 0), (1, 1), (0, 2), (3)\}$.
2. Pour tout $k \geq 1$, on peut exhiber la méthode de construction suivante : pour chaque élément l de U_{k-1}, le fils gauche est l à laquelle est ajoutée 0 à la fin, et le fils droit est l en ajoutant 1 à son dernier élément :



Arbre de construction des listes d'entiers

On peut ainsi faire une preuve par induction :

Base On a prouvé le cas pour $k \leq 4$.

Induction On suppose $\forall i \leq n$, $u_i = 2^{i-1}$. Prouvons le pour u_{n+1}. Comme $u_n = 2^{n-1}$ et que la construction est inductive sous forme d'arbre, il y a 2 fils pour chaque élément de U_n donc la taille est $2 \times u_n = 2 \times 2^{n-1} = 2^n$.

Conclusion On a montré la base, et pour tout $k \geq 1$, on a bien $u_k = 2^{k-1} \Rightarrow u_{k+1} = 2^k$.

3. La première liste de U_k est la liste $(0, \ldots, 0)$ composée de k 0, et la dernière est le singleton $(k-1)$.
4. On peut donner la version récursive suivante, qui se déduit pratiquement immédiatement de l'arbre de construction des listes :

Algorithm 2: Codage d'une liste d'entier

```
1 begin
2     if n = 1 et e_0 = 0 then
3         return 1
4     if e_{n-1} = 0 then
5         return 2 × codage(e_0, ..., e_{n-2})
6     return 2 × codage(e_0, ..., e_{n-1} − 1) + 1
```

Le décodage est ainsi intuitif : on prend un compteur, si $n \mod 2 = 0$, on ajoute 0 à la liste, sinon, tant que $n \mod 2 = 1$, on ajoute 1 au compteur :

Algorithm 3: Décodage en liste d'entier

```
1 begin
2     if n = 1 then
3         return (0)
4     if n mod 2 = 0 then
5         return cons(decodage(n/2), 0)
6     L := decodage(⌊n/2⌋);
7     L_[−1] := L_[−1] + 1;
8     return L
```

$L_{[-1]}$ correspond au dernier élément de la liste.

Exercice 9 - Codage d'entiers

Soit la fonction f suivante de $\mathbb{N}^* \to \mathbb{N}$:

$$f(n) = k \text{ si } n = 2^k$$
$$f(n) = f(n/2) \text{ si } n \text{ est pair et n'est pas une puissance de 2}$$
$$f(n) = f(3n+1) \text{ sinon}$$

Nous appelons $A_i = \{x \mid f(x) = i\}$.

1. Donner quelques éléments de A_i, $\forall i \in \{1, 2, 3, 4, 5, 6\}$.
2. Donner un algorithme qui prend i en paramètre et qui affiche tous les éléments de A_i.
3. Donner un algorithme qui affiche $A_1 \cup A_2$.
4. Donner un algorithme qui affiche $A_4 \cup A_6$.

1. $A_1 = \{2\}$
   $A_2 = \{4\}$
   $A_3 = \{8\}$
   $A_4 = \{16, 3, 5, 10, 24, 48, 20, 40, \ldots\}$
   $A_5 = \{32, \ldots\}$
   $A_6 = \{64, 21, 42, 84, \ldots\}$
2. Intuitivement, on voudrait faire l'algorithme suivant :

Algorithm 4: Afficher A_i

```
1 begin
2     k := 0;
3     while true do
4         if f(k) == i then
5             afficher(k)
6         k := k + 1
```

Seulement, on ne sait pas si $f(k)$ est calculable. Par exemple, on pourrait avoir $f(10^{38}) = i$, f qui boucle pour $k = 10^{39}$, et $f(10^{40}) = i$. Dans ce cas, 10^{40} ne sera jamais affiché alors qu'il fait bel et bien partie de A_i. Cet algorithme est donc faux.

Ainsi, c'est l'algorithme suivant qui affiche A_i :

Algorithm 5: Afficher A_i

```
1  begin
2      afficher(2^i);
3      L := ∅;
4      if (2^i − 1) mod 3 = 0 then
5          L = {(2^i − 1)/3}
6      while L ≠ ∅ do
7          n := tete(L);
8          afficher(n);
9          L := queue(L);
10         L := ajouter(2n, L);
11         if (n−1)/3 est impair then
12             L := ajouter((n − 1)/3, L)
```

3. Comme A_1 et A_2 sont finis ($(2^i - 1) \mod 3 \neq 0$), il suffit d'afficher 2^1 et 2^2 (et 1 si on considère que A_0 n'existe pas).
4. Comme A_4 et A_6 sont infinis, on ne peut pas afficher A_4 puis A_6, car on n'afficherait que l'un ou l'autre, et pas les deux. Ainsi, il faut alterner entre élément de A_4 et élément de A_6.

1.3 Diagonalisation

La diagonalisation est un procédé assez intuitif une fois qu'il est compris. On utilise cette technique pour prouver qu'il n'existe pas d'énumération d'ensembles infinis. La procédure est simple : il suffit de poser un tableau qui énumère tous les ensembles e_i possibles, puis de prendre un ensemble contradictoire : celui qui prend la valeur v_i de l'ensemble e_i. Comme notre tableau énumère tous les ensembles possibles, cela veut dire que l'ensemble contradictoire est dans le tableau. Or, si c'est le k-ème, on aura v_k de $e_k = v_k$ et v_k de $e_k \neq v_k$, ce qui amène à une contradiction !

Exercice 10 - Diagonalisation

1. Montrer que l'ensemble des parties d'un ensemble E infini dénombrable n'est pas dénombrable.
2. Que peut-on conclure sur la cardinalité de l'ensemble des fonctions ? Et de l'ensemble des programmes ?
3. Préciser le cas où E est un ensemble fini (donc dénombrable) ?

1. Soit $\mathcal{P}(E)$ l'ensemble des parties de E. Supposons que tout élément de $\mathcal{P}(E)$ est représentable par $(P_i)_{i\in\mathbb{N}}$. Soit $\mathcal{X}_E$ la fonction caractéristique de E, $\mathcal{X}_E : E \to \{0, 1\}$. On peut construire le tableau suivant :

	e_0	e_1	e_2	...	e_i	...
P_0	$\boxed{0}$	1	0	...	1	...
P_1	1	$\boxed{1}$	0	...	1	...
P_2	0	0	$\boxed{0}$	...	0	...
⋮	⋮	⋮	⋮	⋱	⋮	⋮
P_i	1	0	1	...	$\boxed{0}$	...
⋮	⋮	⋮	⋮	⋮	⋮	⋱

On prend l'ensemble $P \in \mathcal{P}(E)$ tel que $P = \{i \mid e_i \notin P_i\}$. On suppose que $\exists i.P = P_i$, car $\forall k \in \mathbb{N}.P_k \in \mathcal{P}(E)$. Si $P = P_i$, alors si $e_i \in P_i$, $e_i \notin P$ par construction, donc $P \neq P_i$. De même, si $e_i \notin P_i$, par construction, $e_i \in P$, donc $P \neq P_i$. On ne peut pas trouver i tel que $P = P_i$, donc le tableau ne contient pas tous les éléments de $\mathcal{P}(E)$, donc l'ensemble des parties d'un ensemble E infini dénombrable n'est pas dénombrable ($|\mathcal{X}_E| > |\mathbb{N}|$).

2. L'ensemble des fonctions caractéristiques est un sous-ensemble de l'ensemble des fonctions : $\mathcal{X} \subset \mathcal{F}$ car $\mathcal{X} : \mathbb{N} \to \{0,1\}$ et $\mathcal{F} : \mathbb{N} \to \mathbb{N}$. Comme $|\mathcal{X}| > \mathbb{N}$ et $|\mathcal{F}| > |\mathcal{X}|$ (car $\mathcal{X} \subset \mathcal{F}$), on a $|\mathcal{F}| > |\mathbb{N}|$, donc l'ensemble des fonctions n'est pas dénombrable.

   L'ensemble des programmes, lui, est dénombrable. On peut par exemple prendre le taille puis l'ordre lexicographique à taille égale. Vu qu'on travaille sur un alphabet fini, cet ensemble sera fini.

3. Si E est fini, $\mathcal{P}(E)$ est dénombrable. En effet, on ne peut pas construire de tableau infini, on ne peut donc pas appliquer la technique de diagonalisation et on conclut pratiquement immédiatement que c'est dénombrable.

Exercice 11 - Diagonalisation

Montrer que l'ensemble des sous-ensembles d'un ensemble dénombrable n'est pas dénombrable. Pour cela, considérer un ensemble dénombrable $A = \{a_0, a_1, a_2, \ldots\}$, et S l'ensemble de ses sous-ensembles.

L'ensemble des sous-ensembles d'un ensemble est l'ensemble des parties d'un ensemble $\mathcal{P}(A)$. On vient de démontrer cette exacte propriété dans l'exercice précédent.

Exercice 12 - Diagonalisation

1. Soit une suite quelconque d'ensembles $E_i \subset \mathbb{N}$. Construire un ensemble qui n'appartient pas à cette suite (en vous inspirant de la diagonalisation).
2. Que pouvons nous conclure sur l'ensemble des sous-ensembles de $\mathbb{N}$?

1. On prend l'ensemble $E = \{e_i \mid \mathcal{X}_{E_i}(e_i) = 0\}$. Cet ensemble est différent de tous les ensembles de E_i, et n'est donc pas dans la suite.
2. L'ensemble des sous-ensembles de $\mathbb{N}$ n'est pas dénombrable.

Exercice 13 - Diagonalisation

Montrer que $[0,1[$ n'est pas dénombrable.

On travaille en binaire dans la suite de cet exercice. On pose la fonction $d_i(x)$ qui renvoie le i-ème décimal après la virgule de x. Si $[0,1[$ est dénombrable, on peut construire un tableau qui contient tous les éléments de $[0,1[$. On prend tous les éléments qui ont des chiffres après la virgule ($x_i = \sum_{k=0}^{\infty} d_k(x_i) \times 2^{-k-1}$) :

	d_0	d_1	d_2	$\ldots$	d_i	$\ldots$
x_0	$\boxed{1}$	1	0	$\ldots$	1	$\ldots$
x_1	1	$\boxed{1}$	0	$\ldots$	1	$\ldots$
x_2	0	0	$\boxed{0}$	$\ldots$	0	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
x_i	1	0	1	$\ldots$	$\boxed{1}$	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Soit le réel x suivant : $x = \sum_{k=0}^{\infty}((-d_k(x_k) + 1) \times 2^{-k-1})$, autrement dit, $d_k(x) = 0$ si $d_k(x_k) = 1$ et $d_k(x) = 1$ sinon. Comme le tableau contient tous les éléments de $[0,1[$, $\exists i.x = x_i$. Or, $d_i(x_i) \neq d_i(x)$ par construction, donc $x \neq x_i$. Contradiction, il n'y a pas de i tel que $x = x_i$, et dans ce cas, le tableau n'énumère pas tous les éléments de $[0,1[$, donc $[0,1[$ n'est pas dénombrable.

Exercice 14 - Diagonalisation

On considère l'ensemble U des suites $(u_n)_{n\in\mathbb{N}}$ à la valeur dans $\{0,1\}$, c'est à dire $\forall n \in \mathbb{N}$. Montrer que U n'est pas dénombrable.

	u_0	u_1	u_2	$\dots$	u_i	$\dots$
U_0	$\boxed{0}$	1	0	$\dots$	0	$\dots$
U_1	0	$\boxed{1}$	1	$\dots$	1	$\dots$
U_2	1	1	$\boxed{1}$	$\dots$	1	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
U_i	0	1	0	$\dots$	$\boxed{1}$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Soit la suite V suivante : $V(i) = 0$ si $U_i(i) = 1$ et $V(i) = 1$ sinon. On suppose $\exists i. V = U_i$. Or, $U_i(i) \neq V(i)$ par construction. Donc, $V \neq U_i$, et ainsi l'ensemble U n'est pas dénombrable.

1.4 Dénombrabilité

Exercice 15 - Ensemble fini/infini

Un ensemble est fini si on ne peut pas le mettre en bijection avec une partie stricte de lui-même. Il est infini sinon.
Montrer que l'ensemble des entiers est infini.

On peut mettre l'ensemble des entiers en bijection avec une partie stricte de lui-même : soit la fonction $f : \mathbb{N} \to \mathbb{N}^*$ où $f(n) = n + 1$. f est en bijection avec une partie stricte de $\mathbb{N}$. On peut exhiber la fonction inverse $q : \mathbb{N}^* \to \mathbb{N}$ avec $q(n) = n - 1$. L'ensemble des entiers est donc bien infini.

Exercice 16 - Taille des ensembles

Soit E un ensemble, et soit $\mathcal{P}(E)$ l'ensemble des parties de E. Montrer que $|E| < |\mathcal{P}(E)|$.

On sait que $|E| \neq |\mathcal{P}(E)|$. On veut montrer que $|E| \leq |\mathcal{P}(E)|$. Pour ce faire, il suffit de trouver $\varphi : E \to \mathcal{P}(E)$ injective, ou bien $\varphi : \mathcal{P}(E) \to E$ surjective. On prend par exemple $\varphi : E \to \mathcal{P}(E)$ avec $\varphi(e) = \{e\}$.

Exercice 17 - Dénombrabilité

1. Donner les bijections :
   (a) de $\mathbb{N}$ sur $\mathbb{N} - \{0\}$.
   (b) de $\mathbb{N}$ sur $2\mathbb{N}$.
   (c) de $\mathbb{N}$ sur $\mathbb{Z}$.
2. Est-ce que la fonction $f(n) = (-1)^n \lceil \frac{n}{2} \rceil$ est une bijection de $\mathbb{N}$ sur $\mathbb{Z}$?
3. Montrer que tout sous-ensemble $X \subset \mathbb{N}$ est dénombrable.
4. Il existe une application $f : X \to \mathbb{N}$ qui est injective si et seulement si X est dénombrable.
5. Un produit fini d'ensembles dénombrables est dénombrable.
6. Il existe une application $f : \mathbb{N} \to X$ qui est surjective si et seulement si X est dénombrable.
7. Soit E un ensemble dénombrable infini. Alors il existe une bijection de $\mathbb{N}$ sur E. Autrement dit, on peut numéroter les éléments de E, i.e. écrire $E = \{e_0, e_1, \ldots, e_n, \ldots\}$.
8. Montrer que $\mathbb{Q}$ est dénombrable.
9. Soit $(E_n)_{n\in\mathbb{N}}$ une famille dénombrable de sous-ensembles dénombrables d'un ensemble E. Montrer que la réunion $\bigcup_{n\in\mathbb{N}} E_n$ est dénombrable.
10. Soit $A = \mathbb{Q} \cap [0, 1]$ et $B = \mathbb{Q} \cap]0, 1[$. Existe-t'il une bijection de A vers B ?

1. (a) $\varphi : \mathbb{N} \to \mathbb{N} - \{0\}$ avec $\varphi(n) = n + 1$.
   (b) $\varphi : \mathbb{N} \to 2\mathbb{N}$ avec $\varphi(n) = 2n$.
   (c) $\varphi : \mathbb{N} \to \mathbb{Z}$ avec $\varphi(n) = \frac{-(n+1)}{2}$ si n est impair et $\varphi(n) = \frac{n}{2}$ sinon.
2. Oui, cette fonction est une bijection de $\mathbb{N}$ sur $\mathbb{Z}$, c'est celle que j'ai décrit en 1.(c).
3. Pour montrer que tout sous-ensemble $X \subset \mathbb{N}$ est dénombrable, il suffit d'exhiber une bijection entre X et $\mathbb{N}$. Soit $X = \{x_0, x_1, \ldots, x_i, \ldots\}$. Soit $f : X \to \mathbb{N}$ l'application suivante : $f(x_i) = i$. On a $f^{-1}(i) = x_i$. Tout sous-ensemble de $\mathbb{N}$ est donc bien dénombrable.
4. $\Rightarrow$ On suppose qu'il existe une application $f : X \to \mathbb{N}$ injective, et on veut montrer que X est dénombrable. On sait que $|X| \leq \mathbb{N}$ par l'existence de la fonction injective. Ainsi, il existe une fonction injective $g : \mathbb{N} \to X$ telle que $g(i) = x_i$ (avec $x_i \in X, \forall i$). Par le théorème de Cantor-Bernstein, il existe donc une bijection entre X et $\mathbb{N}$, ce qui montre que X est dénombrable.

   $\Leftarrow$ X est dénombrable, donc il existe une bijection entre X et $\mathbb{N}$. Ainsi, il existe deux fonctions injectives $g : X \to \mathbb{N}$ et $h : \mathbb{N} \to X$. On pose $g = f$ qui vérifie l'existence d'une application injective de X dans $\mathbb{N}$.
5. Pour prouver que le produit fini d'ensembles dénombrables est dénombrables, prenons $E = \{E_0, E_1, \ldots, E_n\}$ ces ensembles dénombrables. Soient $F = \{f_0, f_1, \ldots, f_n\}$ les bijections de ces ensembles. On peut exhiber la bijection suivante : $h(x_0, x_1, \ldots, x_n) = c(\ldots(c(c(f_0(x_0), f_1(x_1)), \ldots)), f_n(x_n))$, avec c la fonction qui met en bijection $\mathbb{N}^2$ avec $\mathbb{N}$. On a exhibé une bijection entre $E_0 \times E_1 \times \cdots \times E_n$ et $\mathbb{N}$, le produit cartésien des ensembles de E est donc dénombrable.
6. $\Rightarrow$ On suppose qu'il existe une application $f : \mathbb{N} \to X$ surjective, et on veut montrer que X est dénombrable. C'est à dire que $|X| \leq \mathbb{N}$. De plus, si $|X| \leq \mathbb{N}$, alors il existe une application injective de X dans $\mathbb{N}$. Par la question (4), X est dénombrable.

   $\Leftarrow$ X est dénombrable, c'est à dire qu'il existe une fonction $g : \mathbb{N} \to X$ bijective. Comme celle-ci est bijective, elle est injective et surjective. Soit $f = g$, cette application est bien surjective, donc il existe bien une application surjective $f : \mathbb{N} \to X$ si X est dénombrable.
7. Oui, c'est la définition de dénombrable.

8. Évident (on a déjà exhibé la bijection de $\mathbb{Q}$ avec $\mathbb{N}$ précédemment, exercice 6).
9. On peut remarquer que $|\bigcup_{n\in\mathbb{N}} E_n| \leq |\Pi_{n\in\mathbb{N}} E_n|$ et que comme le second est dénombrable, le premier l'est aussi. De la même manière que ce dernier, on peut construire une bijection de $E = \bigcup_{n\in\mathbb{N}} E_n$ dans $\mathbb{N}$ en diagonalisant les éléments des ensembles : soit $e_{i,j}$ le j-ème élément du i-ème ensemble de E. On a :

$$\begin{array}{ccccc} \vdots & \vdots & \vdots & \vdots & \vdots \\ e_{0,j} & e_{1,j} & \dots & e_{i,j} & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ e_{0,1} & e_{1,1} & \dots & e_{i,1} & \dots \\ e_{0,0} & e_{1,0} & \dots & e_{i,0} & \dots \end{array}$$

   En appliquant l'application inverse $f : \mathbb{N}^2 \to \mathbb{N}$, on associe chaque élément $e_{i,j}$ à un entier de $\mathbb{N}$.
10. On peut exhiber la bijection suivante : $f : \mathbb{Q} \cap [0,1] \to \mathbb{Q}\cap]0,1[$ avec $f(x) = \frac{x+1}{3}$. Pour $x = 0$, $f(x) = \frac{1}{3} \in]0,1[$ et pour $x = 1$, $f(x) = \frac{2}{3} \in]0,1[$. De plus, $\forall x \in \mathbb{Q}\cap]0,1[, f(x) = \frac{p+q}{3\times q}$ et ainsi $0 < f(x) < 1$.

1.5 Fonctions (non)-calculables

Exercice 18 - Calculabilité

Soit $f : \mathbb{N} \to \{0,1\}$ une fonction totale non calculable.

1. Rappeler la définition d'une fonction totale et d'une fonction non calculable.
2. Construire une fonction $g : \mathbb{N} \to \mathbb{N}$ totale, croissante et non calculable à partir de f.

1. Une fonction totale est une fonction définie pour tout n. Une fonction non-calculable est une fonction qui ne peut pas être calculée par une procédure automatique. Autrement dit, f est non calculable si et seulement si $\forall n.f(n)\downarrow, \nexists p \forall n.p(n) = f(n)$.
2. On peut construire la fonction g suivante : $g(n) = \sum_{i=0}^{n} f(i)$. Cette fonction est forcément croissante, car l'ensemble d'arrivée de f est $\{0,1\}$ (donc $f(i) + f(i-1) \geq f(i-1)$), $\forall i.f(i)\downarrow$ par hypothèse, donc $(\sum_{i=0}^{n} f(i))\downarrow$.
   Enfin, supposons que g soit calculable. On a $g(n-1) = \sum_{i=0}^{n-1} f(i)$ et $g(n) = \sum_{i=0}^{n} f(i)$, donc $f(n) = g(n) - g(n-1)$, si g est calculable, alors f est calculable. Or, f n'est pas calculable, donc g ne l'est pas non plus (contraposée).

Exercice 19 - Calculabilité

Montrer que l'inverse d'une fonction f calculable et bijective est calculable.

f calculable, cela signifie qu'il existe une procédure automatique p telle que $p(n) = f(n)$. Si on trouve une procédure automatique q qui calcule g, alors g est calculable. Soit la procédure suivante :

```
int q(int n) {
    for (int i = 0; ; ++i) {
        if (p(i) == n)
            return i;
    }
}
```

Comme f est bijective et totale, q termine forcément : pour n'importe quelle entrée n, $p(n)$ termine, et comporte un unique antécédent dans $\mathbb{N}$.

Dans le cas où f est injective, q ne termine pas forcément : la procédure p peut boucler pour certains n. Dans le cas où f est surjective, q renverra $min\{x \mid p(x) = n\}$ le minimum des antécédents de n par p.

Exercice 20 - Calculabilité

Montrer qu'une fonction f totale $\mathbb{N} \to \mathbb{N}$ est calculable si et seulement si son graphe

$$G = \{(x, f(x)) | x \in \mathbb{N}\}$$

est décidable.

Si $E \subset \mathbb{N}$ est décidable, c'est qu'on peut écrire une procédure p qui calcule la fonction caractéristique $\mathcal{X}_E$ de cet ensemble : $\exists p \forall n \in \mathbb{N}. p(n) = \begin{cases} 1 & \text{si } n \in E \\ 0 & \text{sinon} \end{cases}$

$\Rightarrow$ On suppose que f totale est calculable. Montrons que le graphe G_f de f est décidable. La procédure p calcule f. Soit q la procédure qui calcule la fonction caractéristique $\mathcal{X}_{G_f}$:

```
int q(int x, int y) {
    return y == p(x);
}
```

Comme p est calculable, q l'est aussi. On a trouvé une procédure q qui calcule la fonction caractéristique $\mathcal{X}_{G_f}$, donc si f est calculable, alors G_f est décidable.

$\Leftarrow$ On suppose que G_f est décidable. Montrons que la procédure f du graphe G_f est calculable. La procédure q calcule la fonction caractéristique $\mathcal{X}_{G_f}$. Soit p la procédure qui calcule f :

```
int p(int x) {
    for (int i = 0; ; ++i) {
        if (q(x, i)) return i;
    }
}
```

Comme q est calculable, alors p l'est aussi. On a trouvé une procédure p qui calcule f, donc si G_f est décidable, alors f est calculable.

Exercice 21 - Calculabilité

Soit E un ensemble et ϕ une fonction telle que $\phi(n)$ est égale au nombre d'éléments de E strictement inférieurs à n. Montrer que ϕ totale est calculable si et seulement si E est décidable.

$\Rightarrow$ On suppose que ϕ est calculable. Montrons que E est décidable. La procédure p calcule ϕ. Soit q la procédure qui calcule la fonction caractéristique $\mathcal{X}_E$:

```
int q(int x) {
    return p(x) != p(x + 1); // On pourrait aussi dire p(x) == p(x + 1) - 1
}
```

p termine et définie pour tout x, donc q termine et définie pour tout x. On a trouvé une procédure q qui calcule la fonction caractéristique $\mathcal{X}_E$, donc si ϕ calculable, alors E est décidable.

$\Leftarrow$ On suppose que E est décidable, c'est à dire qu'il existe une procédure automatique q telle que q calcule $\mathcal{X}_E$. Montrons que ϕ est calculable. Soit p la procédure qui calcule ϕ :

```
int p(int x) {
    if (x == 0) return 0;
    if (x == 1) return q(0);
    return q(x - 1) + p(x - 1);
}
```

Comme q termine et définie pour tout $n \in \mathbb{N}$, p termine. On a trouvé une procédure p qui calcule ϕ, donc si E décidable, alors ϕ est calculable.

1.6 Problèmes indécidables

Exercice 22 - Variantes du problème de l'arrêt

1. Self-Halt : le programme Self-Halt(p) s'arrête sur p si Halt(p, p) s'arrête où Halt(p, v) désigne le problème de l'arrêt pour un programme p appliquée à des données v.
2. Anti-Self-Halt : le programme Anti-Self-Halt(p) s'arrête si et seulement si p ne s'arrête pas.
3. Montrer que les problèmes suivants sont indécidables.
   (a) Halt$_\exists$: le problème de l'arrêt existentiel, existe-t-il une entrée pour laquelle le programme s'arrête ?
   (b) Halt$_\forall$: le problème de l'arrêt universel, le programme s'arrête-t-il pour toutes les entrées ? La réduction est la même que la précédente.
   (c) Negval : le problème du test de valeur négative, la variable v du programme prend-elle une valeur négative au cours du calcul ? Ce problème est à rapprocher de « cet indice de tableau évolue-t-il toujours dans les bornes du tableau ? »
   (d) Equiv : problème du test d'équivalence de programmes, les programmes P_1 et P_2 ont-ils le même comportement pour toutes les entrées ?
   Pour illustrer l'intérêt de ce problème : on peut se poser la question concernant d'un programme source et d'un programme objet correspondant produit par un compilateur.
   Certaines optimisations tendantes changent le comportement du programme.
   (e) Return$_0$: problème du test de rendu nul, existe-t-il une entrée pour laquelle le programme retourne la valeur 0.

1. Par définition, le problème Self-Halt est indécidable.
2. Par définition, le problème Anti-Self-Halt est indécidable.
3. (a) Supposons l'existence de la procédure automatique p_{HE} qui résout Halt$_\exists$, c'est à dire que $\forall p \in P. p_{HE}(p) = 1$ s'il existe une entrée v telle que $p(v)$ s'arrête et 0 sinon. Soit gamma la procédure automatique complémentaire de Halt$_\exists$ suivante :

```
int gamma(procedure x) {
    if (p_HE(x)) while(1) ;
    else return 0;
}
```

gamma prend en entrée une procédure automatique, et si pour une donnée v cette procédure automatique termine, alors gamma ne termine pas. Si la procédure automatique ne termine pour aucune donnée, alors gamma termine.

Si p_{HE} existe, alors gamma existe. Si on appelle gamma(gamma), que se passe-t-il ? Si gamma termine pour une donnée, il va renvoyer 0. Seulement, s'il fait cela, c'est que gamma ne s'arrête pour aucune donnée.

Si gamma ne s'arrête pas, c'est à dire qu'il rentre dans une boucle infinie, alors il retourne 0 et alors gamma s'arrête.

Par contradiction, gamma ne peut pas exister, et ainsi, p_{HE} n'existe pas non plus.

(b) Supposons l'existence de la procédure automatique p_{HA} qui résout $\text{HALT}_\forall$, c'est à dire que $\forall p \in P.p_{HA}(p) = 1$ si p s'arrête pour toutes les entrées v et $p_{HA}(p) = 0$ sinon. Soit gamma la procédure automatique complémentaire de $\text{HALT}_\forall$ suivante :

```
int gamma(procedure x) {
    if (p_HA(x)) while (1) ;
    else return 0;
}
```

gamma prend en entrée une procédure automatique, et si pour toute donnée v cette procédure automatique termine, alors gamma ne termine pas. Si la procédure automatique ne termine pas pour une donnée, alors gamma termine.

Si p_{HA} existe, alors gamma existe. Si on appelle gamma(gamma), que se passe-t-il ? Si gamma termine pour toute donnée, alors par définition, gamma boucle et ainsi $p_{HA}(\text{gamma}) = 0$.

Si gamma boucle pour une donnée, alors $p_{HA}(\text{gamma}) = 0$ et gamma termine, donc $p_{HA}(\text{gamma}) = 1$.

Par contradiction, gamma ne peut pas exister, et ainsi, p_{HA} n'existe pas non plus.

(c) Supposons l'existence de la procédure automatique p_{NV} qui résout NEGVAL, c'est à dire que $\forall p \in P.p_{NV}(p, v) = 1$ si v prend une valeur négative au cours du calcul, 0 sinon. Soit gamma la procédure automatique complémentaire de NEGVAL suivante :

```
void gamma(procedure x, int v) {
    if (!p_NV(x, v)) {
        v = -1;
        return;
    }
}
```

Si on appelle gamma(gamma, 1), il y a deux cas :

— Si v ne prend pas de valeur négative durant le calcul, alors $v = -1$ et v prend une valeur négative pendant le calcul de gamma, ce qui est une contradiction.

— Si v prend une valeur négative durant le calcul, alors v n'est pas touchée donc $v = 1$ et v ne prend pas de valeur négative durant le calcul de gamma, ce qui est une contradiction.

gamma ne peut pas être calculée, donc p_{NV} ne peut pas l'être non plus.

(d) Supposons l'existence de la procédure automatique p_{EQ} qui résout EQUIV, c'est à dire que $\forall p, q \in P.p_{EQ}(p, q) = 1$ si p et q sont équivalents à chaque étape de calcul, 0 sinon. Soit gamma la procédure automatique complémentaire de EQUIV suivante :

```
int gamma(procedure p, procedure q) {
    if (p_EQ(p, q)) return random();
    return 1;
}
```

Si on appelle gamma(gamma, gamma), il y a deux cas :

— Si gamma et gamma sont équivalents à chaque étape de calcul, alors gamma et gamma renvoient une valeur différente ($\frac{1}{\infty} \approx 0$ chances d'avoir un entier égal), ce qui est une contradiction.

— Si gamma et gamma ne sont pas équivalents à chaque étape de calcul, alors gamma et gamma renvoient 1, et sont équivalents à chaque étape de calcul, ce qui est une contradiction.

gamma ne peut pas être calculé, donc p_{EQ} ne peut pas l'être non plus.

(e) Supposons l'existence de la procédure automatique p_{R0} qui résout RETURN_0, c'est à dire que $\forall p \in P. p_{R0}(p, v) = 1$ si p renvoie 0 sur l'entrée v, 0 sinon. Soit gamma la procédure automatique complémentaire de RETURN_0 suivante :

```
int gamma(procedure x, int v) {
    if (p_R0(x, v)) return 1;
    else return 0;
}
```

Si on appelle gamma(gamma, 0), il y a deux cas :

— Si gamma renvoie 0, alors gamma va renvoyer 1, ce qui est une contradiction.

— Si gamma renvoie 1, alors gamma va renvoyer 0, ce qui est une contradiction.

gamma ne peut pas être calculée, donc p_{R0} non plus.

1.7 Théorème de Rice

Exercice 23 - Calculabilité

En vous inspirant du théorème de Rice, donnez le prédicat (indécidable) et la fonction contradictoire qui prouve par l'absurde le résultat d'indécidabilité pour chacun des exemples suivants : on ne peut décider si une procédure calcule

1. une fonction totale
2. une fonction injective
3. une fonction croissante
4. une fonction à valeurs bornées

1. Soit le prédicat P tel que $P(p) = 1$ si p calcule une fonction totale, et 0 sinon. Le prédicat n'est pas trivial : soit rac la procédure qui ne calcule pas une fonction totale :

```
int rac(int n) {
    r = 0;
    while (r * r != n) r += 1;
    return r;
}
```

$P(\text{rac}) = 0$, car si un nombre n'a pas de racine carré entière, alors cette procédure ne s'arrêtera pas, elle n'est pas totale. Soit la procédure id qui calcule une fonction totale :

```
int id(int n) { return n; }
```

$P(\text{id}) = 1$, car la procédure automatique termine pour n'importe quel élément de $\mathbb{N}$. Le prédicat P n'est pas trivial, donc on ne peut pas décider si une procédure calcule une fonction totale.

2. Soit le prédicat P tel que $P(p) = 1$ si p calcule une fonction injective, et 0 sinon. Le prédicat n'est pas trivial : soit p_0 la procédure qui ne calcule pas une fonction injective :

```
int p_0(int n) {
    if (n % 2 == 0) return n;
    else return 2*n;
}
```

$P(p_0) = 0$, car $p_0(3) = p_0(6) = 6$, la fonction a deux antécédents pour la même image, elle n'est donc pas injective. Soit la procédure id qui calcule une fonction injective :

```
int id(int n) { return n; }
```

$P(\mathsf{id}) = 1$, car la procédure automatique associe une seule image pour chaque antécédent de $\mathbb{N}$. Le prédicat P n'est pas trivial, donc on ne peut pas décider si une procédure calcule une fonction injective.

3. Soit le prédicat P tel que $P(p) = 1$ si p calcule une fonction croissante, et 0 sinon. Le prédicat n'est pas trivial : soit p_0 la procédure qui ne calcule pas une fonction croissante :

```
int p_0(int n) {
    return n % 2;
}
```

$P(p_0) = 0$, car $p_0(2) = 0 < p_0(1) = 1$, la fonction n'est donc pas croissante. Soit la procédure id qui calcule une fonction croissante :

```
int id(int n) { return n; }
```

$P(\mathsf{id}) = 1$, car $\mathsf{id}(n) = \mathsf{id}(n-1) + 1$. Le prédicat P n'est pas trivial, donc on ne peut pas décider si une procédure calcule une fonction croissante.

4. Soit le prédicat P tel que $P(p) = 1$ si p calcule une fonction à valeurs bornées, et 0 sinon. Le prédicat n'est pas trivial : soit id la procédure qui ne calcule pas une fonction valeurs bornées :

```
int id(int n) { return n; }
```

$P(\mathsf{id}) = 0$, car on peut toujours trouver une valeur de id supérieure à une valeur k fixée, la fonction n'est donc pas à valeurs bornées. Soit la procédure p_1 qui calcule une fonction à valeurs bornées :

```
int p_1(int n) { return n % 17; }
```

$P(p_1) = 1$, car $0 \leq p_1(n) < 17$. Le prédicat P n'est pas trivial, donc on ne peut pas décider si une procédure calcule une fonction à valeurs bornées.

1.8 Décidabilité et récursivement énumérable

Exercice 24 - Récursivement énumérable

Soit A un ensemble énumérable et $P : A \to Bool$ un programme total tel que $\forall a \in A.P(a) \downarrow$. Alors l'ensemble $B := \{a \in A | P(a) = Vrai\}$ est énumérable.

On rappelle que si E est énumérable, cela signifie qu'il existe une procédure p_E qui affiche tous les éléments de E. Si on peut construire cette procédure, on obtient ce qu'on appelle une preuve constructive.

A est énumérable, c'est à dire qu'il y a une procédure p_A qui affiche tous les éléments de A. On pose la procédure $q_A : \mathbb{N} \to Bool$: $\begin{cases} 1 & \text{si } n \in \mathbb{N} \text{ est affiché par } p_A \\ 0 & \text{sinon} \end{cases}$

Cette procédure termine : comme tous les éléments de A sont affichés par p_A dans l'ordre $(a_0 < a_1 < \cdots < a_i < \cdots)$, si on trouve k tel que $n = a_k$, q_A renvoie vrai, et si on ne trouve pas ce k mais que pour un certain i, $a_i > n$, q_A renvoie faux. On peut alors construire le programme p_B qui énumère B suivant :

```
void p_B() {
    for (int i = 0; ; ++i) {
        if (q_A(i) && P(i)) {
            afficher(i);
        }
    }
}
```

La procédure p_B termine car $q_A(n)$ totale et termine $\forall n \in \mathbb{N}$ et renvoie vrai si et seulement si $n \in A$, et $\forall a \in A.P(a) \downarrow$.

Exercice 25 - Calculabilité

Soit E l'ensemble $val(f)$ où f est calculable et partielle.
Montrer que E est récursivement énumérable (inpirez-vous du fait que l'arrêt en t unités de temps est décidable).

Comme f est calculable, on sait qu'il existe une procédure automatique $p_f : \mathbb{N} \to \mathbb{N}$ qui termine pour tout $n \in D_f$. Soit $A = \{n \in \mathbb{N} | p(n) \downarrow\}$ et $E = \{p_f(n) | p(n)$ s'arrête en n étapes$\}$. On a $E \subseteq A$. Montrons d'abord que A est énumérable. Il suffit de trouver la procédure automatique p_A qui énumère A. Comme f est partielle, si on envoie $m \notin D_f$ à p_f, on ne sait pas si la procédure termine. Ainsi, on défini une surcouche en étapes de p_f : $p_{f_t} : \mathbb{N} \times \mathbb{N} \to Bool$. $p_{f_t}(n, t)$ renvoie vrai si $p_f(n)$ termine en t étapes, faux sinon. On peut ainsi définir p_A :

```
void p_A() {
    for (int i = 0; ; ++i) {
        for (int k = 0; k <= i; ++k) {
            if (p_f_t(k, i))
                afficher p_f(k);
        }
    }
}
```

Comme p_{f_t} termine pour n'importe quelles valeurs et que p_f termine en particulier pour les valeurs où p_{f_t} termine en un nombre d'étapes finies, p_A énumère bien tous les éléments de A. On peut cependant remarquer que le même élément peut être affiché plusieurs fois.

La procédure automatique p_B est assez similaire à p_A, et énumère tous les éléments de B pour les même raisons :

```
void p_B() {
    for (int i = 0; ; ++i) {
        if (p_f_t(i, i)) {
            afficher p_f(i);
        }
    }
}
```

Exercice 26 - Calculabilité

Soit f une fonction calculable, un ensemble B et son image réciproque par f, A :

$$A = f^{-1}(B) = \{x | f(x) \in B\}$$

1. Rappeler la définition d'un ensemble décidable et d'un ensemble récursivement énumérable.
2. A-t-on B décidable implique A décidable ?
3. A-t-on B récursivement énumérable implique A récursivement énumérable ?

1. Un ensemble est décidable s'il y a une procédure automatique qui calcule sa fonction caractéristique.
   Un ensemble est récursivement énumérable s'il y a une procédure automatique qui calcule sa fonction semi-caractéristique (donc énumère tous les éléments de l'ensemble).
2. A est décidable si et seulement si f est totale. En effet, dans ce cas, on peut écrire la procédure suivante : `bool X_A(int n) { return X_B(p_f(n)); }` (avec $\mathcal{X}_B$ la fonction caractéristique de B et p_f la procédure automatique qui calcule la fonction f). On voit bien que si f n'est pas totale, $\mathcal{X}_A$ ne termine pas toujours.

3. En utilisant une surcouche de p_f avec le nombre d'étapes, comme à l'exercice précédent, et qu'on défini q_B de manière analogue à l'exercice 24 ($q_B : \mathbb{N} \to Bool : \begin{cases} 1 & \text{si } n \in \mathbb{N} \text{ est affiché par } p_B \\ 0 & \text{sinon} \end{cases}$) si B est récursivement énumérable, alors A l'est aussi :

```
void p_A() {
    for (int i = 0; ; ++i) {
        for (int k = 0; k <= i; ++k) {
            if (q_B(f(i, k)))
                afficher(i)
        }
    }
}
```

Exercice 27 - Calculabilité

1. Montrer qu'un ensemble énuméré par une fonction calculable strictement croissante f est décidable.
2. En déduire que tout ensemble récursivement énumérable non décidable contient un sous-ensemble infini et décidable.

1. f est calculable, et par définition, comme l'ensemble est énuméré par une fonction, celle-ci est totale. On peut donc écrire la fonction caractéristique $\mathcal{X}_E$ de l'ensemble E énuméré par f :

```
bool X_E(int n) {
    for (int i = 0; ; ++i) {
        if (p_f(i) == n) return true;
        if (p_f(i) > n) return false;
    }
}
```

2. Si l'ensemble A est énumérable non décidable, ça veut dire qu'il existe une procédure automatique p_A qui affiche tous les éléments de cet ensemble. On suppose que $p_f : \mathbb{N} \to \mathbb{N}$ stocke les éléments de A au fur et à mesure de l'énumération, et qu'à chaque fois que la procédure est appelée, le premier élément stocké est renvoyé et supprimé. On peut écrire la procédure automatique suivante qui calcule la fonction caractéristique d'un sous-ensemble infini décidable :

```
bool X_B(int n) {
    int dernier = 0;
    for (int i = 0; ; ++i) {
        if (p_f(i) > dernier) {
            if (p_f(i) == n) return true;
            if (p_f(i) > n) return false;
            dernier = p_f(i);
        }
    }
}
```

Exercice 28 - Calculabilité

Montrer que tout ensemble récursivement énumérable peut-être énuméré par une fonction sans répétition.

Il suffit de garder en mémoire tous les éléments de l'ensemble qui ont été affichés, et si un élément l'a déjà été, on ne l'affiche pas.

Exercice 29 - Calculabilité

Soient A et B deux ensembles décidables.

1. Est-on sûrs que le complémentaire de A ($\bar{A}$) est décidable ?
2. Est-on sûrs que l'union de A et B est décidable ?
3. Est-on sûrs que l'intersection de A et B est décidable ?
4. Même question en remplaçant décidables par récursivement énumérables.

On suppose A et B décidables, il existe les procédures automatiques p_A et p_B telles que $\forall n \in \mathbb{N}. p_A(n) \downarrow$ et $p_B(n) \downarrow$ qui calculent la fonction caractéristique de ces deux ensembles.

1. Oui, on peut exhiber la procédure automatique qui correspond à la fonction caractéristique de $\bar{A}$:
   ```
   bool p_Abar(int x) { return !p_A(x); }
   ```
2. Oui, on peut exhiber la procédure automatique qui correspond à la fonction caractéristique de $A \cup B$:
   ```
   bool p_AUB(int x) { return p_A(x) || p_B(x); }
   ```
3. Oui, on peut exhiber la procédure automatique qui correspond à la fonction caractéristique de $A \cap B$:
   ```
   bool p_AIB(int x) { return p_A(x) && p_B(x); }
   ```

On suppose maintenant A et B récursivement énumérables. C'est à dire qu'il existe deux procédures automatiques p_A et p_B qui affichent tous les éléments de A et respectivement B dans la sortie standard.

1. On ne peut rien dire sur $\bar{A}$. Cependant, si celui-ci est bien récursivement énumérable, A (et $\bar{A}$) sont décidables :
   ```
   bool P_Aprime(int n) {
       int nbrE = 1;
       while (1) {
           Calculer P_A en nbrE;
           if (P_A affiche n) { return true; }
           Calculer P_Abar en nbrE;
           if (P_Abar affiche n) { return false; }
           ++nbrE;
       }
   }
   ```
2. Oui, on peut exhiber la fonction qui énumère tous les éléments de $A \cup B$:
   ```
   void P_AUB() {
       int n = 0;
       while (1) {
           for (int i = 0; i < n; ++i) {
               if (P_A affiche i en n etapes) { afficher(i); }
               else if (P_B affiche i en n etapes) { afficher(i); }
           }
           ++n;
       }
   }
   ```
3. En utilisant la même astuce, on peut afficher tous les éléments de $A \cap B$:

```
void P_AIB() {
    int n = 0;
    while (1) {
        for (int i = 0; i < n; ++i) {
            if ((P_A affiche i en n etapes) &&
                (P_B affiche i en n etapes)) {
                    afficher(i);
            }
        }
        ++n;
    }
}
```

Exercice 30 - Calculabilité

1. Soit A un ensemble décidable de couples d'entiers. Montrer que la projection de A à savoir $E = \{x | \exists y.(x, y) \in A\}$ est récursivement énumérable.
2. Montrer que réciproquement, tout ensemble récursivement énumérable est la projection d'un ensemble décidable.

1. On peut exhiber la fonction semi-caractéristique p_E de E :

```
void p_E(int x) {
    for (int i = 0; ; ++i) {
        if (p_A(x, i)) return 1;
    }
}
```

Comme A est décidable, il est assez évident que cette procédure termine pour tout x qui a une image dans A, et elle ne termine pas sinon.

2. On peut construire $p_B : \mathbb{N} \times \mathbb{N} \to \mathcal{B}$ tel que vrai si $(i, j) \in B$ et faux sinon :

```
bool p_B(int x, int y) {
    return P_A(x) == y;
}
```

Exercice 31 - Concept de la réduction

Pour deux sous-ensembles A et B de $\mathbb{N}$, on dit que A se réduit à B (ce qu'on note $A \propto B$) s'il existe une fonction totale f telle que $\forall x \in \mathbb{N}, x \in A \Leftrightarrow f(x) \in B$.

1. Montrer que si B est décidable et $A \propto B$, alors A est décidable.
2. Montrer que si B est récursivement énumérable et $A \propto B$, alors A est récursivement énumérable.

1. On peut exhiber la procédure automatique qui calcule la fonction caractéristique de A :

```
bool p_A(int x) {
    return p_B(p_f(x));
}
```

Comme B décidable, p_B termine. Comme f totale et calculable, p_f termine pour tout entier. Donc p_A termine pour tout entier, et p_A calcule la fonction caractéristique de A, donc A est décidable.

2. On peut exhiber la procédure automatique qui calcule la fonction semi-caractéristique de A :

```
bool p_A(int x) {
    return p_B(p_f(x));
}
```

Comme B récursivement énumérable, p_B termine si $f(x) \in B$. Comme f est totale et calculable, p_f termine pour tout entier. Donc, p_A termine pour tout entier $n \in A$. p_A calcule bien la fonction semi-caractéristique de A.

1.9 Sur le point fixe

Exercice 32 - Exemples

Donner les points fixes pour les fonctions suivantes :

1. $\mathbb{N} \to \mathbb{N}, x \mapsto x + 1$
2. $\mathbb{N} \to \mathbb{N}, x \mapsto 0 * x$
3. $\mathbb{N} \to \mathbb{N}, x \mapsto \begin{cases} x/2 & \text{si } x \text{ est pair} \\ x & \text{sinon} \end{cases}$
4. $\mathbb{N} \to \mathbb{N}, x \mapsto \begin{cases} x & \text{si } x \geq n \\ x+1 & \text{sinon} \end{cases}$
5. $\mathbb{N}^* \to \mathbb{N}, x \mapsto \begin{cases} x & \text{si } x \geq n \\ x-1 & \text{sinon} \end{cases}$
6. $\mathbb{N}^* \times \mathbb{N} \to \mathbb{N} \times \mathbb{N}, \langle x, y \rangle \mapsto \begin{cases} \langle x-1, y+1 \rangle & \text{si } x > 0 \\ \langle x, y \rangle & \text{sinon} \end{cases}$

Les points fixes sont l'ensemble $F = \{x | x = f(x)\}$

1. $F = \emptyset$
2. $F = \{0\}$
3. $F = \{0\} \cup \{x | x \mod 2 \neq 0\}$
4. $F = \{x | x \geq n\}$
5. $F = \{x | x \geq n\}$
6. $F = \{\langle 0, y \rangle | y \in \mathbb{N}\}$

La calculabilité, c'est fini. Cependant, le domaine est bien plus vaste et dans ce cours, on a seulement écorché la surface. Un ouvrage très bien écrit sur ce domaine est : « Calculabilité » de Benoit Monin et Ludovic Patey (et en plus, il est en français). Si vous êtes intéressé par le domaine, je le conseille fortement.

2 Complexité

2.1 Rappel

Exercice 33 - Une certaine idée de la complexité

Soit la fonction C suivante :

```
int pf(int x) {
    int y = pg(x);
    return ph(y);
}
```

1. Quelle est la complexité du calcul de pf si pg est de complexité $O(n^4)$, ph de complexité linéaire et si $g(n) < n^2$ (g étant la fonction calculée par pg) ?
2. Si ph s'exécute en temps polynomial, à quelle condition le calcul de pf se fait-il en temps polynomial ?
3. Si les hypothèses de la question précédente est vérifiée, que peut-on en déduire si la fonction h calculée par ph se calcule en temps polynomial ?
4. Soit le calcul de Fibonacci en utilisant directement la formule de récurrence : $f_0 = 1$, $f_1 = 1$, $\forall n > 1. f_n = f_{n-1} + f_{n-2}$. Montrons que le nombre d'additions nécessaires pour faire le calcul est compris entre $\sqrt{2}^n$ et 2^n ?
5. Comment améliorer pour que ce nombre soit en $O(n)$? Peut-on déduire qu'il existe un algorithme qui calcule f_n avec un nombre d'additions polynomial par rapport à la taille de la donnée ? Pourquoi ?
6. Questions difficiles : comment calculer f_n avec un nombre d'additions et de multiplications polynomial par rapport à la taille de la donnée ? Peut-on trouver un algorithme qui s'exécute en temps polynomial par rapport à la taille de la donnée ?

1. On sait que :

$$\begin{aligned} O(\mathsf{pg}) &\subseteq O(n^4) \\ O(\mathsf{ph}) &\subseteq O(n) \\ O(\mathsf{pf}) &\subseteq O(\mathsf{pg}) + O(\mathsf{ph}) \end{aligned}$$

   De plus, comme $g(n) < n^2$, la taille de la donnée en entrée de ph est donc au plus de n^2 de la taille de l'entrée de pf, donc $O(\mathsf{ph}(\mathsf{pg}))) = O(n^2)$.
   Ainsi :

$$\begin{aligned} O(\mathsf{ph}) &\subseteq O(\mathsf{pg}) + O(\mathsf{ph}) \\ &\subseteq O(n^4) + O(n^2) \\ &\subseteq O(n^4) \end{aligned}$$

2. On sait que ph se calcule en temps polynomial. Pour que pf se calcule aussi en temps polynomial, il faut que :
   — n soit de taille polynomiale ;
   — pg se calcule en temps polynomial ;
   — $g(n)$ renvoie une donnée de taille polynomiale.
3. Si la fonction h calculée par ph se calcule en temps polynomial, et que les hypothèses précédentes sont vérifiées, alors f se calcule len temps polynomial.

4. La propriété est fausse : on peut citer le contre-exemple suivant : pour $n = 2$, $f_2 = f_1 + f_0$. Pour 0 et 1, il n'y a pas d'addition, donc le nombre d'additions pour f_2 est 1. Or, $\sqrt{2}^2 = 2$ et $2^2 = 4$, et comme $1 < \sqrt{2}^2$, on n'a pas $\sqrt{2}^2 \nleqslant 1 \nleqslant 2^2$.
   On peut cependant affirmer que le nombre d'addition nécessaires reste en $O(2^n)$ avec cette formule de récurrence (c'est assez évident si on représente les additions sous forme d'arbre binaire).
5. Pour que le nombre d'additions soit en $O(n)$, il suffit stocker chaque valeur de la suite dans un tableau, et faire appel à ces éléments pour calculer le nombre qui nous intéresse :

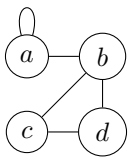
```
int fibonacci(int n) {
    int F[n]; F[0] = 1; F[1] = 1;
    for (int i = 2; i <= n; ++i)
        F[i] = F[i - 1] + F[i - 2];
    return F[n];
}
```

   On ne peut cependant pas déduire qu'il existe un algorithme qui calcule f_n avec un nombre d'additions polynomial par rapport à la taille de la donnée. En effet, comme la taille est $O(log(n))$, le nombre d'additions reste exponentiel : soit $k = log(n)$ la taille de n. L'algorithme s'exécute en temps $O(n)$, mais comme $n = 2^k$, sur la taille de l'entrée, cet algorithme fait $O(2^k)$ additions. Le nombre d'additions est exponentiel par rapport à la taille de l'entrée.

Exercice 34 - Sur le codage d'un graphe

> Donner la taille en nombre de bits pour coder un graphe en utilisant une matrice d'adjacence et listes chaînées.

Soit un graphe $G = (V, E)$ avec V les sommets et E les arêtes. Soit $n = |V|$ et $m = |E|$. Ce graphe peut être représenté de deux manières : par une matrice d'adjacence $\mathcal{M}_{n,n}$ où chaque case $\mathcal{M}_{[i,j]}$ est 1 si $(i, j) \in E$ et 0 sinon, ou bien un tableau de listes chaînées : $T_{[i]}$ contient tous les sommets tels que $(i, v) \in E$ (pour $v \in V$). Par exemple, soit G_0 le graphe non-orienté suivant :



Représentation graphique de G_0

On a $G_0 = (\{a, b, c, d\}, \{(a, a), (a, b), (b, c), (b, d), (c, d)\})$. En donnant les indices suivants aux différents sommets : $a = 0$, $b = 1$, $c = 2$, $d = 3$, on peut définir la matrice $\mathcal{M}_{G_0}$ et la liste d'adjacence $\mathcal{L}_{G_0}$:

a	1	1	0	0
b	1	0	1	1
c	0	1	0	1
d	0	1	1	0

$\mathcal{M}_{G_0}$

a	0	1	
b	0	2	3
c	1	3	
d	1	2	

$\mathcal{L}_{G_0}$

Le nombre de bits nécessaires pour coder $\mathcal{M}_G$ pour n'importe quel graphe G est donc n^2 (car la matrice sera toujours de taille $n \times n$, quelles que soient les arêtes), et le nombre de bits nécessaires pour coder $\mathcal{L}_G$ est en $O(n + m)$ (qui se réduit à n^2 si le graphe est complet).

Le compromis est d'utiliser une matrice d'adjacence lorsque le graphe est dense, et une liste d'adjacence dans le cas contraire.

Exercice 35 - Certificat

Si pour un problème Π vous avez un certificat polynomial pour une réponse positive et un certificat polynomial pour une réponse négative. Que pouvez-vous conclure ? Justifiez votre réponse.
On connaît un algorithme simple en $O(\sqrt{n})$ pour savoir si un nombre n est premier. Peut-on en déduire que savoir si un nombre est premier s'exécute en temps sous-linéaire ?
On peut savoir si n peut s'écrire comme le produit de deux nombres premiers et on connait un algorithme en $O(\sqrt{n})$. Peut-on en déduire que ce problème est dans $\mathcal{P}$? Quel serait l'impact si ce problème était dans $\mathcal{P}$?

Si pour un problème, on a un certificat polynomial pour une réponse positive et négative, on peut seulement dire que le problème appartient à la classe $\mathcal{NP} \cap co\mathcal{NP}$.

Pour l'algorithme des nombres premiers en $O(\sqrt{n})$, l'algorithme est sous-linéaire sur la donnée, mais pour calculer la complexité, il faut calculer selon la taille de la donnée. Comme pour l'exercice 33, la taille de la donnée est en $O(log(n))$, donc par le même raisonnement, si on pose $k = log(n)$, l'algorithme est en complexité $O(\sqrt{2^k})$ (car $n = 2^k$), qui est exponentiel.

Par le même argument que dans le paragraphe précédent, on ne peut pas déduire que ce problème est dans $\mathcal{P}$. Si ce problème était dans $\mathcal{P}$, comme il se réduit à SAT, alors on aurait $\mathcal{P} = \mathcal{NP}$, et les systèmes informatiques sécurisés du monde entier s'effondreraient.

Exercice 36 - Puissance de calcul

Tous les 4 ans, la puissance des machines est multipliée environ par 8. Vous avez deux algorithmes A et B l'un dont le temps d'exécution est proportionnel à n^3 et l'autre dont le temps d'exécution est proportionnel à 2^n. Avec les deux algorithmes vous traitiez un problème de taille $n = 10$ en 1s, il y a 40 ans. Quelle est la taille des problèmes que vous êtes capables de traiter aujourd'hui avec chacun des deux algorithmes en 1s ?

Commençons par compter le nombre de calcul dans les procédures A et B. On pose C_A le nombre de calculs dans la procédure A (en $O(n^3)$) et C_B le nombre de calculs dans la procédure B (en $O(2^n)$) il y a 40 ans :

$$C_A = 10^3 = 1000$$
$$C_B = 2^{10} = 1024 \simeq 1000$$

La puissance des machines étant multipliée par 8 tous les 4 ans, en 40 ans, la puissance des machines à donc été multipliée par 8^{10}. On remarque que :

$$\begin{aligned} 8^{10} &= (2^3)^{10} \\ &= (2^{10})^3 \\ &\simeq (10^3)^3 \\ &= 10^9 \end{aligned}$$

Il suffit maintenant de multiplier C_A et C_B par ce coefficient et d'inverser les résultats trouvés par la complexité de la fonction pour trouver le nombre de données traitées en 1 seconde par les algorithmes A et B de nos jours :

$$
\begin{aligned}
n_A^3 &= 10^9 \times 10^3 \\
n_A &= \sqrt[3]{10^9} \times \sqrt[3]{10^3} \\
n_A &= 10^3 \times 10 \\
n_A &= 10^4
\end{aligned}
$$

L'algorithme A traite 1000 fois plus de données qu'il y a 40 ans, il en traite 10000 au lieu de 10 en une seconde.

$$
\begin{aligned}
2_B^n &= 10^9 \times 2^{10} \\
n_B &= log(10^9 \times 2^{10}) \\
n_B &= log(10^9) + log(2^{10}) \\
n_B &= 9log(10) + 10 \\
n_B &\simeq 40
\end{aligned}
$$

L'algorithme B traite une trentaine de données en plus qu'il y a 40 ans (40 au lieu de 10 en une seconde).

2.2 Autour des classes $\mathcal{P}$ et $\mathcal{NP}$

Exercice 37 - 2-Satisfaisabilité

1. Montrer en calculant le nombre de clauses créées et le nombre de variables ajoutées que la réduction de Satisfaisabilité à 3-Satisfaisabilité vue en cours est bien polynomiale.
2. Sachant que 2-Satisfaisabilité peut-être résolu en temps polynomial, appliquer l'algorithme pour les 2 instances suivantes :
   - $\phi_1 = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_4) \wedge (x_1 \vee x_4) \wedge (x_1 \vee x_3) \wedge (x_4 \vee \neg x_2)$
   - $\phi_2 = (x_1 \vee x_3) \wedge (x_2 \vee \neg x_4) \wedge (x_1 \vee x_4) \wedge (x_4 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3)$
3. Quel problème d'optimisation pouvons-nous étudier dans le cas où la réponse est négative à l'existence d'une solution pour 2-Satisfaisabilité ?

1. Une instance du problème I contient n clauses. Chaque clause i $(0 < i \leq n)$ contient un nombre fini k_i de littéraux. On pose $r = \sum_{i=1}^{n} k_i$. On veut montrer que la réduction vue en cours est polynomiale. On rappelle la réduction vue en cours :

   Considérons une clause $C_i = \{l_{i_1}, l_{i_2}, \ldots, l_{i_k}\}$ de ϕ :
   - Si $k_i = 1$, alors nous introduisons deux nouvelles variables y_{i_1} et y_{i_2} et construisons la formule $\phi_i' = (l_{i_1} \vee y_{i_1} \vee y_{i_2}) \wedge (l_{i_1} \vee y_{i_1} \vee \neg y_{i_2}) \wedge (l_{i_1} \vee \neg y_{i_1} \vee y_{i_2}) \wedge (l_{i_1} \vee \neg y_{i_1} \vee \neg y_{i_2})$
   - Si $k_i = 2$, alors nous introduisons une nouvelle variable y_{i_1} et construisons la formule $\phi_i' = (l_{i_1} \vee l_{i_2} \vee y_{i_1}) \wedge (l_{i_1} \vee l_{i_2} \vee \neg y_{i_1})$
   - Si $k_i = 3$, alors la clause reste inchangée
   - Si $k > 3$, nous introduisons $k-3$ nouvelles variables $y_{i_1}, y_{i_2}, \ldots, y_{i_{k-3}}$ et construisons alors la formule $\phi_i' = (l_{i_1} \vee l_{i_2} \vee y_{i_1}) \wedge_{1 \leq j \leq k-4} (\neg y_{i_j} \vee l_{j+2} \vee y_{i_{j+1}}) \wedge (\neg y_{i_{k-3}} \vee l_{i_{k_i-1}} \vee l_{i_{k_i}})$. La formule ϕ_i' est la conjonction des formules obtenues comme décrit ci-dessus.

   Ainsi, il y a 4 cas à considérer :

— Soit $k_i = 1$, et dans ce cas, deux nouvelles variables sont générées, et trois nouvelle clause sont créées (tandis que l'ancienne est transformée) $\Rightarrow k_i \to 12$.

— Soit $k_i = 2$, et dans ce cas, une nouvelle variable est générée, et une nouvelle clause est créée (tandis que l'ancienne est transformée) $\Rightarrow k_i \to 6$.

— Soit $k_i = 3$, et dans ce cas, aucune nouvelle variable ni aucune nouvelle clause ne s'ajoute à la liste.

— Soit $k_i > 3$, et dans ce cas, $k_i - 3$ nouvelles variables sont créées et $k_i - 3$ clauses sont ajoutées $\Rightarrow k_i \to 3(k_i - 2)$.

> Le principe de cette réduction est d'introduire $k_i - 3$ nouvelles variables « dummy », c'est à dire, avec une valeur de vérité qui n'affecte pas la satisfaisabilité de la formule. Si la formule est satisfiable, elle le reste. Sinon, elle ne peut pas l'être, quelle que soit la valeur de vérité des nouvelles variables introduites. Par exemple, prenons la clause suivante :
>
> $$\phi_i = x_1 \vee x_2 \vee \neg x_3 \vee x_4 \vee x_5$$
>
> Cette clause va introduire $k_i - 3 = 2$ nouvelles variables : z_1 et z_2. La transformation de la clause ϕ_i en clauses de 3-SATISFAISABILITÉ donne la conjonction ϕ_i' suivante :
>
> $$\phi_i' = (x_1 \vee x_2 \vee z_1) \wedge (\neg x_3 \vee \neg z_1 \vee z_2) \wedge (x_4 \vee x_5 \vee \neg z_2)$$
>
> La clause est satisfiable avec $\alpha(x_1) = \alpha(x_3) = \top$, et $\alpha(x_i) = \bot, i \in \{2, 4, 5\}$. On peut trouver des affectations de z_1, z_2 qui satisfont les clauses de ϕ_i' : $\alpha(z_1) = \alpha(z_2) = \bot$. Cependant, si on a une affectation qui ne satisfait pas ϕ_i, par exemple : $\alpha(x_i) = \bot, i \in \{1, 2, 4, 5\}$ et $\alpha(x_3) = \top$, on ne peut pas trouver d'affectation qui satisfasse ϕ_i' grâce à z_1 et z_2 : il faut que $\alpha(z_1) = \alpha(z_2) = \top$ pour satisfaire les deux premières clauses de ϕ_i', mais dans ce cas, la troisième ne sera pas satisfaite.

Dans le pire des cas, le nombre de clause généré est linéaire (entre $\times 3$ et $\times 12$). La réduction est donc bien polynomiale.

2. L'algorithme polynomial de 2-SATISFAISABILITÉ est un algorithme qui transforme les clauses en implication logique, afin de les placer sur un graphe orienté, une implication logique représentant une arête, et chaque variable propositionnelle étant un sommet.

   En effet, on remarque qu'en logique, la clause $(a \vee b)$ est équivalente à $\neg a \Rightarrow b$, ou encore à $\neg b \Rightarrow a$ (par sémantique, ou bien par contraposée).

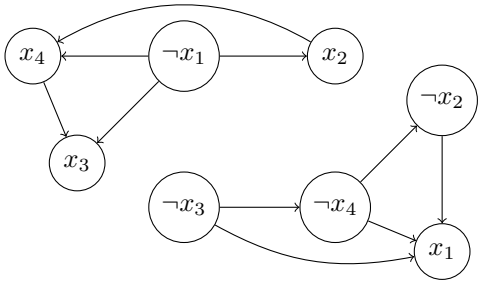
   Le but de l'algorithme est d'abord de trouver des contradictions : si dans une composante fortement connexe du graphe se trouvent a et $\neg a$, alors on a une formule du style $a \Leftrightarrow \neg a$, ce qui est absurde.

   Ensuite, une valeur de vérité est assignée à chaque sommet ($\top$ pour les sommets de polarité positive, et $\bot$ pour les sommets de polarité négative) tant qu'il n'y a pas d'affectation pour tous les sommets.

   En suivant cet algorithme sur ϕ_1, la transformation sera :

   $$\phi_1 = ((\neg x_1 \Rightarrow x_2) \vee (\neg x_2 \Rightarrow x_1)) \wedge ((\neg x_3 \Rightarrow \neg x_4) \vee (x_4 \Rightarrow x_3)) \wedge ((\neg x_1 \Rightarrow x_4) \vee (\neg x_4 \Rightarrow x_1)) \\ \wedge ((\neg x_1 \Rightarrow x_3) \vee (\neg x_3 \Rightarrow x_1)) \wedge ((\neg x_4 \Rightarrow \neg x_2) \vee (x_2 \Rightarrow x_4))$$

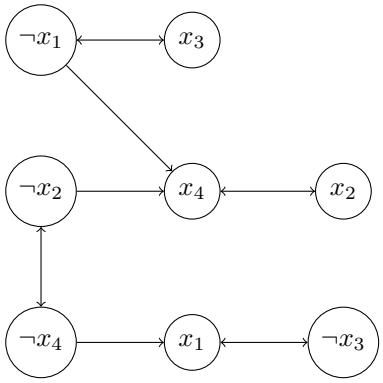
   Ce qui nous donne le graphe suivant :



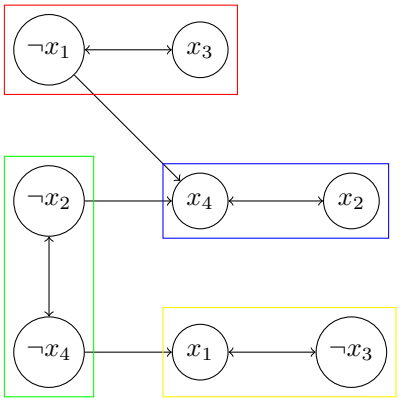
Dans ce graphe, il y a 2 composantes connexes, et chaque composante fortement connexe est composée d'un seul sommet (il n'y a pas de cycle). On peut en déduire que la formule est satisfiable. Une affectation possible est $x_1 = \top$, $x_2 = \top$, $x_3 = \top$, $x_4 = \top$.
Le cas de ϕ_2 est plus intéressant :

$$\phi_2 = ((\neg x_1 \Rightarrow x_3) \vee (\neg x_3 \Rightarrow x_1)) \wedge ((\neg x_2 \Rightarrow x_4) \vee (x_4 \Rightarrow x_2)) \wedge ((\neg x_1 \Rightarrow x_4) \vee (\neg x_4 \Rightarrow x_1)) \\ \wedge ((\neg x_4 \Rightarrow \neg x_2) \vee (x_2 \Rightarrow x_4)) \wedge ((x_1 \Rightarrow \neg x_3) \vee (x_3 \Rightarrow \neg x_1))$$

Qui nous donne le graphe suivant :



Le graphe est connexe, et en particulier, on observe les composantes fortement connexes suivantes :



— La composante rouge contient $\neg x_1$ et x_3, il n'y a pas d'équivalence absurde.
— La composante verte contient $\neg x_2$ et $\neg x_4$, il n'y a pas d'équivalence absurde.
— La composante bleue contient x_2 et x_4, il n'y a pas d'équivalence absurde.

— La composante jaune contient x_1 et $\neg x_3$, il n'y a pas d'équivalence absurde.

Il n'y a aucune composante fortement connexe qui contient une équivalence absurde, donc la formule est satisfiable. En particulier, l'affectation suivante rend la formule satisfiable : $x_1 = \top$, $x_2 = \top$, $x_3 = \bot$, $x_4 = \top$.

3. Si l'existence d'une solution pour 2-SATISFAISABILITÉ est négative, le problème d'optimisation à étudier est le problème du nombre maximum de clauses satisfiables (ou MAX-SAT).

Exercice 38 - Algorithme non-déterministe pour le problème de 3-COLORATION

Proposer un algorithme non-déterministe linéaire pour le problème de 3-COLORATION.

On suppose que le graphe $G = (V, E)$ est coloriable, et que la procédure automatique DEVINECOULEUR(u) renvoie la bonne couleur pour le sommet u du graphe.

Algorithm 6: Algorithme non-déterministe pour la 3-COLORATION

```
1 begin
2     forall u ∈ V do
3         Couleur(u) := DevineCouleur(u)
4     forall (u, v) ∈ E do
5         if Couleur(u) = Couleur(v) then
6             return False
7     return True
```

Cet algorithme est bien linéaire : il est de complexité $O(max(n, m))$.

Exercice 39 - Classification dans $\mathcal{NP}$ ou dans $\mathcal{P}$

Classer les problèmes suivants en fonction de $\mathcal{P}$ et $\mathcal{NP}$:

PROBLÈME P1

Entrée : $G = (V, E)$ un graphe non orienté.

Question : Existe t-il un cycle de longueur égale à $\left\lfloor \frac{|V|}{2} \right\rfloor$?

PROBLÈME P2

Entrée : $G = (V, E)$ un graphe non orienté.

Question : Existe t-il un cycle de longueur égale à 4 ?

PROBLÈME P3

Entrée : $G = (V, E)$ un graphe non orienté.

Question : Existe t-il un chemin simple entre u et v de longueur inférieure ou égale à k ?

PROBLÈME P4

Entrée : $G = (V, E)$ un graphe non orienté et un entier k.

Question : Existe t-il un arbre couvrant tous les sommets de G ayant moins de k feuilles ?

Pour prouver qu'un problème est dans $\mathcal{NP}$, il suffit d'exhiber un certificat positif polynomial (c'est à dire qu'on peut vérifier la solution donnée en temps polynomial).

S'il est dans $\mathcal{P}$, alors il existe un algorithme connu qui résoud le problème en temps polynomial.

Problème P1 La vérification de ce problème peut être fait en temps polynomial. Il suffit de vérifier, grâce au graphe et à la solution, que c'est bien un cycle et que la taille de la solution est $\left\lfloor \frac{|V|}{2} \right\rfloor$.

Cependant, ce problème n'est pas dans $\mathcal{P}$. L'algorithme intuitif serait de faire tous les chemins possibles de taille $\left\lfloor \frac{|V|}{2} \right\rfloor$ et de détecter si l'un deux est un cycle.

De plus, on remarque que le problème est équivalent au problème du cycle Hamiltonien (pour réduire celui-ci en instance de P1, il suffit de doubler le nombre de sommets sans inclure de cycle), il est donc même $\mathcal{NP}$-complet.

Problème P2 Ce problème est dans $\mathcal{P}$. On peut exhiber une procédure automatique qui calcule le résultat attendu en temps polynomial :

Algorithm 7: Cycle de longueur égale à 4 dans un graphe non orienté

```
1 begin
2     for i = 0 à n do
3         for j = 0 à n do
4             for x = 0 à n do
5                 for y = 0 à n do
6                     if {(i,j),(j,x),(x,y),(y,i)} ⊆ E then
7                         return True
8     return False
```

Problème P3 Ce problème est dans $\mathcal{P}$. En effet, il suffit de lancer un algorithme de plus court chemin (Dijkstra, en $O(|E| + |V|log(|V|))$, s'il n'y a pas d'arête à poids négatifs, Bellman-Ford, en $O(|V||E|)$, sinon).

Problème P4 Le problème de la chaîne Hamiltonienne, qui est $\mathcal{NP}$-complet, se réduit à P4. En effet, pour réduire une instance du problème de chaîne Hamiltonienne en P4, il suffit de savoir s'il y a un arbre couvrant tous les sommets de G avec une seule feuille. Ce problème est donc $\mathcal{NP}$-dur, et comme la vérification de son certificat se fait en temps polynomial, le problème est $\mathcal{NP}$-complet.

2.3 Réduction polynomiale

Exercice 40 - Réduction de Karp : une vision algorithmique

Supposons que le problème A est NP-complet. Soit α une instance du problème A. π' est une réduction polynomiale de A vers B, plus formellement, $O(|\alpha|^{c_1}) \Rightarrow |\pi'(\alpha)| \in O(|\alpha|^{c_1})$. $\pi'(\alpha)$ est ainsi une instance du problème B. De plus, $\forall\beta$ instance de B, il y a une solution polynomiale avec l'algorithme P_B.
Quelle est la complexité de l'algorithme A. Déduire la classe du problème A.

Comme toute instance du problème B se résout en temps polynomial, et qu'il existe une réduction en temps polynomial pour toute instance du problème A vers le problème B, A est polynomial. Comme $\mathcal{P} \neq \mathcal{NP}$, alors il y a une contradiction : A n'est pas NP-complet, mais NP-dur (donc B est aussi NP-dur).

Exercice 41 - Concept de la réduction (suite)

Pour réduire un problème A à un problème B, il suffit de montrer que la résolution de B permet de résoudre A à condition qu'une solution à B soit disponible.
Pour illustrer, supposons que A est le problème suivant : $A(n) =$ le plus petit nombre premier plus grand que n, et B le problème de décision $B = \{n | n \text{ is prime}\}$.

1. Donner pour quelques valeurs de n la valeur $A(n)$.
2. Proposer une réduction du problème A au problème B.

1. $A(5) = A(6) = 7$ ou bien $A(11) = 13$ ou encore $A(29) = 31$.
2. Supposons que la procédure automatique P_B calcule B. La réduction suivante transforme toute instance du problème A en problème B :

```
int P_A(int n) {
    n = n + 1;
    while (!P_B(n)) ++n;
    return n;
}
```

De manière équivalente, supposons que P_A est la procédure automatique qui calcule A. La réduction suivante transforme toute instance du problème B en problème A :

```
bool P_B(int n) {
    if (n == 0 || n == 1) return false;
    return P_A(n - 1) == n;
}
```

Ces deux problèmes se réduisent l'un à l'autre, ils sont donc de la même classe de complexité. Cependant, on n'est pas sûrs que A se réduise à B en temps polynomial : rien ne nous assure qu'il n'y a pas 2^n entiers entre le n et $n+1$-ème nombre premier.

Exercice 42 - Réduction

Montrer que les deux problèmes Problème du carré d'un entier et Problème de la multiplication se réduisent l'un à l'autre. L'addition, la soustraction et la division sont des opérations autorisées.
Problème de la multiplication

Entrée : Soient $a \in \mathbb{N}$ et $b \in \mathbb{N}$.

Question : Peut-on multiplier a et b ?

Problème du carré d'un entier

Entrée : Soient $a \in \mathbb{N}$.

Question : Peut-on élever a au carré ?

On suppose qu'on peut élever au carré un entier. On peut réduire le Problème de la multiplication en Problème du carré d'un entier en remarquant que :

$$(a+b)^2 = a^2 + 2ab + b^2 \Rightarrow ab = \frac{(a+b)^2 - a^2 - b^2}{2}$$

De manière équivalente, on peut réduire le Problème du carré d'un entier en Problème de la multiplication en multipliant le nombre donné par lui-même.

Exercice 43 - Réduction entre deux problèmes polynomiaux

2-Satisfaisabilité

Entrée : Étant donné une formule conjonctive ϕ sur n variables et m clauses chacune de taille deux.

Question : Existe-t-il une affectation de vérité aux variables qui satisfasse ϕ ?

2-Coloration

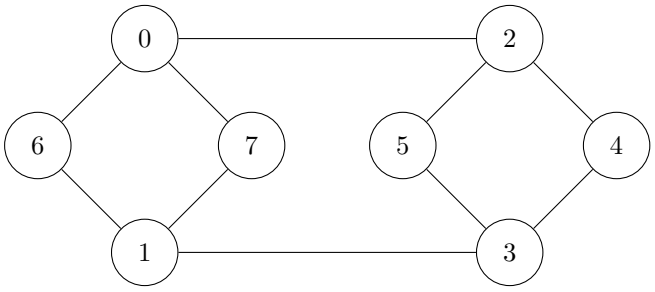
Entrée : Soit $G = (V, E)$ et deux couleur.

Question : Existe-t-il une 2-coloration valide, i.e. une fonction totale $f : \mathbb{N} \to \{1, 2\}$ telle que $f(u) \neq f(v), \forall (u, v) \in E$?

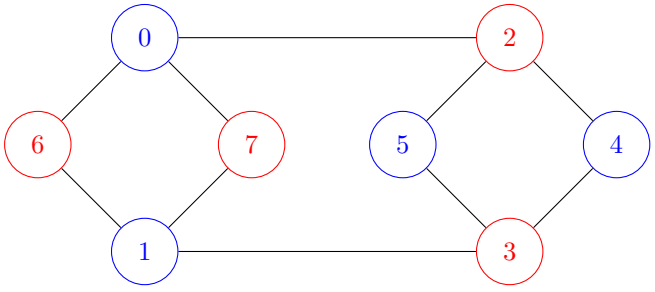
1. Montrer qu'il existe une réduction polynomiale entre 2-Coloration et 2-Satisfaisabilité.
2. Conclure sur la complexité du problème 2-Coloration.

Pour connaître la classe de complexité maximum de 2-Coloration, sachant que 2-Satisfaisabilité est dans $\mathcal{P}$, il faut trouver une réduction polynomiale entre 2-Coloration et 2-Satisfaisabilité pour prouver que 2-Coloration est aussi dans $\mathcal{P}$:

1. La donnée de 2-Coloration est un graphe $G = (V, E)$. Savoir si un graphe est bi-coloriable est exactement équivalent à se demander si, pour chaque arête $(u, v) \in E$, les deux sommets peuvent avoir une valeur de vérité différente ($\alpha(u) \neq \alpha(v)$).
   La réduction est alors assez évidente : il suffit, pour chaque arête, de faire un xor. On se retrouve donc avec la formule φ suivante :
   $$\varphi = \bigwedge_{(u,v)\in E} (u \vee v) \wedge (\overline{u} \vee \overline{v})$$
   Prenons en exemple un graphe bi-coloriable :



   On peut, par exemple, le colorier de la manière suivante avec les couleurs rouge et bleu, c'est bien un graphe bi-coloriable :



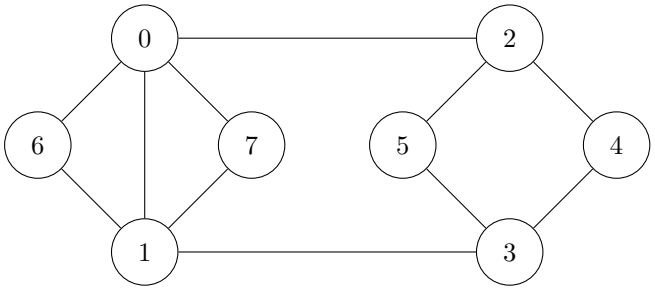
Ce graphe est transformé, grâce à notre réduction, en formule φ suivante :

$$\begin{aligned}\varphi = (x_0 \vee x_2) \wedge (\neg x_0 \vee \neg x_2) \wedge (x_0 \vee x_6) \wedge (\neg x_0 \vee \neg x_6) \wedge (x_0 \vee x_7) \wedge (\neg x_0 \wedge \neg x_7) \\ \wedge (x_1 \vee x_6) \wedge (\neg x_1 \vee \neg x_6) \wedge (x_1 \vee x_7) \wedge (\neg x_1 \vee \neg x_7) \wedge (x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \\ \wedge (x_2 \vee x_5) \wedge (\neg x_2 \vee \neg x_5) \wedge (x_2 \vee x_4) \wedge (\neg x_2 \vee \neg x_4) \\ \wedge (x_3 \vee x_5) \wedge (\neg x_3 \vee \neg x_5) \wedge (x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4)\end{aligned}$$

Cette formule est satisfiable : on prend $\alpha(x_0) = \alpha(x_1) = \alpha(x_4) = \alpha(x_5) = \top$ et $\alpha(x_2) = \alpha(x_3) = \alpha(x_6) = \alpha(x_7) = \bot$, ce qui nous donne :

$$\begin{aligned}\varphi = (\top \vee \bot) \wedge (\bot \vee \top) \wedge (\top \vee \bot) \wedge (\bot \vee \top) \wedge (\top \vee \bot) \wedge (\bot \vee \top) \\ \wedge (\top \vee \bot) \wedge (\bot \vee \top) \wedge (\top \vee \bot) \wedge (\bot \vee \top) \wedge (\top \vee \bot) \wedge (\bot \vee \top) \\ \wedge (\bot \vee \top) \wedge (\top \vee \bot) \wedge (\bot \vee \top) \wedge (\top \vee \bot) \\ \wedge (\bot \vee \top) \wedge (\top \vee \bot) \wedge (\bot \vee \top) \wedge (\top \vee \bot)\end{aligned}$$

Chaque clause comporte au moins une valeur de vérité à $\top$, on a donc bien $\alpha(\varphi) = \top$.
Seulement, si on ajoute une arête entre le sommet 0 et 1, qui nous donne le graphe suivant :



et qu'on le transforme en instance de 2-Satisfaisabilité :

$$\varphi' = \varphi \wedge (x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$$

L'affectation α ne satisfait pas cette formule. En effet, on a $\alpha(\neg x_0 \vee \neg x_1) = \alpha(\bot \vee \bot) = \bot$, et ne peut pas trouver d'affectation qui satisfasse φ', le graphe n'est donc pas 2-coloriables !
Pour reconstruire la 2-Coloration depuis l'instance de 2-Satisfaisabilité, il suffit de prendre la valeur de vérité de chaque variable, et d'affecter une couleur. Par exemple : $\top$ est bleu et $\bot$ est rouge : $\forall u \in V.couleur(u) = \alpha(u)$. On peut ainsi retrouver le graphe bi-colorié que j'ai donné un peu plus haut en exemple.

2. Comme 2-Satisfaisabilité se réduit en 2-Coloration, 2-Coloration est plus dur que 2-Satisfaisabilité. De plus, 2-Coloration est dans $\mathcal{P}$ (en effet, l'algorithme de 2-Coloration est assez simple, l'idée est de prendre un sommet arbitraire, de le colorier d'une couleur, et de colorier ses voisins d'une autre couleur. Lorsque tous les sommets sont coloriés de cette manière, il suffit de regarder si tous les sommets adjacents ont une couleur différente, ou non), donc 2-Satisfaisabilité est aussi dans $\mathcal{P}$.

Exercice 44 - Problèmes équivalents polynomialement

Pour les problèmes suivants, indiquez si les problèmes sont polynomialement équivalents.

1. Arbre couvrant de poids minimum et Arbre couvrant de poids maximum

   Arbre couvrant de poids minimum

   Entrée : Un graphe $G = (V, E)$ et une valuation sur les arêtes

   Question : Trouver un sous-graphe connexe de poids minimum

   Arbre couvrant de poids maximum

   Entrée : Un graphe $G = (V, E)$ et une valuation sur les arêtes

   Question : Trouver un sous-graphe connexe de poids maximum

2. Arborescence des plus courts chemins et Plus long Chemin

   Arborescence des plus courts chemins

   Entrée : Un graphe $G = (V, E)$ orienté

   Question : Trouver une arborescence des plus court chemins

   Plus long Chemin

   Entrée : Un graphe $G = (V, E)$ un graphe orienté

   Question : Trouver un plus long chemin sans répétition de sommets

3. Coupe de valeur maximale et Coupe de valeur minimale

   Coupe de valeur maximale

   Entrée : Un graphe $G = (V, E)$ orienté

   Question : Trouver une coupe de valeur maximale

   Coupe de valeur minimale

   Entrée : Un graphe $G = (V, E)$ un graphe orienté

   Question : Trouver une coupe de valeur minimale

4. Couplage maximum de valeur maximum et Couplage minimum de valeur minimum

   Couplage maximum de valeur maximum

   Entrée : Un graphe $G = (V, E)$, et une valuation sur les arêtes

   Question : Trouver un couplage maximum de poids maximum

   Couplage maximum de valeur minimum

   Entrée : Un graphe $G = (V, E)$, et une valuation sur les arêtes

   Question : Trouver un couplage maximum de poids maximum

5. Voyageur de Commerce de coût minimum et Voyageur de Commerce de coût maximum

   Voyageur de Commerce de coût minimum

   Entrée : Un graphe $G = (V, E)$, et une valuation sur les arêtes

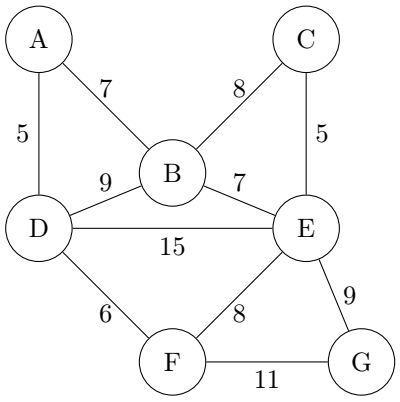
   Question : Trouver un chemin Hamiltonien de poids minimum

   Voyageur de Commerce de coût maximum

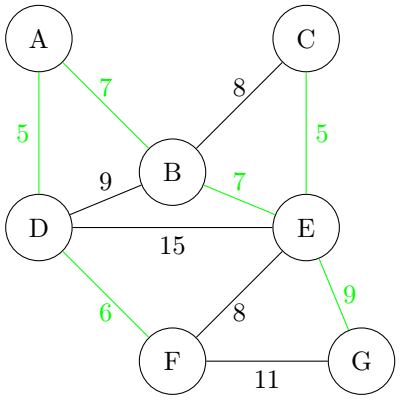
   Entrée : Un graphe $G = (V, E)$, et une valuation sur les arêtes

   Question : Trouver un chemin Hamiltonien de poids maximum

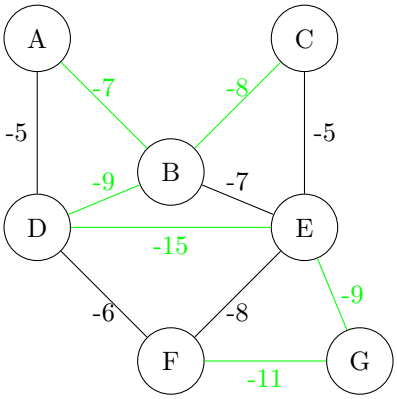
1. Pour commencer, on peut classifier le problème de l'Arbre couvrant de poids minimum dans $\mathcal{P}$. En effet, l'algorithme de Kruskal [1] s'exécute en temps polynomial ($O(Elog(V))$). Pour illustrer le problème, prenons le graphe G suivant :



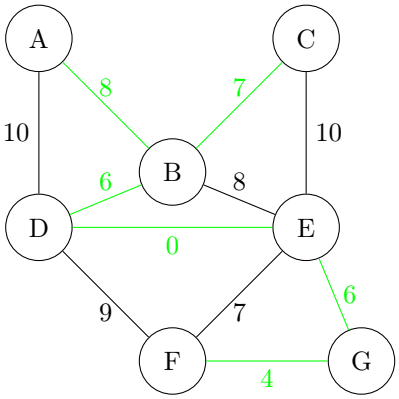
On peut exhiber son arbre couvrant minimum, en vert sur cette figure :



La somme des valuations de l'arbre couvrant minimum (ABEGDF) est 39, on ne peut pas faire mieux ici. On peut réduire le problème de l'arbre couvrant minimum de manière polynomiale : il suffit de multiplier par -1 toutes les valuations. On peut aussi changer la valuation de chaque arête en faisant une différence avec la valuation maximale, les deux solutions marchent pour réduire le problème de l'arbre couvrant de poids maximum en arbre couvrant de poids minimum :



Valuations inversées



Différence avec la valuation max

1. voir `https://en.wikipedia.org/wiki/Kruskal's_algorithm`

2. Les deux problèmes ne sont pas polynomialement équivalents, l'arborescence des plus courts chemins est $\mathcal{P}$ (avec l'algorithme de Floyd-Warshall [2] par exemple), alors que le plus long chemin sans répétition de sommets est NP-complet (car le cycle Hamiltonien se réduit au problème du plus long chemin sans répétition).
3. Les deux problèmes ne sont pas polynomialement équivalents, la coupe minimale est dans $\mathcal{P}$ (avec la correspondance MinCut-MaxFlow et l'algorithme d'Edmond-Karp [3]), alors que la coupe de valeur maximale est NP-complet (car Max2Sat se réduit au problème de MaxCut).
4. En utilisant la même réduction que pour l'arbre couvrant de poids minimum, le couplage maximum de poids maximum (Max-Matching) et le couplage maximum de poids minimum (Min-Matching) sont polynomialement équivalents.
5. En utilisant la même réduction que pour l'arbre couvrant de poids minimum, le voyageur de commerce de coût minimum et le voyageur de commerce de coût maximum sont polynomialement équivalents (ils sont cependant tous les deux NP-difficiles, car le problème du chemin Hamiltonien se réduit au problème du voyageur de commerce, et il n'existe pas de certificat polynomial pour le problème d'optimisation).

Exercice 45 - Problème de décision

Mettre les problèmes suivants sous forme de problème de décision et évaluer la taille de leurs instances.

1. de savoir s'il existe un chemin entre deux sommets disjoints dans un graphe ;
2. de connaître la distance entre deux sommets disjoints dans un graphe ;
3. de connaître la longueur de la chaîne maximum dans un graphe pondéré.

— Existence d'un chemin

Entrée : $G = (V, E)$ un graphe non orienté, $u, v \in V$ deux sommets du graphe

Question : Y-a-t'il un chemin qui relie u et v dans G ?

La taille de l'instance est de $O(|V|^2)$ si le graphe est représenté comme une matrice, $O(|V|+|E|)$ sinon.

— Longueur du chemin

Entrée : $G = (V, E)$ un graphe non orienté, $u, v \in V$ deux sommets du graphe, k une distance

Question : Y-a-t'il un chemin qui relie u et v dans G de longueur k ?

La taille de l'instance est de $O(|V|^2)$ si le graphe est représenté comme une matrice, $O(|V| + |E|)$ sinon. La taille de l'entier $(O(log(k)))$ n'influence pas la taille totale de l'instance, car $O(log(k)) \ll O(|V| + |E|)$.

— Longueur du chemin maximum

Entrée : $G = (V, E)$ un graphe non orienté, $u, v \in V$ deux sommets du graphe, k une distance, et w une valuation des arêtes

Question : Y-a-t'il un chemin maximum qui relie u et v dans G de longueur $\geq k$?

La taille de l'instance est de $O(|E|k) + O(|V|^2)$ si le graphe est représenté par une matrice, $O(|V| + |E|(1 + k))$ sinon.

2. Floyd-Warshall : `https://en.wikipedia.org/wiki/Floyd\T1\textendashWarshall_algorithm`
3. Edmond-Karp : `https://en.wikipedia.org/wiki/Edmonds\T1\textendashKarp_algorithm`

Exercice 46 - Optimisation versus décision

Soit le problème du stable de taille k :

STABLE

Entrée : Un graphe orienté $G = (V, E)$, une distance k.

Question : Existe-t-il un stable (c'est à dire un sous-ensemble de sommets tel que deux sommets de ce sous-ensemble ne soient jamais reliés par une arête) de taille k

et sa version optimisation :

MAX-STABLE

Entrée : Un graphe orienté $G = (V, E)$.

Question : Trouver un stable (c'est à dire un sous-ensemble de sommets tel que deux sommets de ce sous-ensemble ne soient jamais reliés par une arête) de taille maximum.

1. Montrer que s'il existe un algorithme polynomial qui résout le problème de stabilité maximum alors la version décisionnelle est résoluble, elle aussi, en temps polynomial.
2. Montrer que s'il existe un algorithme qui résout le problème de stable de taille k en temps polynomial alors le problème de stabilité maximum est résoluble, lui aussi, en temps polynomial.

1. Pour prouver que la version décisionnelle du problème peut être résolue par la version optimisation, il suffit de trouver une réduction polynomiale de la version décisionnelle à la version optimale. Si MAX-STABLE trouve une solution de taille $\geq k$, c'est qu'une solution de taille k existe. Ainsi, si on pose $P_{MS}(G)$ la procédure automatique qui calcule MAX-STABLE de G, la réduction est la suivante :

```
bool P_S(G, int k) {
    return |P_MS(G)| >= k;
}
```

   Comme cette réduction est constante, on peut affirmer que s'il existe un algorithme polynomial qui résout MAX-STABLE, alors la version décisionnelle est aussi résolue en temps polynomial.

2. Pour prouver que la version optimisation du problème peut être résolue par la version décisionnelle, il suffit de trouver une réduction polynomiale de la version optimisation à la version décisionnelle. Cette réduction se résume à partir de $k = 0$, et de l'augmenter tant qu'on trouve un STABLE de cette taille dans G avec la procédure automatique P_S qui calcule le problème décisionnel :

```
int P_MS(G) {
    int k = 0;
    while (P_S(G, k)) k += 1;
    return k - 1;
}
```

   Comme cette réduction est linéaire (en nombre de sommets), on peut affirmer que s'il existe un algorithme polynomial qui résout STABLE, alors MAX-STABLE est aussi résolue en temps polynomial.

2.4 Autour des classes $\mathcal{NP}$ et $\mathcal{NP}$-complet

Exercice 47 - Certificats polynomiaux et réductions polynomiales

1. Quels sont les certificats des problèmes de décision suivants ? Sont-ils des certificats polynomiaux ?
   (a) 2-Partition
   (b) Circuit Hamiltonien
   (c) Satisfaisabilité
   (d) Clique
   (e) 3-Coloration
2. Même question pour les co-problèmes (la question est formulée de façon négative).

1. (a) Rappelons le problème de 2-Partition :

   2-Partition

   Entrée : Un multiensemble S de n nombres entiers positifs.

   Question : Existe-t-il deux sous-ensembles S_1 et S_2 tels que $S_1 \cap S_2 = \emptyset$, $S_1 \cup S_2 = S$ et $\sum_{s_1 \in S_1} s_1 = \sum_{s_2 \in S_2} s_2$?

   Le certificat de ce problème sont les ensembles S_1 et S_2. Il est de taille polynomiale ($O(|S_1|+|S_2|) = O(|S|)$) et vérifiable en temps polynomial :

   Algorithm 8: $\pi_{\text{2-Partition}}(S_1, S_2)$

   ```
   begin
       if |S1| ≠ |S2| then
           return non
       if Σ_{s1∈S1} s1 ≠ Σ_{s2∈S2} s2 then
           return non
       return oui
   ```

   (b) Rappelons le problème du Circuit Hamiltonien :

   Circuit Hamiltonien

   Entrée : $G = (V, E)$ un graphe non orienté.

   Question : Existe-t-il un cycle qui passe par tous les sommets du graphe une et une seule fois ?

   Le certificat de ce problème est la liste de sommets par lequel le cycle passe. Il est de taille polynomiale ($O(|V|)$) et vérifiable en temps polynomial ($O(|V| \times |C| + |E| \times |C|)$) :

Algorithm 9: $\pi_{\text{Circuit\_Hamiltonien}}(G = (V, E), C)$

```
1  begin
2      if C_[0] ≠ C_[-1] ou (C_[0], C_[0]) ∉ E then
3          return non
4      seen := ∅
5      for i = 0 à |C| − 1 do
6          if (C_[i], C_[i+1]) ∉ E then
7              return non
8          seen := seen ∪ {C_[i], C_[i+1]}
9      forall v ∈ V do
10         if v ∉ seen then
11             return non
12     return oui
```

(c) La Satisfaisabilité est le problème par excellence en informatique. C'est le premier problème qui a été prouvé $\mathcal{NP}$-complet, c'est à dire que tous les problèmes se réduisent en temps polynomial en Satisfaisabilité. On sait donc que ce problème possède un certificat polynomial. Cependant, nous pouvons faire en sorte de l'exhiber. Le certificat est une affectation α des variables de ϕ ($\alpha(x_i) = \top$ ou $\alpha(x_i) = \bot$, et $\alpha(\neg x_i) = \neg(\alpha(x_i))$). Ce certificat est en espace polynomial (nombre de littéraux de la formule ϕ), et la vérification se fait en temps polynomial (toujours en $O(|\phi|)$) :

Algorithm 10: $\pi_{\text{Satisfaisabilité}}(\phi, \alpha)$

```
1  begin
2      forall C ∈ φ do
3          found := ⊥
4          forall c_i ∈ C do
5              if α(c_i) = ⊤ then
6                  found := ⊤
7          if ¬found then
8              return non
9      return oui
```

(d) Rappelons le problème de Clique :

Clique

Entrée : Un graphe $G = (V, E)$

Question : Existe-t-il une clique, c'est à dire un sous-graphe complet, dans G ?

Le certificat de Clique est polynomial. C'est l'ensemble des sommets qui composent la clique. La taille du certificat en espace est au pire des cas $O(|V|)$, et la vérification se fait en temps polynomial ($O(|S| \times |E|)$ avec S la liste des sommets qui composent la clique) :

Algorithm 11: $\pi_{\text{Clique}}(G, S)$

```
1  begin
2      forall s ∈ S do
3          forall (u, v) ∈ E do
4              if u est s then
5                  marquer v
6              else if v est s then
7                  marquer u
8          if ∃v ∈ S tel que v n'est pas marqué then
9              return non
10     return oui
```

(e) Rappelons le problème de 3-Coloration :

3-Coloration

Entrée : Un graphe $G = (V, E)$ non orienté

Question : G est-il 3-coloriable ?

Dans ce cas, 3-coloriable signifie qu'en prenant trois couleur, on peut colorier chaque sommet d'une couleur différente de ses voisins.

Le certificat est le coloriage C du graphe. Il est bien polynomial ($O(|V|)$) en espace, et il est polynomial ($O(|E|)$) en temps :

Algorithm 12: $\pi_{\text{3-Coloration}}(G = (V, E), C)$

```
1  begin
2      forall (u, v) ∈ E do
3          if C[u] = C[v] then
4              return non
5      return oui
```

2. (a) Pour le co-problème de 2-Partition, non, le certificat n'est pas polynomial. En effet, il faudrait appeler l'algorithme de 2-Partition pour savoir si, oui ou non, il n'existe pas de partitions.

   (b) Pour le co-problème du Circuit Hamiltonien, c'est la même chose. Pour prouver qu'il n'en existe pas, il faudrait explorer toutes les options, il n'y a donc pas de certificat polynomial.

   (c) Pour le co-problème de Satisfaisabilité, c'est la même chose. Il faudrait tester toutes les affectations pour avoir le certificat qu'il n'existe pas d'affectation qui satisfasse la formule, donc le certificat ne serait pas polynomial.

   (d) Pour le co-problème de la Clique, il faudrait aussi tester la combinaison de tous les sommets, ce qui n'est pas polynomial.

   (e) Pour le co-problème de la 3-Coloration, il faudrait énumérer tous les cas, ce qui n'est pas polynomial non plus.

Dans cet exercice d'introduction aux certificats, j'ai donné le certificat polynomial, et exhibé l'algorithme de vérification. Cependant, il n'est pas nécessaire d'exhiber l'algorithme de vérification quand un certificat est demandé, il suffit de donner ce certificat et de prouver qu'il est polynomial. C'est ce qui sera fait dans les exercices suivants.

Exercice 48 - Certificat positif

Pour les problèmes suivants, donner le certificat positif.

1. 3-Satisfaisabilité
2. k-Coloration
3. Sous-somme Maximale
4. 2-Partition
5. Circuit Hamiltonien
6. Satisfaisabilité
7. Clique
8. 3-Coloration
9. Soit le problème Chemin $= \{< G, s, t > \mid G$ est un graphe orienté possédant un chemin de s à $t\}$

1. Ce certificat a été donné au (c) de l'exercice précédent.
2. Ce certificat est le même que la 3-Coloration du (e) de l'exercice précédent.
3. Rappelons le problème de Sous-Somme Maximale :

   Sous-Somme Maximale

   Entrée : Un tableau T et un entier k.

   Question : Existe-t-il une suite d'éléments contigus S du tableau T tels que $\sum_{s \in S} s > k$?

   Le certificat positif de ce problème sont les indices de début et de fin de cette suite d'éléments contigus, pour vérifier qu'ils sont bien contigus et que leur somme est supérieure à k.
4. Ce certificat a été donné au (a) de l'exercice précédent.
5. Ce certificat a été donné au (b) de l'exercice précédent.
6. Ce certificat a été donné au (c) de l'exercice précédent.
7. Ce certificat a été donné au (d) de l'exercice précédent.
8. Ce certificat a été donné au (e) de l'exercice précédent.
9. Le certificat positif est la liste des arêtes à emprunter entre s et t. La vérification est facile : il suffit de regarder si l'arête est bien dans E et que la suite d'arête forme bien un chemin de s à t.

2.5 Propriétés des classes $\mathcal{NP}$ et $\mathcal{NP}$-complet

Exercice 49 - Propriétés des classes $\mathcal{NP}$ et $\mathcal{NP}$-complet

Soient A et B deux langages. Prouver ou réfuter les deux assertions suivantes :

1. Si A et B sont dans $\mathcal{NP}$, alors on a $A \cup B \in \mathcal{NP}$ et $A \cap B \in \mathcal{NP}$
2. Si A et B sont $\mathcal{NP}$-complet, alors ni $A \cup B$ ni $A \cap B$ peuvent être $\mathcal{NP}$-complets.

1. A est dans $\mathcal{NP}$, donc $\forall x \in A, \exists y \in \{0,1\}^*.|y| \leq p(|x|) \mid C_A(x,y) = 1$. C'est à dire que $C_A(x,y) = 1$ si le certificat y vérifie que $x \in A$ en temps polynomial. On a la même définition pour B : $\forall x \in B, \exists y \in \{0,1\}^*.|y| \leq p(|x|) \mid C_B(x,y) = 1$.
   Pour que $A \cup B$ soit dans $\mathcal{NP}$, il suffit de trouver un algorithme polynomial qui vérifie si x est dans $A \cup B$. Il suffit de prendre l'algorithme $C_{A \cup B}(x,y) = C_A(x,y) \vee C_B(x,y)$. Comme C_A s'exécute en temps polynomial $(O(y^{p(|x|)}))$ et C_B s'exécute en temps polynomial $(O(y^{p(|x|)}))$, $C_{A \cup B}$ s'effectue aussi en temps polynomial $(O(2y^{p(|x|)}))$.

On peut définir $C_{A\cap B}$ de la même manière, avec $C_{A\cap B}(x,y) = C_A(x,y) \wedge C_B(x,y)$, qui possède la même complexité que $C_{A\cup B}$.

2. Prenons un langage $\mathcal{NP}$-complet L. Posons :

$$A = \{0l \mid l \in L\}$$
$$B = \{1l \mid l \in L\}$$

A et B sont aussi $\mathcal{NP}$-complets (car un langage $\mathcal{NP}$-complet se réduit à A en concaténant un 0 devant et B en concaténant un 1 devant). Cependant, $A \cap B = \emptyset$, qui n'est pas $\mathcal{NP}$-complet.
On veut maintenant montrer que l'union de deux langages $\mathcal{NP}$-complets n'est pas nécessairement $\mathcal{NP}$-complet. Posons les langages A' et B' suivants :

$$A' = A \cup \{1x \mid x \in \Sigma^*\}$$
$$B' = B \cup \{0x \mid x \in \Sigma^*\}$$

A' et B' sont bien $\mathcal{NP}$-complets. En effet, L se réduit à A' et à B' en temps polynomial. Cependant, on a $A' \cup B' = \Sigma^*$, qui n'est pas $\mathcal{NP}$-complet. On a donc $A' \cup B' \notin \mathcal{NPC}$.

2.6 Classes *co*$\mathcal{NP}$ et *co*$\mathcal{NP}$-complet

Exercice 50 - Définition de *co*$\mathcal{NP}$ par les langages formels

Un langage A est dans *co*$\mathcal{NP}$ si et seulement s'il existe un polynôme $p(n)$ et un langage $B \in \mathcal{P}$ tels que

$$x \in A \Leftrightarrow \forall y \in \{0,1\}^{p(|x|)}(x,y) \in B$$

Si A est dans *co*$\mathcal{NP}$, alors $\bar{A}$ est dans $\mathcal{NP}$. Soit p un polynôme et $C_{\bar{A}}$ l'algorithme qui vérifie le langage $\bar{A}$ en temps polynomial, on a :

$$\forall x \in \bar{A}, \exists y \in \{0,1\}^*.|y| \leq p(|x|) \mid C_{\bar{A}}(x,y) = 1$$

Comme y est un mot binaire, si $|y| \leq p(|x|)$, alors $y \in \{0,1\}^{p(|x|)}$. De plus, si $\forall x \in \bar{A}$ on peut trouver un y qui vérifie la proposition, on peut dire que $x \in \bar{A}$ est équivalent à trouver ce y. On peut alors réécrire la définition précédente, avec une dernière étape qui correspond simplement à une fonction caractéristique :

$$\begin{aligned}\forall x \in \bar{A}, \exists y \in \{0,1\}^*.|y| \leq p(|x|) \mid C_{\bar{A}}(x,y) = 1 &\equiv \forall x \in \bar{A}, \exists y \in \{0,1\}^{p(|x|)}.C_{\bar{A}}(x,y) = 1 \\ &\equiv x \in \bar{A} \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)}.C_{\bar{A}}(x,y) = 1 \\ &\equiv x \in \bar{A} \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)}.(x,y) \in C_{\bar{A}}\end{aligned}$$

On veut connaître la définition de *co*$\mathcal{NP}$. Ainsi, on nie la formule transformée (car on sait qu'un problème est dans *co*$\mathcal{NP}$ si son complémentaire est dans $\mathcal{NP}$) :

$$\begin{aligned}\neg(x \in \bar{A} \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)}.(x,y) \in C_{\bar{A}}) &\equiv \neg(x \in \bar{A}) \Leftrightarrow \neg(\exists y \in \{0,1\}^{p(|x|)}.(x,y) \in C_{\bar{A}}) \\ &\equiv x \in A \Leftrightarrow \neg(\exists y \in \{0,1\}^{p(|x|)}.(x,y) \in C_{\bar{A}}) && \text{car } x \notin \bar{A} \Leftrightarrow x \in A \\ &\equiv x \in A \Leftrightarrow \forall y \in \{0,1\}^{p(|x|)}.(x,y) \notin C_{\bar{A}} && \text{car } \neg\forall \equiv \exists \\ &\equiv x \in A \Leftrightarrow \forall y \in \{0,1\}^{p(|x|)}.(x,y) \in C_A && \text{car } C_A \text{ vérifie négativement } \bar{A} \\ &\equiv x \in A \Leftrightarrow \forall y \in \{0,1\}^{p(|x|)}.(x,y) \in B && \text{si on pose } B = C_A\end{aligned}$$

On a bien vérifié que si A est dans *co*$\mathcal{NP}$, alors pour un polynôme p et un langage B, on a bien $x \in A \Leftrightarrow \forall y \in \{0,1\}^{p(|x|)}.(x,y) \in B$.

Exercice 51 - Propriétés pour $co\mathcal{NP}$

Supposons que $A \in \mathcal{NP}$ et que $B \in co\mathcal{NP}$. Nous supposerons pour cet exercice $\mathcal{NP} \neq co\mathcal{NP}$. Montrer la véracité ou trouver un contre-exemple aux assertions suivantes :

1. $\bar{A} \cup B \in co\mathcal{NP}$.
2. $A \cap \bar{B} \in co\mathcal{NP}$.
3. $A \cup B \in co\mathcal{NP}$.
4. $A \cap B \in co\mathcal{NP}$.

Pour savoir si un problème est dans $co\mathcal{NP}$, il faut montrer que le complémentaire de ce problème est dans $\mathcal{NP}$.

1. On veut montrer que $\overline{A} \cup B \in co\mathcal{NP}$:

$$\begin{aligned} \overline{A} \cup B \in co\mathcal{NP} &\Leftrightarrow \overline{\overline{A} \cup B} \in \mathcal{NP} \\ &\Leftrightarrow A \cap \overline{B} \in \mathcal{NP} \end{aligned}$$

   Comme $A \in \mathcal{NP}$ et $\overline{B} \in \mathcal{NP}$, on a bien (prouvé à l'exercice 49) $A \cap \overline{B} \in \mathcal{NP}$, donc $\overline{A} \cup B \in co\mathcal{NP}$.

2. Procédons de la même manière qu'à la question précédente :

$$\begin{aligned} A \cap \overline{B} \in co\mathcal{NP} &\Leftrightarrow \overline{A \cap \overline{B}} \in \mathcal{NP} \\ &\Leftrightarrow \overline{A} \cup B \in \mathcal{NP} \end{aligned}$$

   Or, on vient de prouver que $\overline{A} \cup B \in co\mathcal{NP}$, donc $\overline{A} \cup B \notin \mathcal{NP}$, mais on ne peut rien conclure sur l'appartenance à $co\mathcal{NP}$. Essayons de trouver un contre exemple : soit $B = \emptyset$ le langage vide. On a $\overline{B} = \Sigma^*$, donc $A \cap \overline{B} = A$. Si on prend $A \in \mathcal{NP} - co\mathcal{NP}$ (car $\mathcal{NP} \neq co\mathcal{NP}$), alors $A \cap \overline{B} = A \notin co\mathcal{NP}$.

3. On applique toujours la même méthode :

$$\begin{aligned} A \cup B \in co\mathcal{NP} &\Leftrightarrow \overline{A \cup B} \in \mathcal{NP} \\ &\Leftrightarrow \overline{A} \cap \overline{B} \in \mathcal{NP} \end{aligned}$$

   On ne peut pas directement conclure avec cette méthode. Essayons de trouver un contre-exemple : soit $\emptyset$ le langage vide. Posons $B = \emptyset$. On sait que $\emptyset \in \mathcal{NP}$, mais il est aussi dans $co\mathcal{NP}$, car le certificat positif et négatif du langage peut être trouvé en temps constant. On a donc $B \in co\mathcal{NP}$. Ainsi, si $B = \emptyset$ et A est un problème de $\mathcal{NP} - co\mathcal{NP}$, alors $A \cup B \in \mathcal{NP}$ donc $A \cup B \notin co\mathcal{NP}$.

4. On ne peut pas montrer que $A \cap B \in co\mathcal{NP}$ car on ne sait pas si $\overline{A} \cup \overline{B} \in \mathcal{NP}$. Essayons de trouver un contre-exemple. On prend toujours $A \in \mathcal{NP} - co\mathcal{NP}$. On prend $B = \Sigma^*$. Comme $\emptyset \in \mathcal{NP}$, on a Σ^* (qui est le complémentaire de $\emptyset$) dans $co\mathcal{NP}$. On a donc $A \cap B = A \in \mathcal{NP}$, donc $A \cap B \notin co\mathcal{NP}$.

Exercice 52 - Absence de certificat positif

Considérons le problème suivant :

Co-Voyageur de Commerce (COTSP)

Entrée : Un ensemble de m villes X, un ensemble de routes entres villes E. Une fonction de coût $v : E \to \mathbb{N}$ où $v(x, y)$ est le coût de déplacement de x à y, $k \in \mathbb{N}$.

Question : N'existe-t-il pas de cycle Hamiltonien de distance inférieure ou égale à k ?

Ce problème appartient-il à la classe $\mathcal{NP}$?

Intuitivement, on ne pense pas. Cependant, on n'a pas encore de preuve qu'il appartient, ou pas, à $\mathcal{NP}$, donc pour le moment, on ne sait pas.

Exercice 53 - Propriétés de $co\mathcal{NP}$

1. Montrer que si π est un problème $\mathcal{NP}$-complet tel que $\bar{\pi} \in \mathcal{NP}$ alors nous obtenons $co\mathcal{NP} = \mathcal{NP}$.
2. Montrer que si $co\mathcal{NP} \neq \mathcal{NP}$, alors $\mathcal{P} \neq \mathcal{NP}$.
3. Est-ce que nous pouvons avoir $\mathcal{P} \neq \mathcal{NP}$ et $co\mathcal{NP} = \mathcal{NP}$?

1. Comme $\bar{\pi} \in \mathcal{NP}$, alors on a $\pi \in co\mathcal{NP} \cap \mathcal{NP}$. De plus, $\pi \in \mathcal{NP}$-complet, donc tous les problèmes de $\mathcal{NP}$ se réduisent à π. On a donc $\mathcal{NP} \subseteq co\mathcal{NP} \cap \mathcal{NP}$. Réciproquement, on a $\bar{\pi} \in co\mathcal{NP} \cap \mathcal{NP}$ car $\bar{\pi} \in \mathcal{NP}$ et $\pi \in \mathcal{NP}$. De plus, $\bar{\pi} \in co\mathcal{NP}$-complet, donc tous les problèmes de $co\mathcal{NP}$ se réduisent à $\bar{\pi}$. Ainsi, $co\mathcal{NP} \subseteq co\mathcal{NP} \cap \mathcal{NP}$. Comme $\mathcal{NP} \subseteq co\mathcal{NP} \cap \mathcal{NP}$ et $co\mathcal{NP} \subseteq co\mathcal{NP} \cap \mathcal{NP}$, on a $co\mathcal{NP} = \mathcal{NP}$.
2. Prouvons la contraposée : si $\mathcal{P} = \mathcal{NP}$, alors $co\mathcal{NP} = \mathcal{NP}$. C'est trivial, car $\mathcal{P}$ étant fermée par complémentaire, on a $\mathcal{P} = co\mathcal{P}$. Ainsi, comme $\mathcal{P} = \mathcal{NP}$, on a $co\mathcal{P} = co\mathcal{NP}$, donc $\mathcal{P} = co\mathcal{P} \Rightarrow \mathcal{NP} = co\mathcal{P} \Rightarrow \mathcal{NP} = co\mathcal{NP}$.
3. On ne sait pas, mais on pense fortement que c'est le cas. Pour le prouver, il suffirait d'exhiber un problème dans $co\mathcal{NP} \cap \mathcal{NP}$ qui n'est pas dans $\mathcal{P}$, mais pour l'instant, il n'a pas été trouvé.

Exercice 54 - Problème $co\mathcal{NP}$-complet

De tout problème dans $\mathcal{NP}$, on peut construire un problème dual dans $co\mathcal{NP}$ de problèmes suivants :

1. Satisfaisabilité (sat)
   Entrée : Étant donné une formule booléenne sous forme normale conjonctive.
   Question : Existe-t-il une assignation de ses variables qui la rend vraie ?
2. Chemin Hamiltonien
   Entrée : Un graphe $G = (V, E)$.
   Question : Existe-t-il un chemin Hamiltonien ?
3. Clique
   Entrée : $G = (V, E)$ et $k \in \mathbb{N}^*$.
   Question : Existe-t-il une clique de taille k ?

1. Co-Satisfaisabilité
   Entrée : Étant donné une formule booléenne sous forme normale conjonctive.
   Question : N'existe-t-il pas d'assignation de ses variables qui la rend vraie ?
2. Co-Chemin Hamiltonien
   Entrée : Un graphe $G = (V, E)$.
   Question : N'existe-t-il pas de chemin Hamiltonien ?
3. Co-Clique
   Entrée : $G = (V, E)$ et $k \in \mathbb{N}^*$.
   Question : N'existe-t-il pas de clique de taille k ?

Exercice 55 - $co\mathcal{NP} \cap \mathcal{NP}$

Considérons le problème du Flot Maximum.

Flot Maximum (Max Flow)

Entrée : $G = (V, E, c)$, une source s et un puit t, et $K \in \mathbb{N}$.

Question : Est-il vrai que G possède un flot de valeur au moins K entre s et t ?

Montrer que le problème du Flot Maximum est dans $\mathcal{NP} \cap co\mathcal{NP}$.

La démonstration que ce problème est dans $\mathcal{NP} \cap co\mathcal{NP}$ se fait en deux temps : d'abord, on montre que ce problème est dans $\mathcal{NP}$, puis on montre que le complémentaire est aussi dans $\mathcal{NP}$:

— Ce problème est dans $\mathcal{NP}$. Le certificat est le chemin parcouru entre s et t, et il suffit de vérifier si la valeur du flot, donné par c est bien supérieur ou égal à K. C'est bien un certificat polynomial par rapport à la taille de l'entrée.

— Soit le problème complémentaire :

Co-Flot Maximum

Entrée : $G = (V, E, c)$, une source s et un puit t, $K \in \mathbb{N}$.

Question : G ne possède-t-il pas de flot de valeur supérieure ou égale à K entre s et t ?

Comme l'algorithme du Flot Maximum se fait en temps polynomial (Edmonds-Karp), on vérifie aussi que ce n'est pas possible en temps polynomial, ce problème est donc bien dans $\mathcal{NP}$.

Le problème Flot Maximum est dans $\mathcal{NP}$, et son co-problème est aussi dans $\mathcal{NP}$, donc Flot Maximum$\in \mathcal{NP} \cap co\mathcal{NP}$.

Exercice 56 - $co\mathcal{NP} \cap \mathcal{NP}$

Considérons le problème du Couplage Parfait.

Couplage Parfait (Perfect Matching)

Entrée : $G = (U \cup V, E)$.

Question : Est-il vrai que G possède un couplage parfait ?

Montrer que le problème du Couplage Parfait est dans $\mathcal{NP} \cap co\mathcal{NP}$.

La démonstration que ce problème est dans $\mathcal{NP} \cap co\mathcal{NP}$ se fait en deux temps : d'abord, on montre que ce problème est dans $\mathcal{NP}$, puis on montre que le complémentaire est aussi dans $\mathcal{NP}$:

— Ce problème est dans $\mathcal{NP}$. Le certificat est l'ensemble des arêtes. Il suffit de vérifier qu'un sommet est incident a une et une seule arête, ce qui se fait en temps polynomial.

— Soit le problème complémentaire :

Co-Couplage Parfait

Entrée : $G = (U \cup V, E)$.

Question : G ne possède-t-il pas de couplage parfait ?

Comme l'algorithme du Couplage Parfait se fait en temps polynomial (Edmonds - algorithme des fleurs et des pétales), on vérifie aussi que ce n'est pas possible en temps polynomial, ce problème est donc bien dans $\mathcal{NP}$.

Le problème Couplage Parfait est dans $\mathcal{NP}$, et son co-problème est aussi dans $\mathcal{NP}$, donc Couplage Parfait$\in \mathcal{NP} \cap co\mathcal{NP}$.

2.7 Classe $\mathcal{NP}$-complétude

2.7.1 Autour de la Satisfaisabilité

Exercice 57 - Satisfaisabilité optimale

Nous considérons des problèmes de satisfaisabilité maximum ou nous cherchons une affectation de valeur de vérité sur les variables d'une formule conjonctive ϕ qui vise à satisfaire non pas toutes les clauses, mais un nombre maximum d'entre elles.
La variante décisionnelle de Satisfaisabilité Maximale est définie comme suit :

Satisfaisabilité Maximale (MaxSat)

Entrée : Étant donné une formule conjonctive ϕ sur n variables et m clauses et un entier $k \leq m$

Question : Existe-t-il une affectation de valeurs de vérité aux variables de ϕ qui satisfait au moins k clauses ?

Soit la formule insatisfiable en forme normale conjonctive suivante :

$$(x_0 \vee x_1) \wedge (x_0 \vee \neg x_1) \wedge (\neg x_0 \vee x_1) \wedge (\neg x_0 \vee \neg x_1)$$

Quelle que soit l'affectation, on voit bien qu'elle rend au moins une clause insatisfiable :

- $\alpha(x_0) = \alpha(x_1) = \top \Rightarrow \alpha(\neg x_0 \vee \neg x_1) = \bot$;
- $\alpha(x_0) = \alpha(x_1) = \bot \Rightarrow \alpha(x_0 \vee x_1) = \bot$;
- $\alpha(x_0) = \top$ et $\alpha(x_1) = \bot \Rightarrow \alpha(\neg x_0 \vee x_1) = \bot$;
- $\alpha(x_0) = \bot$ et $\alpha(x_1) = \top \Rightarrow \alpha(x_0 \vee \neg x_1) = \bot$.

On s'intéresse donc au problème de satisfaisabilité maximal : quelle affectation parmi ces 4 rend vrai le plus de clauses. Le problème de décision se pose avec un k. On veut donc savoir s'il existe une affectation qui rend k clauses vraies. Posons l'affectation α telle que $\alpha(x_0) = \alpha(x_1) = \top$.

- L'affectation α satisfait-elle au moins une clause ? Oui : $\alpha(x_0 \vee x_1) = \top$;
- L'affectation α satisfait-elle au moins deux clauses ? Oui : $\alpha(x_0 \vee x_1) = \alpha(x_0 \vee \neg x_1) = \top$;
- L'affectation α satisfait-elle au moins trois clauses ? Oui : $\alpha(x_0 \vee x_1) = \alpha(x_0 \vee \neg x_1) = \alpha(\neg x_0 \vee x_1) = \top$.

On vient de réduire le variante d'optimisation au problème de décision en décrivant le problème ! Cela peut donner l'intuition de la preuve de NP-difficulté de la variante d'optimisation.

1. (a) Montrer Satisfaisabilité $\propto$ Satisfaisabilité Maximale.
   (b) En déduire une preuve pour k-Satisfaisabilité $\propto$ k-Satisfaisabilité Maximale.

1. (a) On veut montrer que Satisfaisabilité Maximale est NP-dur. Pour ce faire, nous allons exhiber la réduction de Satisfaisabilité à Satisfaisabilité Maximale, puis prouver sa correction.
   Soit φ une instance de Satisfaisabilité et m le nombre de clauses de φ, donc $\varphi = \bigwedge_{i=1}^{m} C_i$. On a une solution à Satisfaisabilité depuis Satisfaisabilité Maximale si et seulement si MaxSat$(\varphi, m) = \top$.
   En effet, si on peut trouver une affectation qui satisfasse les m clauses de la formule, cette affectation rend la formule vraie. On a donc trouvé un modèle, et celle-ci est satisfiable. D'un autre côté, si une formule est satisfiable, ça veut dire qu'il existe une affectation α telle que les m clauses sont toutes satisfaites, donc MaxSat(φ, m) aura aussi une solution.

(b) En utilisant exactement la même réduction, et le même argument de preuve, on peut réduire k-Satisfaisabilité à k-Satisfaisabilité Maximale.

2. Nous voulons prouver maintenant que 2-Satisfaisabilité Maximale est $\mathcal{NP}$-dur.
   (a) Peut-on procéder à la réduction 2-Satisfaisabilité $\propto$ 2-Satisfaisabilité Maximale ?
   (b) Nous allons procéder à la réduction suivante : 3-Satisfaisabilité Maximale $\propto$ 2-Satisfaisabilité Maximale.

   Considérons une instance de 3-Satisfaisabilité Maximale sur les variables $x_1, x_2, \ldots, x_n$ et les clauses $C_1, \ldots, C_m$ avec $\forall i.|C_i| = 3$. On lui associe ϕ' définie sur les variables $x_1, x_2, \ldots, x_n, y_1, \ldots, y_m$ (m nouvelles variables) et $10m$ clauses $C'_1, \ldots, C'_{10m}$ construites comme suit : considérons la clause $C_i = a_i \vee b_i \vee d_i$ où a_i, b_i et d_i sont des littéraux de variables $x_1, x_2, \ldots, x_n$ à C_i, nous lui associons l'ensemble des dix nouvelles clauses suivantes :

   $$\{a_i, b_i, d_i, (\bar{a_i} \vee \bar{b_i}), (\bar{a_i} \vee \bar{d_i}), (\bar{b_i} \vee \bar{d_i}), (a_i \vee \bar{y_i}), (b_i \vee \bar{y_i}), (d_i \vee \bar{y_i}), y_i\}$$

   La formule ϕ' est obtenue par la conjonction de $10m$ clauses ainsi construites. Montrer que 2-Satisfaisabilité Maximale est $\mathcal{NP}$-difficile.

Les problèmes k-Sat supposent une formule avec *au plus* k littéraux par clause. Cependant, dans le cas de la 2-Sat (ou 2-MaxSat), on peut supposer que toutes les clauses ont 2 littéraux sans perte de généralité : pour chaque clause avec un seul littéral, il suffit de le répéter pour avoir une clause avec deux littéraux, sans changer la satisfaisabilité de la formule !

2. (a) Supposons que 2-Satisfaisabilité Maximale est $\mathcal{NP}$-dur. On sait que 2-Satisfaisabilité est dans $\mathcal{P}$. Comme $\mathcal{P} \subseteq \mathcal{NP}$, et que tous les problèmes de $\mathcal{NP}$ se réduisent à un problème $\mathcal{NP}$-dur, alors 2-Satisfaisabilité se réduirait aussi à 2-Satisfaisabilité Maximale.

   Cependant, la réciproque n'est pas vraie : on ne peut rien déduire de la classe de complexité de 2-Satisfaisabilité Maximale en réduisant 2-Satisfaisabilité à ce problème, on saura juste que 2-Satisfaisabilité Maximale est plus dur que 2-Satisfaisabilité, ce qui est le cas d'un bon nombre de problèmes.

   (b) Pour montrer que 2-Satisfaisabilité Maximale est $\mathcal{NP}$-difficile, prouvons que la réduction est correcte :

   $\Rightarrow$ Commençons par supposer qu'il existe une affectation α qui satisfait k clauses d'une formule φ de 3-Satisfaisabilité Maximale. Pouvons nous trouver k' le nombre de clauses satisfaites dans l'instance 2-Satisfaisabilité Maximale associée ?

Affectations	a_i	b_i	d_i	$(\bar{a_i} \vee \bar{b_i})$	$(\bar{a_i} \vee \bar{d_i})$	$(\bar{b_i} \vee \bar{d_i})$	$(a_i \vee \bar{y_i})$	$(b_i \vee \bar{y_i})$	$(d_i \vee \bar{y_i})$	y_i
$\alpha(a_i) = \alpha(b_i) = \alpha(d_i) = \top$	$\top$	$\top$	$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\top$
$\alpha(a_i) = \alpha(b_i) = \top; \alpha(d_i) = \bot$	$\top$	$\top$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\bot/\top$	$\top/\bot$
$\alpha(b_i) = \alpha(d_i) = \top; \alpha(a_i) = \bot$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\bot$	$\bot/\top$	$\top$	$\top$	$\top/\bot$
$\alpha(a_i) = \alpha(d_i) = \top; \alpha(b_i) = \bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\bot/\top$	$\top$	$\top/\bot$
$\alpha(a_i) = \alpha(b_i) = \bot; \alpha(d_i) = \top$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\bot$
$\alpha(d_i) = \alpha(b_i) = \bot; \alpha(a_i) = \top$	$\top$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\bot$
$\alpha(a_i) = \alpha(d_i) = \bot; \alpha(b_i) = \top$	$\bot$	$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\bot$
$\alpha(a_i) = \alpha(b_i) = \alpha(b_i) = \bot$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\bot$

Si une clause de φ est satisfiable, alors 7 clauses de φ' sont aussi satisfiables. Sinon, 6 clauses seulement sont satisfiables. Ainsi, s'il existe une affectation qui satisfasse k clauses de 3-MaxSat, alors il y a $7k + 6(m - k)$ clauses satisfaites dans le problème 2-MaxSat associé.

⇐ Supposons qu'il existe une affectation d'une instance de 2-Satisfaisabilité Maximale qui satisfasse k' clauses telle que $\exists k.k' = 7k + 6(m-k)$ avec $m = \frac{|\varphi'|}{10}$. Montrons qu'au moins k clauses de 3-Satisfaisabilité Maximale sont satisfaites. Si 7 clauses de 2-Satisfaisabilité Maximale sont satisfaites, alors 1 clause de 3-Satisfaisabilité Maximale est satisfaite. Ainsi, il y a au moins $7k + 6(m-k) = k + \frac{6}{7}(m-k)$ clauses satisfaites. Comme $k \leq m$, on a $m - k \geq 0$ et ainsi $k + \frac{6}{7}(m-k) \geq k$.

La réduction de 3-Satisfaisabilité Maximale à 2-Satisfaisabilité Maximale est correcte, ce problème est donc $\mathcal{NP}$-difficile.

3. La variante décisionnelle de la version générale de Satisfaisabilité Minimale est définie comme suit :

   Étant donné une formule conjonctive ϕ sur n variables et m clauses et un entier $k \leq m$, existe-t-il une affectation de valeurs de vérité aux variables de ϕ qui satisfait au plus k clauses ?

   Montrer que Satisfaisabilité Minimale est $\mathcal{NP}$-complet à partir de Satisfaisabilité Maximale. Pour cela, aidez-vous de la réduction polynomiale suivante :

   Considérons une instance de 2-Satisfaisabilité Maximale notée (ϕ, K_{max}) où ϕ est une formule conjonctive définie sur les variables $x_1, x_2, \ldots, x_n$ et les clauses $C_1, \ldots, C_m$. On lui associe ϕ' définie sur les variables $x_1, x_2, \ldots, x_n, y_1, \ldots, y_m$ (m nouvelles variables) et $2m$ clauses $C'_1, \ldots, C'_{2m}$ construites comme suit : considérons la clause $C_i = a_i \vee b_i$ où a_i, b_i sont des littéraux de variables $x_1, x_2, \ldots, x_n$ à C_i, nous lui associons l'ensemble de 2 nouvelles clauses suivantes :

   $$(\bar{a_i} \vee y_i) \wedge (\bar{b_i} \vee \bar{y_i})$$

   La formule ϕ' est obtenue par la conjonction des $2m$ clauses ainsi construites. Nous posons $K_{min} = 2m - K_{max}$. Montrer que 2-Satisfaisabilité Minimale est $\mathcal{NP}$-difficile en montrant qu'au plus K_{min} clauses sont satisfaites dans ϕ' si et seulement si au moins K_{max} clauses sont satisfaites dans ϕ.

3. Montrons que la réduction est correcte en suivant le même schéma.

   ⇒ On suppose qu'on a une affectation α qui satisfait k clauses de l'instance ϕ. Montrons qu'ainsi, $2m - k$ clauses sont satisfaites dans l'instance ϕ' associée. Il y a 4 cas :

   — $\alpha(a_i) = \alpha(b_i) = \top$, et dans ce cas, soit $\bar{a_i} \vee y_i$, soit $\bar{b_i} \vee \bar{y_i}$ est satisfait, et une seule clause de C'_i est satisfaite.

   — $\alpha(a_i) \neq \alpha(b_i)$, et dans ce cas, on pose $\alpha(y_i) = \alpha(b_i)$, et une seule clause de C'_i est satisfaite.

   — $\alpha(a_i) = \alpha(b_i) = \bot$, et dans ce cas, les deux clauses de C'_i sont satifaites.

   Ainsi, si k clauses de ϕ sont satisfaites, alors k clauses de ϕ' sont satisfaite. De plus, $m - k$ clauses de ϕ ne sont pas satisfaites, donc $2(m-k)$ clauses de ϕ' sont satisfaites. Ainsi, le nombre de clauses K_{min} satisfaites est :

   $$K_{min} = K_{max} + 2(m - K_{max}) = 2m - 2K_{max} + K_{max} = 2m - K_{max}$$

   ⇐ On suppose que le nombre de clauses satisfaites dans ϕ est $k' < K_{max}$. Ainsi, en utilisant le même calcul que précédemment, on a $K_{min} = k' + 2(m - k') = 2m - k'$. Or, comme $k' < K_{max}$, on a $2m - k' > 2m - K_{max}$, donc si $k' < K_{max}$ clauses de ϕ sont satisfaites, $2m - k' > K_{min}$ clauses sont satisfaites dans ϕ'. Par contraposée, on a montré que si au plus K_{min} clauses de ϕ' sont satisfaites, alors au moins K_{max} clauses sont satisfaites dans ϕ.

Exercice 58 - Autour de Satisfaisabilité

Non Égal Satisfaisabilité (NaeSat)

Entrée : Étant donnée une formule conjonctive ϕ sur n variables m clauses.

Question : Existe-t-il une affectation de valeurs de vérité aux variables qui satisfassent ϕ tel que chaque clause à un littéral vrai et un faux ?

Montrer que Non Égal Satisfaisabilité est $\mathcal{NP}$-complet. La preuve se fera à partir de Satisfaisabilité.

Pour montrer que NaeSat est $\mathcal{NP}$-complet, on procède en 2 étapes : d'abord, on montre que NaeSat est dans $\mathcal{NP}$, et ensuite, on réduit un problème $\mathcal{NP}$-complet à NaeSat :

— NaeSat $\in \mathcal{NP}$: on vérifie en temps linéaire que chaque clause possède un littéral à vrai, et un à faux.

— On pose une instance de Sat : $\phi = \bigwedge_{i=1}^{m}(l_i^1 \vee \cdots \vee l_i^k)$. Soit P l'algorithme qui transforme une instance de Sat en instance de NaeSat. On pose une nouvelle variable z telle que $P(\phi) = \bigwedge_{i=1}^{m}(l_i^1 \vee \cdots \vee l_i^k \vee z)$ avec pour valeur de vérité $\alpha(z) = \bot$ (si $\alpha(z) = \top$, alors il suffit de faire le complémentaire de chaque clause). S'il existe une affectation telle que $\alpha(\phi) = \top$, alors $\alpha(P(\phi)) = \top$ aussi, mais on est sûrs que dans chaque clause, il y a au moins un littéral à vrai et un à faux.

NaeSat est dans $\mathcal{NP}$, et on peut réduire Sat à NaeSat. Donc NaeSat est plus difficile que Sat, donc NaeSat est $\mathcal{NP}$-complet.

Exercice 59 - Autour de Satisfaisabilité (suite)

Coupe Maximum (MaxCut)

Entrée : Soit $G = (V, E)$ un graphe non orienté, $k \in \mathbb{N}$

Question : Existe-t-il une partition de sommets en deux sous-ensembles V_1 et V_2 tel que le nombre d'arêtes entre V_1 et V_2 est k ?

Réduire Non Égal 3-Satisfaisabilité à Coupe Maximum (il faut d'abord réduire 3-Satisfaisabilité en 4 Non Égal Satisfaisabilité pour ensuite réduire 4 Non Égal Satisfaisabilité en 3 Non Égal Satisfaisabilité et enfin réduire en Coupe Maximum). Conclure.

De la même manière que l'exercice 58, on procède d'abord par montrer que Coupe Maximum est dans $\mathcal{NP}$, pour ensuite réduire le problème 3-NaeSat en MaxCut et prouver que ce dernier est $\mathcal{NP}$-complet :

— Il suffit de prendre le nombre d'arêtes entre V_1 et V_2 et de la comparer à k. La complexité est $O(|E|)$.

— La réduction entre k-Sat et $(k+1)$-Sat a été exposée lors de l'exercice précédent. La réduction de 3-Sat à 4-NaeSat est donc triviale. Ensuite, pour réduire 4-NaeSat à 3-NaeSat, supposons que nous avons une instance ϕ de 4-NaeSat :

$$\phi = \bigwedge_{i=1}^{m}(l_i^1 \vee l_i^2 \vee l_i^3 \vee l_i^4)$$

Il suffit d'introduire une variable z_i pour chaque clause telle que :

$$P(\phi) = \bigwedge_{i=1}^{m}(l_i^1 \vee l_i^2 \vee z_i) \wedge (l_i^3 \vee l_i^4 \vee \overline{z_i})$$

On pourra en effet trouver une affectation de z_i qui satisfasse ϕ si $P(\phi)$ est satisfaite :

— $\alpha(l_i^1) = \top$ et $\alpha(l_i^2) = \bot$, si $\alpha(l_i^3) = \alpha(l_i^4)$, il suffit de prendre $\alpha(z) = \neg\alpha(l_i^3)$.

— $\alpha(l_i^1) = \top$ et $\alpha(l_i^3) = \bot$, si $\alpha(l_i^2) = \alpha(l_i^4) = \bot$, il suffit de prendre $\alpha(z) = \bot$ et si $\alpha(l_i^4) \neq \alpha(l_i^2)$, alors $\alpha(z) = \neg\alpha(l_i^2)$.

— Pour les autres cas, il suffit d'inverser les 2 clauses.

On a réduit 4-NaeSat à 3-NaeSat. Réduisons maintenant 3-NaeSat à MaxCut. Construisons $G = (V, E)$:

— Pour chaque littéral l_i, $V = \{l_i, \overline{l_i}\} \cup V$ et $E = \{(l_i, \overline{l_i})\} \cup E$.

— Pour chaque clause $C_i = (l_i^1 \vee l_i^2 \vee l_i^3)$ dans ϕ, $E = \{(l_i^1, l_i^2), (l_i^1, l_i^3), (l_i^2, l_i^3)\} \cup E$.

Le nombre d'arêtes totales est donc $3m+n$ (les arêtes entre les littéraux positifs et négatifs, et 3 arêtes pour chaque clause).

Prouvons que 3-NaeSat$(\Phi) \Leftrightarrow$ MaxCut$(C(\Phi), 2m + n)$:

$\Rightarrow$ Supposons que V_1 contienne tous les sommets évalués à $\top$, et V_2 tous les sommets évalués à $\bot$. Si une formule de 3-NaeSat est satisfiable, alors 2 littéraux sont à $\top$, et un à $\bot$ ou bien 2 à $\bot$ et 1 à $\top$ (2 arêtes vont d'un côté à un autre du graphe). Il y a donc $2m + n$ arêtes entre V_1 et V_2. Pour une formule ϕ qui satisfait 3-NaeSat, il existe une coupe maximum de taille $k = 2m + n$.

$\Leftarrow$ Supposons que nous avons une coupe maximum de taille $2m + n$. Montrons qu'il existe une affectation qui satisfait la formule ϕ. Si, pour un littéral l_i, l_i et $\overline{l_i}$ sont dans la même partition, alors le nombre d'arêtes coupées sera $2m+n-1 < 2m+n$. De même, si les trois littéraux d'une clause sont dans la même partition, alors le nombre d'arêtes coupées sera de $2(m-1)+n < 2m+n$. La seule configuration pour qu'exactement $2m + n$ arêtes soient coupées est celle qui satisfait 3-NaeSat.

On a réduit 3-NaeSat à MaxCut, ce problème est donc $\mathcal{NP}$-dur. De plus, MaxCut est dans $\mathcal{NP}$, donc MaxCut est $\mathcal{NP}$-complet.

2.7.2 Problèmes autour des graphes

Exercice 60 - Problème de la coloration

Montrer que 3-Coloriable est $\mathcal{NP}$-complet (réduction à partir de 3-Satisfaisabilité).

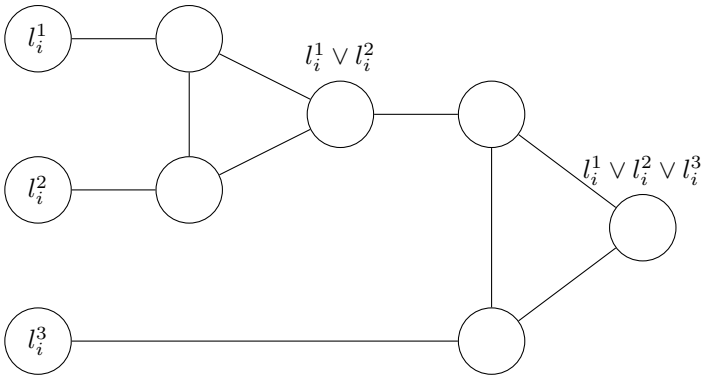
Posons une instance φ de 3-Satisfaisabilité. On suppose sans perte de généralité que $\forall m.|C_m| = 3$ pour m le nombre de clauses de φ. On a donc :

$$\varphi = \bigwedge_{i=1}^{m} (l_i^1 \vee l_i^2 \vee l_i^3)$$

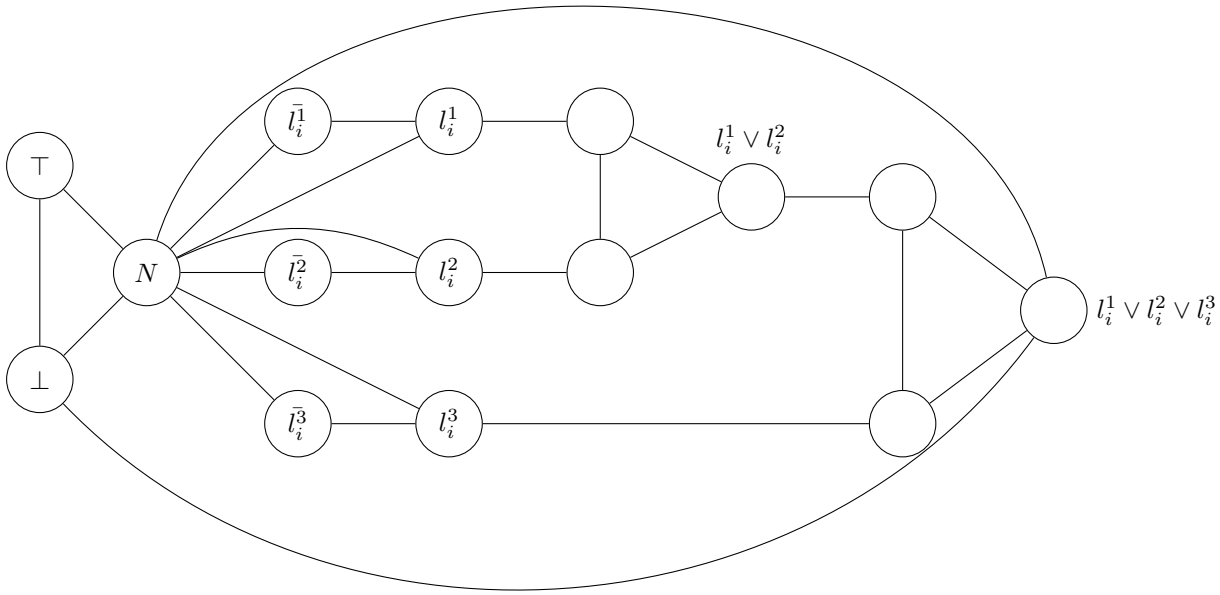
Posons la réduction suivante. Soit $G = (V, E)$ le graphe 3-Coloriable généré à partir de φ. On a $V = \{\top, \bot, N\}$ et $E = \{(\top, \bot), (\bot, N), (\top, N)\}$. De plus :

$$\forall i.V = \{l_i, \bar{l}_i\} \cup V$$
$$\forall i.E = \{(l_i, \bar{l}_i), (l_i, N), (\bar{l}_i, N)\} \cup E$$

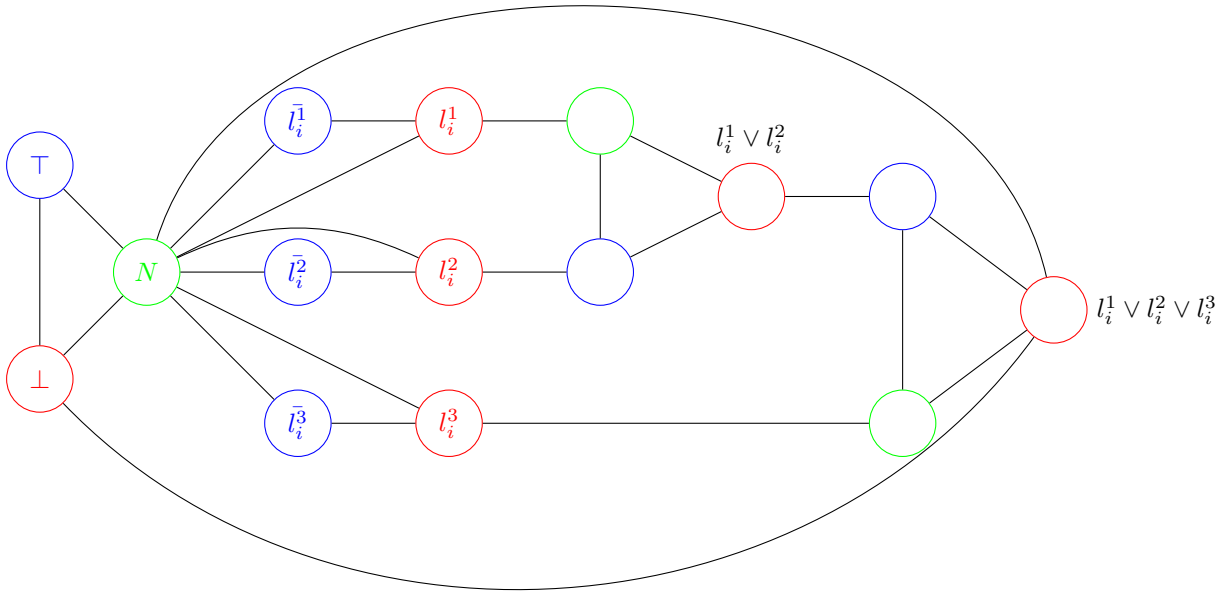
Enfin, pour chaque clause, on crée une porte logique « ou » à partir des littéraux (et le noeud de sortie, donc le noeud $l_i^1 \vee l_i^2 \vee l_i^3$ doit être relié avec le noeud neutre et le noeud faux) :



Pour résumer, pour une clause $i = (l_i^1 \vee l_i^2 \vee l_i^3)$, voici le graphe produit par la réduction :



Montrons la correction de la réduction. Tout d'abord, supposons que le graphe est 3-coloriable. On construit l'assignement aux littéraux de φ en mettant à $\top$ tous les sommets en bleu, et $\bot$ tous les sommets en rouge. Si la formule n'est pas satisfiable, alors il existe une clause i telle que $C_i = (l_i^1 \vee l_i^2 \vee l_i^3)$ non satisfiable, donc la couleur de l_i^1, l_i^2 et l_i^3 est rouge. Le noeud $l_i^1 \vee l_i^2 \vee l_i^3$ sera, lui aussi, rouge, et ainsi, le graphe n'est pas coloriable :



Le noeud $l_i^1 \vee l_i^2 \vee l_i^3$ et $\bot$ sont de la même couleur, et on ne peut pas trouver de 3-Coloration valide dans ce graphe. On a prouvé que s'il existe i tel que $\alpha(C_i) = \alpha(l_i^1 \vee l_i^2 \vee l_i^3) = \bot$, alors le graphe n'est pas 3-Coloriable.

Supposons maintenant que nous avons une affectation qui valide toutes les clauses de ϕ. Si on reprend les couleurs du graphe précédent, un des trois littéraux de la clause sera coloré en bleu. Ainsi, la porte « ou » (qui est toujours 3-coloriable) sortira un noeud bleu (sur $l_i^1 \vee l_i^2 \vee l_i^3$), et ainsi le graphe sera 3-Coloriable.

La réduction est correcte, 3-Coloriable est $\mathcal{NP}$-dur. Prouvons maintenant que 3-Coloriable est $\mathcal{NP}$-complet. Il nous reste à savoir s'il existe un certificat positif polynomial pour le problème. Soit $\mathcal{COLS}$ la liste de couples (couleur, sommet). La vérification que $\mathcal{COLS}$ est une 3-Coloration valide se fait en temps linéaire sur $|E|$: il suffit de vérifier que, pour chaque sommet, tous ses voisins ont une couleur différente.

Exercice 61 - Voyageur de commerce

Voyageur de Commerce (TSP)

Entrée : Un ensemble de m villes X, un ensemble de routes entre les villes E. Une fonction de coût $v : E \to \mathbb{N}$ où $v(x, y)$ est le coût de déplacement de x à $y, k \in \mathbb{N}$.

Question : Existe-t-il un cycle Hamiltonien de distance inférieure ou égale à k ?

Montrer que Voyageur de Commerce est $\mathcal{NP}$-complet (la preuve se fait à partir de Cycle Hamiltonien). Qu'en est-il si on autorise l'inégalité triangulaire $\forall i, j, k. c_{ik} \leq c_{ij} + c_{jk}$?

Prouvons que TSP est $\mathcal{NP}$-complet. Tout d'abord, TSP est dans $\mathcal{NP}$: il suffit de vérifier que tous les sommets du graphe sont visités et que la somme des distances est inférieur ou égal à k, ce qui se fait en $O(|V|)$. Montrons maintenant que le problème est $\mathcal{NP}$-dur en réduisant Cycle Hamiltonien à TSP.

Posons la réduction de $G = (V, E)$ une instance de Cycle Hamiltonien à $G' = (V', E')$ une instance

de Voyageur de Commerce suivante :

$$V' = V$$
$$E' = E \cup \bar{E}$$
$$v(x,y) = \begin{cases} 1 & \text{si } (x,y) \in E \\ 2 & \text{sinon} \end{cases}$$

En clair, G' est le graphe complet de G, et toutes les arêtes originelles de G ont une valuation à 1. Ainsi, pour réduire Cycle Hamiltonien à Voyageur de Commerce, si on pose P la fonction qui construit G' à partir de G, il suffit de lancer $\text{TSP}(P(G), |V|)$. Montrons la correction de cette réduction : on a une solution à Cycle Hamiltonien si et seulement si on a une solution à $\text{TSP}(P(G'))$.

$\Rightarrow$ Supposons qu'il y a une solution de Cycle Hamiltonien dans G. De toute évidence, il y a une solution de taille $|V|$ à $\text{TSP}(P(G))$.

$\Leftarrow$ Supposons qu'il y ait une solution de distance $|V|$. Comme le Voyageur de Commerce doit passer par tous les sommets du graphe, le seul moyen d'avoir une solution de distance $|V|$ est le Cycle Hamiltonien de G.

Exercice 62 - Recouvrement de sommets

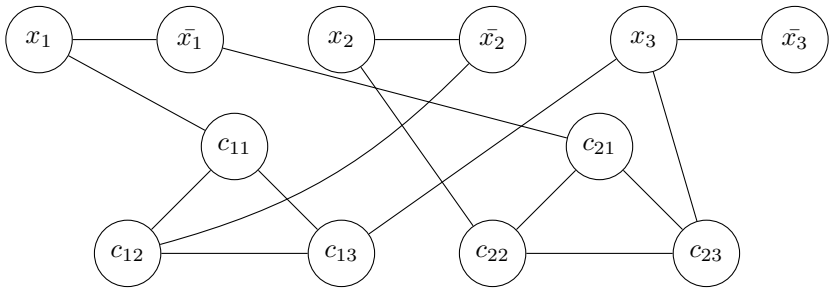
On veut montrer que le problème Recouvrement de sommets est $\mathcal{NP}$-complet. La preuve se fera à partir de 3-Satisfaisabilité.
Aide pour la transformation polynomiale : considérons les variables $x_1, \bar{x_1}, x_2, \bar{x_2}, \ldots, x_n, \bar{x_n}$ et n arêtes $(x_i, \bar{x_i}), \forall i = 1 \ldots n$. Nous considérons m triangles constitués des littéraux. Pour une clause C_i, nous notons c_{i1}, c_{i2}, c_{i3} et nous relions le sommet x_i à un sommet d'un triangle noté $c_{jk}, k = 1, 2, 3$ si la variable x_i apparît dans la clause C_j à la position k.

Pour rappel, le problème de Recouvrement de sommets (Vertex-Cover) cherche un ensemble de sommets qui couvrent toutes les arêtes d'un graphe.

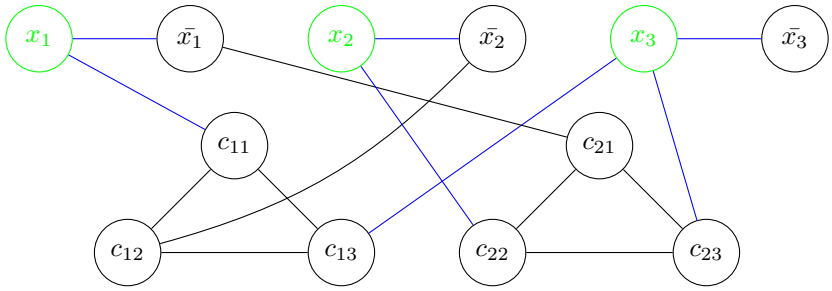
La transformation polynomiale est donnée dans l'énoncé, illustrons-la avec la formule suivante :

$$(x_1 \vee \bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee x_2 \vee x_3)$$

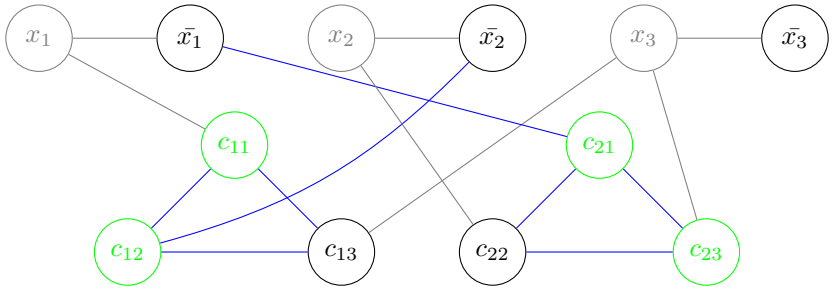
Cette formule est clairement satisfiable, il suffit que la valeur de vérité de x_3 soit à $\top$. Le graphe construit grâce à la transformation polynomiale est le suivant :



Essayons de construire un recouvrement de sommets en prenant, dedans, toutes les variables valuées à $\top$:



On remarque que, pour couvrir toutes les arêtes, on a besoin de deux des trois sommets de chaque « triangle » qui représentent une clause. De plus, toutes les arêtes des littéraux valués à $\bot$ ne sont pas couvertes, il faut donc prendre les sommets des triangles qui couvrent celles-ci en priorité (les sommets et arêtes grisées sont celles qui ont déjà été prise précédemment) :



Ave cette technique, toutes les arêtes sont couvertes avec pour taille de recouvrement 7, on remarque que c'est égal à la somme du nombre de littéraux et de deux fois le nombre de clauses. On peut généraliser cette méthode, et en notant n le nombre de littéraux de la formule et m le nombre de clauses, on peut toujours construire un Recouvrement de sommets de taille $2m + n$ si une formule est satisfiable.

Prouvons que cette construction est valide. Pour ce faire, prouvons qu'une formule est satisfiable si et seulement s'il existe un Recouvrement des sommets de taille $2m + n$.

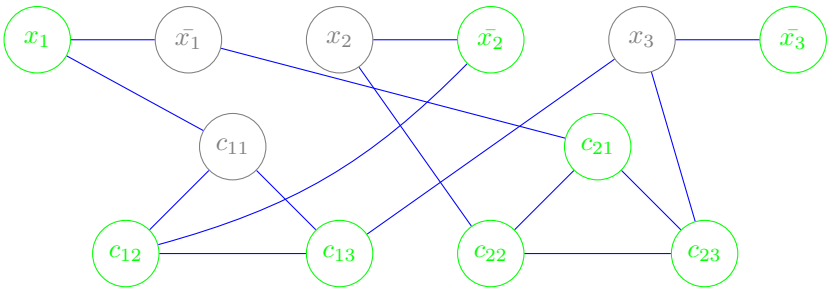
$\Rightarrow$ Soit l'affectation α celle qui rend la formule vraie. Pour chaque clause $(l_i^1 \vee l_i^2 \vee l_i^3)$, il y a 3 cas :

- $\alpha(l_i^1) = \alpha(l_i^2) = \alpha(l_i^3) = \top$. Tous les littéraux sont à vrai, dans ce cas, on prend dans notre couverture de sommets les littéraux l_i^1, l_i^2 et l_i^3, ainsi que 2 des 3 sommets du triangle associé à la clause au hasard, c_{i1} et c_{i2} par exemple.
- $\alpha(l_i^{k_1}) = \alpha(l_i^{k_2}) = \top$ et $\alpha(l_i^{k_3}) = \bot$ avec $k_1 \neq k_2 \neq k_3$. Deux des trois littéraux sont à vrai, dans ce cas, on prend dans notre couverture de sommets les littéraux valués à $\top$: $l_i^{k_1}$ et $l_i^{k_2}$, ainsi que $\overline{l_i^{k_3}}$. De plus, 2 des 3 sommets du triangle associé à la clause doivent aussi être pris dans le recouvrement de sommets, en particulier, on prend celui associé au littéral valué à faux (c_{ik_3}) et un des deux autre, au hasard : c_{ik_1} par exemple.
- $\alpha(l_i^{k_1}) = \top$ et $\alpha(l_i^{k_2}) = \alpha(l_i^{k_3}) = \bot$. Un des trois littéraux est à vrai. Dans le recouvrement de sommets, on met $l_i^{k_1}$ ainsi que $\overline{l_i^{k_2}}$ et $\overline{l_i^{k_3}}$. De plus, on prend les 2 sommets associés à la clause reliés aux littéraux valués à faux pour bien couvrir toutes les arêtes : c_{ik_2} et c_{ik_3}.

Dans les 3 cas, tous les sommets de la clause sont dans le recouvrement, ainsi que 2 des 3 sommets sur le triangle d'une clause. Sachant qu'il y a n littéraux, et m clauses, le nombre total de sommets dans le recouvrement sera de $2m + n$ si la formule est satisfiable.

$\Leftarrow$ Supposons maintenant que nous avons un recouvrement de sommets de taille $2m + n$. Pour reconstruire une solution à 3-Sat, il suffit de mettre tous les sommets de la couverture qui représentent des littéraux à $\top$. En effet, on vient de prouver que si une clause était satisfaite (au moins un littéral est valué à $\top$), alors il y a $2m + n$ sommets dans le recouvrement.

Montrons maintenant que, si la formule est insatisfiable, alors le nombre de sommets du recouvrement n'est pas $2m+n$. Si la formule est insatisfiable, on se retrouve dans une situation ou aucune des arêtes liées à un triangle ne sont couvertes. Donc il existe une clause C_i telle que c_{i1}, c_{i2} et c_{i3} sont dans le recouvrement de sommets. Par exemple, en gardant le graphe de tout à l'heure, mais en affectant des valeurs qui ne satisfont pas la deuxième clause, on devra prendre ces sommets pour avoir une couverture :



8 sommets doivent être pris pour recouvrir toutes les arêtes du graphe avec un assignement qui ne satisfait pas la formule de base. Si une formule est insatisfiable, pour toute affectation α', il existe une clause C_i telle que $\alpha'(C_i) = \bot$, et on se retrouvera dans le cas de figure ci-dessus : obligés de prendre les 3 sommets du triangle qui la représente pour couvrir toutes les arêtes. Ainsi, la taille du recouvrement serait de $2m+n+1 \neq 2m+n$.

On en conclut que la taille du recouvrement sera de $2m+n$ seulement si la formule est satisfiable.

Le problème 3-Satisfaisabilité se réduit en Recouvrement de sommets, donc le problème Recouvrement de sommets est $\mathcal{NP}$-dur. Montrons maintenant qu'il est aussi $\mathcal{NP}$-complet.

Recouvrement de sommets est dans $\mathcal{NP}$ car il suffit de vérifier si la taille de la solution est bien égale à la taille demandée, et que pour chaque arête, un des deux sommets est dans la solution. La vérification du certificat se fait en $O(|E|)$, le certificat est donc polynomial et Recouvrement de sommets est bien dans $\mathcal{NP}$. Ainsi, Recouvrement de sommets est $\mathcal{NP}$-complet.

Exercice 63 - Recouvrement de sommets (suite)

Montrer que Recouvrement de sommets reste $\mathcal{NP}$-complet même si tous les sommets sont de degré pairs.

Pour cet exercice, on a besoin de savoir que le nombre de sommets à degré impair dans un graphe est pair. On peut le remarquer assez facilement en faisant quelques exemples.

Pour montrer que Recouvrement de sommets de degré pairs est $\mathcal{NP}$-complet, on va d'abord montrer que ce problème est dans $\mathcal{NP}$, et que Recouvrement de sommets $\propto$ Recouvrement de sommets de degré pairs.

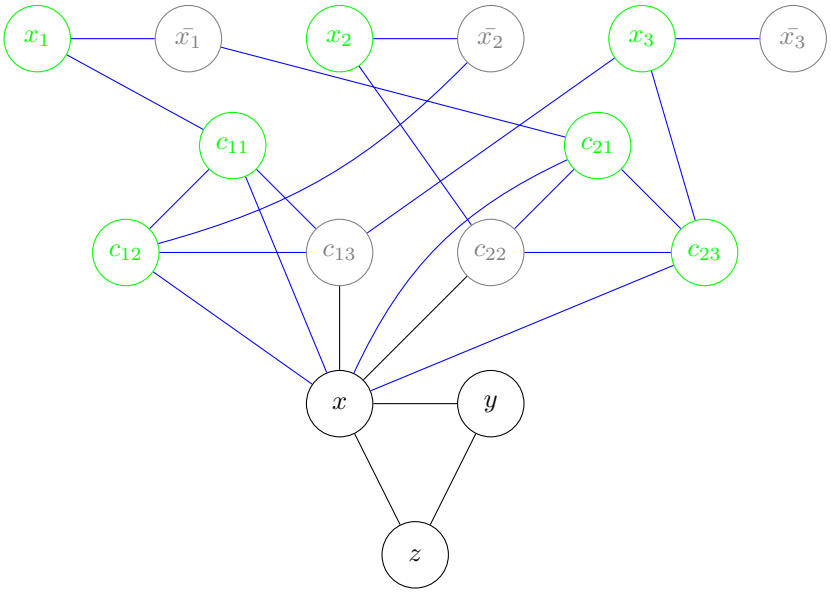
— Recouvrement de sommets de degré pairs est dans $\mathcal{NP}$: on a besoin du même certificat que Recouvrement de sommets : S l'ensemble des sommets qui couvrent toutes les arêtes. On vérifie ensuite si, pour chaque arête (u,v), $u \in S$ ou $v \in S$. Le certificat positif est bien polynomial : il est en $O(|E|)$.

— Montrons que Recouvrement de sommets de degré pairs est $\mathcal{NP}$-dur. Prenons une instance $G=(V,E)$ de Recouvrement de sommets. On va transformer polynomialement G en $G'=(V',E')$, instance de Recouvrement de sommets de degré pairs. Soit d la fonction qui, pour tout sommet

de V, renvoit son degré. Soit $V_{pair} = \{v \mid v \in V \wedge d(v) \mod 2 = 0\}$. Construisons G' :

$$V' = V \cup \{x, y, z\}$$
$$E' = E \cup \{(x, y), (y, z), (x, z)\} \cup \bigcup_{u \in V_{pair}} \{(u, x)\}$$

En reprenant l'exemple de l'exercice précédent, le graphe suivant serait construit (en reprenant ses sommets couvrants en vert et les arêtes couvertes en bleu) :



On remarque que, pour que toutes les arêtes soient couvertes, il faut couvrir les arêtes (c_{13}, x), (c_{22}, x), (x, y), (x, z) et (y, z). Ainsi, on peut prendre le sommet x, puis un des deux sommets y ou z au choix. Si 7 sommets suffisaient à couvrir le graphe G, il en faut 9 pour couvrir le graphe G'.

On peut généraliser ce principe, et, pour prouver la correction de cette réduction, prouvons qu'il existe un Recouvrement de sommets de taille k si et seulement s'il existe un Recouvrement de sommets de degré pairs de taille $k + 2$:

$\Rightarrow$ On sait que le nombre de sommets de degré impairs d'un graphe est pair. De plus, en comptant seulement le triangle (x, y, z), $d(x) = 2$. La somme de 2 et d'un nombre pair reste pair. Donc $d(x)$ reste pair dans tous les cas.

Il existe un recouvrement de taille k, donc toutes les arêtes internes au graphe sont couvertes par ce recouvrement. De plus, prendre x dans le recouvrement va couvrir toutes les arêtes non-couvertes des sommets de degré précédemment impair. Il reste alors une arête encore non couverte : (y, z). En prenant un de ces deux sommets, la couverture est totale.

On a montré que, s'il existe un recouvrement de taille k dans un graphe, alors il existe un recouvrement de taille $k + 2$ dans un graphe avec seulement des sommets de degré pairs.

$\Leftarrow$ Montrons que s'il existe un recouvrement de taille $k+2$ dans G', alors il existe un recouvrement de taille k dans G. Pour reconstruire la solution de G à partir d'une solution de G', il suffit de prendre la solution S' et d'y enlever tous les sommets qui ne font pas partie de G. On a $S = S' - \{x, y, z\}$.

Or, comme seulement deux des trois sommets sont dans S' (par construction), on a $|S| = |S'| - 2$. Comme $|S'| = k + 2$, on a $|S| = k$.

On a montré que s'il existe un recouvrement de taille $k+2$ dans G', alors il existe un recouvrement de taille k dans G.

Recouvrement de sommets de degré pairs est dans $\mathcal{NP}$, et Recouvrement de sommets $\propto$ Recouvrement de sommets de degré pairs, donc Recouvrement de sommets de degré pairs est aussi $\mathcal{NP}$-dur. On en conclut que Recouvrement de sommets de degré pairs est $\mathcal{NP}$-complet.

Exercice 64 - Réductions autour de Hamiltonisme

Montrer que les problèmes sont tous $\mathcal{NP}$-complets si l'un d'eux l'est :

1. Cycle Hamiltonien dans un graphe non orienté
2. Chaîne Hamiltonienne dans un graphe non orienté
3. Circuit Hamiltonien dans un graphe orienté
4. Chemin Hamiltonien dans un graphe orienté
5. Cycle Hamiltonien dans un graphe biparti non orienté
6. Chaîne Hamiltonienne dans un graphe biparti non orienté

Pour rappel, les problèmes Hamiltoniens sont des problèmes où on cherche une solution qui passe par tous les sommets du graphe.

Pour prouver que les problèmes sont tous $\mathcal{NP}$-complets si l'un d'eux l'est, on va procéder en deux étapes : prouver qu'ils sont tous dans $\mathcal{NP}$, puis et réduire chacun l'un à l'autre :

— On considère que l'entrée de chaque problème est $G = (V, E)$.

1. Cycle Hamiltonien est dans $\mathcal{NP}$. En effet, en prenant comme certificat les sommets du cycle, il est polynomial en $O(|V|)$.
2. Chaîne Hamiltonienne est dans $\mathcal{NP}$. En effet, on peut prendre le même certificat que pour le Cycle Hamiltonien.
3. Circuit Hamiltonien est l'équivalent de Cycle Hamiltonien dans un graphe orienté, on peut donc prendre le même certificat, polynomial en $O(|V|)$, donc Circuit Hamiltonien est dans $\mathcal{NP}$.
4. Chemin Hamiltonien est la notion de Chaîne Hamiltonienne dans un graphe orienté. On prend le même certificat pour dire que Chemin Hamiltonien est dans $\mathcal{NP}$.
5. Cycle Hamiltonien dans un graphe biparti non orienté est aussi dans $\mathcal{NP}$, car la notion reste similaire.
6. Chaîne Hamiltonienne dans un graphe biparti non orienté est aussi dans $\mathcal{NP}$, car la notion reste identique.

Ces problèmes sont bien tous dans $\mathcal{NP}$, prouvons maintenant qu'ils se réduisent l'un à l'autre.

— Montrons d'abord que Cycle Hamiltonien $\propto$ Chaîne Hamiltonienne, et que Chaîne Hamiltonienne $\propto$ Cycle Hamiltonien.
Soit $G = (V, E)$ une instance de Cycle Hamiltonien, et $G' = (V', E')$ une instance de Chaîne Hamiltonienne. Soit $u \in V$ un sommet quelconque. Posons la réduction de G à G' suivante :

$$V' = V \cup \{v\}$$
$$E' = E \cup \{(u, v)\}$$

Cette réduction est clairement polynomiale. Montrons maintenant la correction de cette réduction, c'est à dire, montrons qu'il existe un cycle Hamiltonien dans G si et seulement s'il existe un chaîne Hamiltonienne dans G' :

⇒ Supposons qu'il existe un cycle Hamiltonien dans G. Posons u' un sommet voisin de u. Un cycle Hamiltonien est un cycle qui passe par tous les sommets du graphe une fois et une seule. Ainsi, si $|V| = n$, on peut noter ce cycle d'u' à lui-même : $C = (u', v_1, \ldots, v_{n-1}, u')$ avec $\forall i.v_i \in V$ et $u = v_{n-1}$ sans perte de généralité (si $u = v_1$, alors il suffit de parcourir le graphe dans l'autre sens pour avoir $u = v_{n-1}$). On peut exhiber la chaîne Hamiltonienne suivante dans le graphe G' : $C' = (u', v_1, \ldots, u, v)$. C'est bien une chaîne Hamiltonienne : elle passe une fois et une seule par v et chaque sommet de V, puisque C est un cycle Hamiltonien.

⇐ Supposons qu'il existe une chaîne Hamiltonienne dans G'. Soit $(u', v_1, \ldots, u, v)$ cette chaîne. Comme u' est voisin de u, alors $(u', v_1, \ldots, u, u')$ est un cycle Hamiltonien pour G.

Chaîne Hamiltonienne est dans $\mathcal{NP}$, et Cycle Hamiltonien $\propto$ Chaîne Hamiltonienne. Donc, si Cycle Hamiltonien est $\mathcal{NP}$-complet, Chaîne Hamiltonienne est aussi $\mathcal{NP}$-complet. Montrons l'autre sens.

Soit $G = (V, E)$ une instance de Chaîne Hamiltonienne, et $G' = (V', E')$ une instance de Chaîne Hamiltonienne. Soit s, t deux sommets de V tels qu'il existe une Chaîne Hamiltonienne entre s et t. On a :

$$V' = V \cup \{u\}$$
$$E' = E \cup \{(s, u), (t, u)\}$$

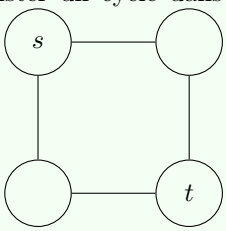
Cette réduction est clairement polynomiale. Montrons maintenant la correction de cette réduction, c'est à dire, montrons qu'il existe une chaîne Hamiltonienne dans G si et seulement s'il existe un cycle Hamiltonien dans G' :

⇒ Supposons qu'il existe une chaîne Hamiltonienne dans G, c'est à dire qu'il existe deux sommets s, t tels que $(s, v_1, \ldots, v_{n-2}, t)$, avec $\forall i.v_i \in V$, est une chaîne Hamiltonienne. Alors $(u, s, v_1, \ldots, v_{n-2}, t, u)$ est un cycle Hamiltonien dans G' : il passe par u deux fois et tous les autres sommets une fois et une seule (car $(s, v_1, \ldots, v_{n-2}, t)$ est une chaîne Hamiltonienne).

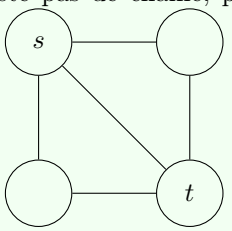
⇐ Supposons qu'il existe un cycle Hamiltonien dans G', c'est à dire qu'il existe $C = (u, s, v_1, \ldots, v_{n-2}, t, u)$ un cycle qui passe par tous les sommets une fois et une seule, excepté u. En retirant u, nous avons bien une chaîne Hamiltonienne dans le graphe G.

Cycle Hamiltonien est dans $\mathcal{NP}$, et Chaîne Hamiltonienne $\propto$ Cycle Hamiltonien. Ainsi, Chaîne Hamiltonienne est polynomialement équivalent à Cycle Hamiltonien. Si un problème $\mathcal{NP}$-complet se réduit à un de ces deux problèmes, alors ils seront tous les deux $\mathcal{NP}$-complets.

Dans les deux réductions précédentes, nous sommes obligés de rajouter un sommet, car sinon, il pourrait exister un cycle dans un graphe alors qu'il n'existe pas de chaîne, par exemple :



Graphe sans chaîne Hamiltonienne entre s et t.



Graphe avec cycle Hamiltonien.

Montrons que Cycle Hamiltonien est polynomialement équivalent à Circuit Hamiltonien. Commençons par montrer que Cycle Hamiltonien $\propto$ Circuit Hamiltonien.

Soit $G = (V, E)$ un graphe non-orienté. Soit $G' = (V', E')$ le graphe orienté produit en transformant

G. La transformation est la suivante :

$$V' = V$$
$$E' = \bigcup_{(u,v)\in E} \{(u,v),(v,u)\}$$

Cette réduction est bien polynomiale (en $O(E)$). La réduction est assez évidente : un graphe non-orienté correspond simplement à un graphe orienté, qui, pour chaque arête, est navigable dans les deux sens.

Prouvons la correction de cette réduction. Montrons qu'il existe un cycle Hamiltonien dans un graphe non-orienté G si et seulement s'il existe un graphe Hamiltonien dans le graphe orienté G' construit à partir de G :

- $\Rightarrow$ Supposons qu'il existe un cycle Hamiltonien dans G un graphe non orienté. C'est à dire que pour $|V| = n$, il existe un chemin $C = (u, v_1, \ldots, v_{n-1}, u)$. Comme chaque arête $(v_{i-1}, v_i) \in E'$, ce même cycle est un cycle Hamiltonien dans un graphe orienté.
- $\Leftarrow$ Supposons qu'il existe un cycle Hamiltonien dans un graphe orienté. Ce même cycle sera un cycle Hamiltonien dans le graphe non-orienté associé.

Cycle Hamiltonien $\propto$ Circuit Hamiltonien, et Circuit Hamiltonien dans $\mathcal{NP}$, donc si Cycle Hamiltonien est $\mathcal{NP}$-complet, alors Circuit Hamiltonien l'est aussi. Montrons maintenant que ces deux problèmes sont polynomialement équivalents, c'est à dire que Circuit Hamiltonien $\propto$ Cycle Hamiltonien.

Soit $G = (V, E)$ un graphe orienté. Soit $G' = (V', E')$ le graphe orienté produit en transformant G. La transformation polynomiale ($O(|E| + |V|)$) est la suivante :

$$V' = \bigcup_{v\in V} \{v_{in}, v, v_{out}\}$$
$$E' = \bigcup_{(u,v)\in E} \{(u_{out}, v_{in})\} \cup \bigcup_{v\in V} \{(v_{in}, v), (v, v_{out})\}$$

Montrons que cette réduction est correcte : prouvons qu'il existe un circuit Hamiltonien dans G si et seulement s'il existe un cycle Hamiltonien dans G'.

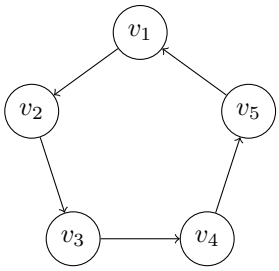
- $\Rightarrow$ Supposons qu'il existe un circuit Hamiltonien dans G : $(u, v_1, \ldots, v_{n-1}, u)$. Il existera alors le cycle suivant de G' : $(u, u_{out}, v_{1_{in}}, v_1, v_{1_{out}}, \ldots, v_{n-1_{in}}, v_{n-1}, v_{n-1_{in}}, u_{in}, u)$.
- $\Leftarrow$ Supposons qu'il existe un cycle Hamiltonien dans G'. Comme chaque $v \in V'$ non-étiqueté (par *in* ou *out*) est de degré 2, il doit être suivi ou précédé par v_{in} et v_{out}. Ainsi, si le cycle est $(u, u_{out}, v_{1_{in}}, v_1, v_{1_{out}}, \ldots, v_{n-1_{in}}, v_{n-1}, v_{n-1_{in}}, u_{in}, u)$, le circuit Hamiltonien correspondant sera $(u, v_1, \ldots, v_{n-1}, u)$.

Circuit Hamiltonien $\propto$ Cycle Hamiltonien, donc ces deux problèmes sont polynomialement équivalents. Si on montre qu'un des deux est $\mathcal{NP}$-complet, l'autre le sera également.

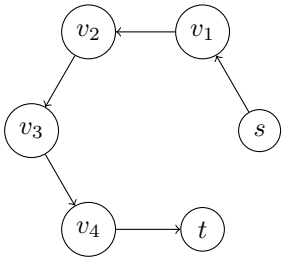
Montrons maintenant que Circuit Hamiltonien $\propto$ Chemin Hamiltonien. Soit $G = (V, E)$ un graphe orienté, $u \in V$ et $G' = (V', E')$ le graphe orienté produit en transformant G de façon suivante :

$$V' = V - \{u\} \cup \{u_{in}, u_{out}\}$$
$$E' = E - \bigcup_{(u,v)\in E} \{(u,v)\} - \bigcup_{(v,u)\in E} \{(v,u)\} \cup \bigcup_{(u,v)\in E} \{(u_{out}, v)\} \cup \bigcup_{(v,u)\in E} \{(v, u_{in})\}$$

En clair, on crée deux noeuds qui vont être l'entrée et la sortie d'un noeud du cycle dans G, et on le « casse » en deux pour avoir un chemin à partir de ce noeud. Pour illustrer, voici un graphe G contenant un cycle à gauche, et le graphe G' construit par cette réduction polynomiale :



Graphe orienté avec cycle : $(v_1, v_2, v_3, v_4, v_5, v_1)$



Graphe orienté avec chemin : $(s, v_1, v_2, v_3, v_4, t)$

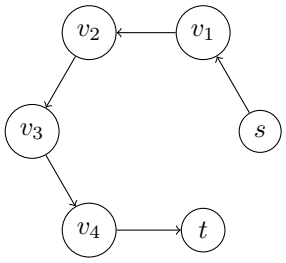
Dans l'illustration ci-dessus, on a remplacé v_5 par les deux noeuds s et t. On voit bien que comme il y a un cycle à gauche, il y a un chemin à droite. Montrons maintenant la correction de cette réduction pour n'importe quel graphe. Prouvons qu'il existe un circuit Hamiltonien dans G si et seulement s'il existe un chemin dans G'.

$\Rightarrow$ Supposons qu'il y a un circuit Hamiltonien dans G, c'est à dire que pour $|V| = n$, il existe un chemin $(u, v_1, \ldots, v_{n-1}, u)$. Avec G', ce on doit remplacer le premier u dans le circuit par u_{out} et le second par u_{in}, les deux nouveaux sommets, ce qui donne le chemin suivant : $(u_{out}, v_1, \ldots, v_{n-1}, u_{in})$. Ce chemin est bien Hamiltonien : il passe par tous les sommets du graphe, et comme il n'y a pas d'arête entre u_{in} et u_{out}, il n'y a pas de circuit. On a montré que s'il existe un circuit dans G, alors il existe un chemin dans G'.

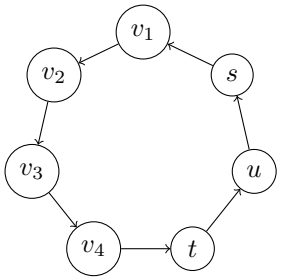
$\Leftarrow$ S'il existe une chaîne Hamiltonienne entre s et t, alors en prenant toutes les arêtes de sorties de s et en les ajoutant à t, ou plus formellement $\forall (s, v) \in E' \rightsquigarrow (t, v) \in E'$, on construit un circuit Hamiltonien à partir de ce chemin. En effet, on passe par tous les sommets (excepté s, qu'on supprime), et le chemin commence à t et fini à t.

On a montré que Circuit Hamiltonien $\propto$ Chemin Hamiltonien, et comme Chemin Hamiltonien est dans $\mathcal{NP}$, alors si Circuit Hamiltonien est $\mathcal{NP}$-complet, alors Chemin Hamiltonien sera aussi $\mathcal{NP}$-complet.

Prouvons maintenant que ces deux problèmes sont polynomialement équivalents, donc que Chemin Hamiltonien $\propto$ Circuit Hamiltonien. En utilisant une réduction similaire à l'équivalent du problème en non-orienté, on se convainc que Chemin Hamiltonien $\propto$ Circuit Hamiltonien. Pour donner une intuition de la correction, on peut l'illustrer de la manière suivante :



Graphe orienté avec chemin : $(s, v_1, v_2, v_3, v_4, t)$



Graphe orienté avec circuit : $(u, s, v_1, v_2, v_3, v_4, t, u)$

Comme on a Chemin Hamiltonien $\propto$ Circuit Hamiltonien et Circuit Hamiltonien $\propto$ Chemin Hamiltonien, ces deux problèmes sont polynomialement équivalents, donc si l'un est $\mathcal{NP}$-complet, l'autre le sera aussi.

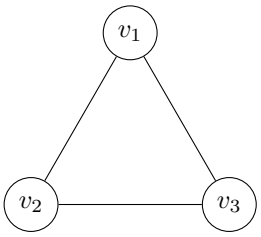
Il ne reste plus qu'à traiter les cas des problèmes bipartis. On va montrer que Cycle Hamiltonien Biparti est polynomialement équivalent à Cycle Hamiltonien, pour ensuite montrer que Chaîne Hamiltonienne Bipartie est polynomialement équivalent à Cycle Hamiltonien Biparti.

Montrons d'abord que Cycle Hamiltonien $\propto$ Cycle Hamiltonien Biparti. Soit $G = (V, E)$ une instance de Cycle Hamiltonien. Soit $G' = (V', E')$ l'instance de Cycle Hamiltonien Biparti

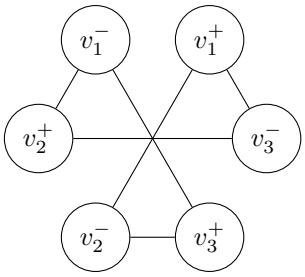
construit grâce à la réduction suivante :

$$V' = \bigcup_{v \in V} \{v^+, v^-\}$$
$$E' = \bigcup_{(u,v) \in E} \{(u^+, v^-), (u^-, v^+)\}$$

On différencie les cas pairs et impairs. Si $|V|$ est pair, alors pour que cette réduction marche, il faut ajouter un autre sommet, x, et pour une arête $(i, j) \in E$, ajouter (i, x) et (x, j) à E, puis réduire comme décrit précédemment. Cette réduction est bien polynomiale : elle s'effectue en $O(|V| + |E|)$. Pour l'illustrer, transformons le graphe (gauche) en graphe biparti (droite) :



Graphe non orienté avec cycle : (v_1, v_2, v_3, v_1)



Graphe non orienté biparti avec cycle : $(v_1^-, v_2^+, v_3^-, v_1^+, v_2^-, v_3^+, v_1^-)$

Montrons que cette réduction est correcte, c'est à dire, montrons qu'il existe un cycle Hamiltonien dans G si et seulement s'il existe un cycle Hamiltonien dans G' :

$\Rightarrow$ Supposons qu'il existe un cycle Hamiltonien dans G, c'est à dire qu'il y a une chaîne $(v_1, v_2, \ldots, v_n, v_1)$. Par construction, il existe le cycle suivant dans G' : $(v_1^-, v_2^+, \ldots, v_n^-, v_1^+, v_2^-, \ldots, v_n^+, v_1^-)$.

$\Leftarrow$ Supposons existe un cycle Hamiltonien dans G'. Pour reconstruire une solution pour G, il suffit de prendre les $n + 1$ premiers sommets de la solution de G' en supprimant les polarités.

On a montré que Cycle Hamiltonien $\propto$ Cycle Hamiltonien Biparti. Comme Cycle Hamiltonien Biparti est dans $\mathcal{NP}$, si Cycle Hamiltonien est $\mathcal{NP}$-complet, Cycle Hamiltonien Biparti le sera aussi.

Montrons maintenant l'équivalence polynomiale entre ces deux problèmes, c'est à dire : prouvons que Cycle Hamiltonien Biparti $\propto$ Cycle Hamiltonien. Soit $G = (V, E)$ une instance de Cycle Hamiltonien Biparti, on construit $G' = (V', E')$ une instance de Cycle Hamiltonien de la manière suivante (avec v_1 et v_2 voisins) :

$$V' = V \cup \{x\}$$
$$E' = E \cup \{(v_1, x), (x, v_2)\}$$

Cette réduction est polynomiale en $O(|V| + |E|)$. Montrons la correction de cette réduction, c'est à dire prouvons qu'il existe un cycle Hamiltonien dans G si et seulement s'il existe un cycle Hamiltonien dans G' :

$\Rightarrow$ Supposons qu'il existe un cycle Hamiltonien dans G, c'est à dire qu'il existe une succession de sommets $(v_1, v_2, \ldots, v_n, v_1)$. Par construction, il existe le cycle suivant dans G' : $(v_1, x, v_2, \ldots, v_n, v_1)$.

$\Leftarrow$ Supposons qu'il existe un cycle Hamiltonien dans G', c'est à dire qu'il existe une succession de sommets $(v_1, x, v_2, \ldots, v_n, v_1)$. Comme v_1 et v_2 sont voisins, on peut reconstruire la solution suivante : $(v_1, v_2, \ldots, v_n, v_1)$.

On a montré que Cycle Hamiltonien Biparti $\propto$ Cycle Hamiltonien, donc si un de ces deux problèmes est $\mathcal{NP}$-complet, l'autre le sera aussi.

Comme un Chaîne Hamiltonienne Biparti est un graphe non-orienté, la réduction à Cycle Hamiltonien est déjà trouvée, c'est la même que la réduction de Chaîne Hamiltonienne à Cycle

Hamiltonien, donc on peut dire que Chaîne Hamiltonienne Biparti $\propto$ Cycle Hamiltonien. De plus, comme Cycle Hamiltonien se réduit à Cycle Hamiltonien Biparti, on a Chaîne Hamiltonienne Biparti $\propto$ Cycle Hamiltonien Biparti. Prouvons maintenant que Cycle Hamiltonien Biparti $\propto$ Chaîne Hamiltonienne. La réduction est la même qu'entre Cycle Hamiltonien et Chaîne Hamiltonienne. Cycle Hamiltonien Biparti est donc bien polynomialement équivalent à Chaîne Hamiltonienne Biparti, et si l'un des deux problèmes est $\mathcal{NP}$-complet, alors l'autre le sera aussi.

On a montré que tous ces problèmes sont dans $\mathcal{NP}$, et qu'ils sont tous polynomialement équivalents (ils peuvent tous se réduire entre eux). Ainsi, si un de ces problèmes est $\mathcal{NP}$-complet, alors ils le seront tous.

Exercice 65 - Quelques réductions classiques

Montrer que Recouvrement de sommets est $\mathcal{NP}$-complet (preuve à partir de 3-Satisfaisabilité).

Cette réduction est montrée en détails dans l'exercice 62.

Montrer que Clique est $\mathcal{NP}$-complet (preuve à partir de 3-Satisfaisabilité).

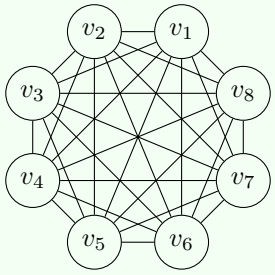
Pour commencer, un petit rappel s'impose sur le problème de Clique :

Clique

Entrée : $G = (V, E)$ un graphe et $k \in \mathbb{N}$

Question : Existe-t-il un sous-graphe complet dans G de taille k ?

Par exemple, le graphe suivant est une clique de taille 8 :



Pour montrer que Clique est $\mathcal{NP}$-complet, on va montrer que Clique est dans $\mathcal{NP}$, et que 3-Satisfaisabilité se réduit à Clique.

— Clique est dans $\mathcal{NP}$: le certificat positif est l'ensemble des sommets. Il suffit de vérifier si, pour chaque sommet, il est relié à tous les autres sommets de l'ensemble solution. Le certificat est bien polynomial.

— Soit φ une formule en forme normale conjonctive, avec chaque clause contenant 3 littéraux (sans perte de généralité, si un littéral contient moins de 3 littéraux, il suffit d'appliquer la réduction de Sat à 3-Sat pour qu'elle ait 3 littéraux). C'est à dire que :

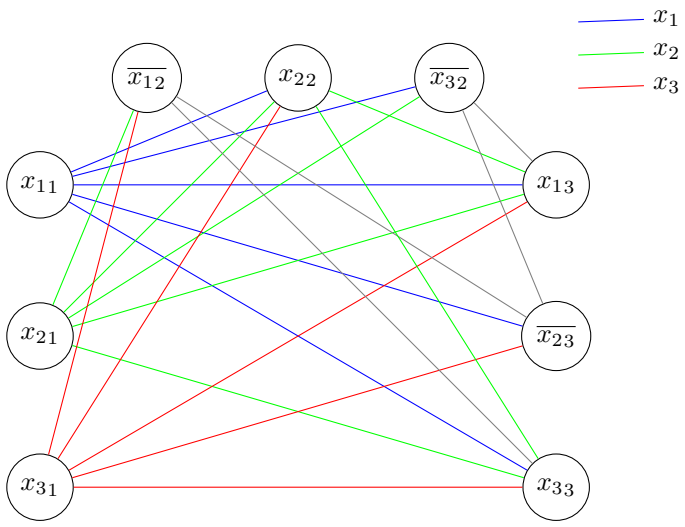
$$\varphi = \bigwedge_{i=1}^{m} (l_i^1 \vee l_i^2 \vee l_i^3)$$

On pose la réduction polynomiale entre φ une instance de 3-SAT et $G = (V, E)$ l'instance de CLIQUE suivante :

$$V = \bigcup_{i=1}^{m} \{l^1_{i\,i}, l^2_{i\,i}, l^3_{i\,i}\}$$

$$E = \bigcup_{i=1}^{m} \bigcup_{j=i+1}^{m} \left(\bigcup_{k=1}^{3} \{(l^k_{i\,i}, l^1_{j\,j}), (l^k_{i\,i}, l^2_{j\,j}), (l^k_{i\,i}, l^3_{j\,j})\} - \bigcup_{k=1}^{3} \{(l^k_{i\,i}, \overline{l^k_{i\,i}})\} \right)$$

Pour être moins formel, on crée 3 sommets par clause (1 par littéral dans la clause). Puis, pour chaque sommet, on les relie avec tous les sommets des autres clauses, excepté leur littéral opposé. Pour bien comprendre la réduction, faisons un exemple. Soit $\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3)$. Cette formule est clairement satisfiable, on peut trouver, par exemple, l'assignement α suivant qui satisfait φ : $\alpha(x_1) = \alpha(x_2) = \alpha(x_3) = \top$. Le graphe construit par la réduction est le suivant :



On voit bien qu'aucun x_i et $\overline{x_i}$ ne sont connectés. Sinon, chaque littéral est connecté à tous les littéraux des autres formules. On veut maintenant prouver la correction de la réduction. Pour ce faire, montrons qu'il existe une affectation qui satisfait φ si et seulement s'il existe une clique de taille m dans G (avec m le nombre de clauses de φ).

$\Rightarrow$ Supposons qu'il existe une affectation satisfiable de φ. Par construction, il existe au moins littéral de chaque clause connecté à un littéral de toutes les autres clauses (car étant connecté à tous les littéraux, sauf son opposé). Ainsi, la CLIQUE dans G est de taille au moins m. Montrons maintenant que la CLIQUE est au maximum de taille m. Le sommet d'un littéral n'est pas connecté aux sommets correspondant aux littéraux de sa propre formule, donc la taille de la CLIQUE ne peut pas dépasser m. On a montré que si une formule est satisfiable, alors il existe une clique de taille m dans G.

$\Leftarrow$ Supposons qu'il existe une CLIQUE de taille m dans G. Montrons qu'il existe une affectation satisfiable de φ. Il suffit de prendre tous les sommets des littéraux dans la CLIQUE et d'affecter leur valeur à $\top$ pour que la formule soit satisfiable. Comme un sommet de chaque clause est à $\top$, la formule est satisfiable. On a montré que s'il existe une CLIQUE de taille m dans G, alors il existe une affectation satisfiable de φ.

On a trouvé une réduction polynomiale correcte, donc 3-SATISFAISABILITÉ $\propto$ CLIQUE. CLIQUE est $\mathcal{NP}$-dur, et CLIQUE dans $\mathcal{NP}$, donc CLIQUE est bien $\mathcal{NP}$-complet.

Soit le problème

2-Partition à valeurs Paires

Entrée : Un ensemble fini A et une fonction τ de A dans $\mathbb{N}$ telle que $\forall e \in A.\tau(e)$ est pair.

Question : Est-il possible de les partager en deux sous-ensembles de même poids total P ?

Prouver que le problème 2-Partition à valeurs Paires est $\mathcal{NP}$-complet.

— 2-Partition à valeurs Paires est dans $\mathcal{NP}$. En effet, le certificat polynomial est la solution : les 2 partitions S_1 et S_2. Il suffit de vérifier que $S_1 \cap S_2 = \emptyset$ et que $\sum_{s \in S_1} \tau(s) = \sum_{s' \in S_2} \tau(s')$.

— Pour prouver que 2-Partition à valeurs Paires est $\mathcal{NP}$-dur, on va réduire 2-Partition à ce problème. Soit A l'ensemble de 2-Partition, et τ la fonction de A dans $\mathbb{N}$. On τ' la fonction de A dans $\mathbb{N}$ fournie à 2-Partition à valeurs Paires telle que :

$$\forall a \in A.\tau'(a) = 2 \times \tau(a)$$

Montrons que cette réduction est correcte. On veut prouver qu'il existe une 2-Partition de poids P si et seulement s'il existe une 2-Partition à valeurs Paires de poids $2 \times P$.

⇒ Supposons qu'il existe une 2-Partition de poids P, c'est à dire qu'il existe $S_1 \subset A$ et $S_2 \subset A$ tels que $S_1 \cap S_2 = \emptyset$ et $\sum_{s \in S_1} \tau(s) = \sum_{s' \in S_2} \tau(s') = P$. On a alors $\sum_{s \in S_1} \tau'(s) = \sum_{s' \in S_2} \tau'(s') = \sum_{s \in S_1} 2 \times \tau(s) = \sum_{s' \in S_2} 2 \times \tau(s') = 2 \times P$. Le poids de chaque élément est bien pair et le poids total de ces 2 sous-ensemble est bien $2 \times P$.

⇐ Supposons qu'il existe une 2-Partition à valeurs Paires de poids $2 \times P$, c'est à dire qu'il existe $S_1 \subset A$ et $S_2 \subset A$ tels que $S_1 \cap S_2 = \emptyset$ et $\sum_{s \in S_1} \tau'(s) = \sum_{s' \in S_2} \tau'(s') = 2 \times P$. La solution de 2-Partition à valeurs Paires est la même que 2-Partition.

On a montré que 2-Partition à valeurs Paires est dans $\mathcal{NP}$, et que ce même problème est $\mathcal{NP}$-dur, 2-Partition à valeurs Paires est donc bien $\mathcal{NP}$-complet.

On rappelle l'énoncé du problème Somme de sous ensemble.

Somme de sous ensemble

Entrée : Un ensemble $E = \{e_1 \dots e_n\}$, une fonction $\tau : E \to \mathbb{N}$ et un entier T.

Question : Existe-t-il un sous-ensemble $E' \subseteq E$ tel que $T = \sum_{e \in E'} \tau(e)$?

Prouver, à partir de Partition que Somme de sous ensemble est $\mathcal{NP}$-complet.

— Somme de sous ensemble est dans $\mathcal{NP}$: le certificat polynomial est l'ensemble solution E'.

— Montrons que Somme de sous ensemble est $\mathcal{NP}$-dur en réduisant 2-Partition à ce problème. La réduction est la suivante : on pose $T = \frac{1}{2} \sum_{e \in E} \tau(e)$. Prouvons la correction de cette réduction, c'est à dire, prouvons qu'il existe deux partitions S_1 et S_2 de poids P à 2-Partition si et seulement s'il existe un sous-ensemble E' de poids T à Somme de sous ensemble.

⇒ Supposons qu'il existe une solution à 2-Partition, c'est à dire qu'il existe S_1 et S_2 tels que $\sum_{s \in S_1} \tau(s) = \sum_{s' \in S_2} \tau(s') = T$. En prenant S_1 ou S_2, on a clairement une solution à Somme de sous-ensemble.

⇐ Supposons qu'il existe une solution à Somme de sous ensemble, c'est à dire qu'il existe $E' \subset E$ tel que $\sum_{e \in E'} \tau(e) = T$. On a clairement une solution à 2-Partition : $S_1 = \{e \mid e \in E'\}$ et $S_2 = \{e \mid e \in E \wedge e \notin E'\}$.

Le professeur Ykcid prétend avoir un algorithme polynomial pour construire, quand elle existe, une solution au problème Somme de sous ensemble pour tout ensemble quand le poids τ de chaque élément est impair (et ce quelle que soit la valeur T).
Prouvez lui que soit il a démontré que $\mathcal{P} = \mathcal{NP}$, soit il s'est trompé.
Indication : on pourra considérer la valeur $S_E = \sum_{e \in E} \tau(e)$, la valeur S_{imp} qui sera le plus petit nombre impair strictement supérieur à $\sum_{e \in E} \tau(e)$, et une fonction $\tau' = 2\tau + S_{imp}$. Si un sous ensemble $E' \subseteq E$ est tel que $\sum_{e \in E'} \tau(e) = \frac{S_E}{2}$ et si $|E'| = k$, quelle est la valeur de $\sum_{e \in E'} \tau'(e)$?

— Somme de sous ensemble à valeur impaires est dans $\mathcal{NP}$. En effet, on prend en certificat polynomial la valeur de la solution.

— On cherche à montrer que Somme de sous ensemble à valeur impaires est $\mathcal{NP}$-dur. On va montrer la correction de la réduction donnée de 2-Partition à Somme de sous ensemble à valeur impaires, c'est à dire qu'on va prouver qu'il existe deux partitions S_1 et S_2 de poids P à 2-Partition si et seulement s'il existe un sous-ensemble E' de poids $S_E + kS_{imp}$ à Somme de sous ensemble à valeur impaires.

⇒ On suppose qu'il existe une solution à 2-Partition, c'est à dire qu'il existe S_1 et S_2 tels que $\sum_{s \in S_1} \tau(s) = \sum_{s' \in S_2} \tau(s') = \frac{S_E}{2}$. En prenant S_1 ou S_2, on a clairement une solution à Somme de sous-ensemble à valeur impaires. En effet, toutes les valeurs sont impaires (car S_{imp} est impair, et l'addition d'un pair et d'un impair est impair) et on a bien $\sum_{e' \in S_1} \tau'(e') = S_E + kS_{imp}$.

⇐ On suppose qu'il existe une solution à Somme de sous ensemble à valeur impaires, c'est à dire qu'il existe $E' \subset E$ tel que $\sum_{e' \in E'} \tau'(e') = S_E + kS_{imp}$. Cette solution est aussi solution de 2-Partition.

Comme le problème Somme de sous ensemble à valeur impaires est dans $\mathcal{NP}$, et que ce problème est $\mathcal{NP}$-dur, soit le professeur Ykcid s'est trompé, soit il a trouvé un algorithme polynomial pour un problème $\mathcal{NP}$-complet, et dans ce cas, $\mathcal{P} = \mathcal{NP}$.

Montrer que Partition en triangles est $\mathcal{NP}$-complet (preuve à partir de Exact Cover by 3-SET).

Couverture Exacte d'ensemble

Entrée : Un ensemble fini X avec $|X| = 3q$ et une collection C de sous-ensembles de trois éléments de X.

Question : Est-ce que C contient une couverture exacte de X, i.e. une sous-collection $C' \subseteq C$ telle que chaque élément de X apparaît exactement une et une seule fois dans C' ?

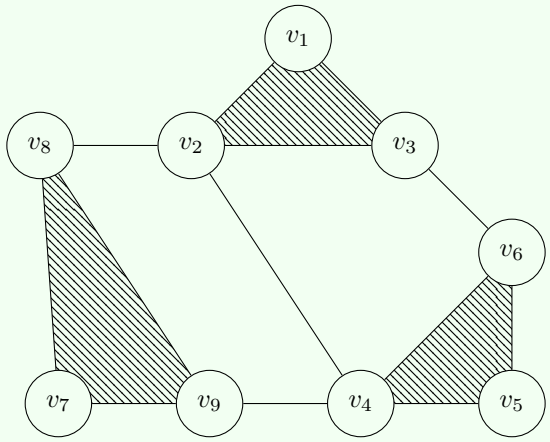
Partition en Triangles

Entrée : Un graphe $G = (V, E)$ avec $|V| = 3q, q \in \mathbb{N}$.

Question : Est-ce qu'il existe une partition de V en q ensembles disjoints $V_1, V_2, \ldots, V_q$ de trois sommets chacun tel que pour chaque $V_i = \{v_{i_1}, v_{i_2}, v_{i_3}\}$ les trois arêtes appartiennent à E ? Autrement dit peut-on couvrir le graphe par des triangles de taille trois ?

Trouvons des exemples pour illustrer les problèmes précédents, pour bien les comprendre. Soit $X = \{1, 2, 3, 4, 5, 6\}$ une instance de Exact-Cover, et $C = \{\{1, 2, 3\}, \{2, 3, 4\}, \{1, 2, 5\}, \{2, 5, 6\}, \{1, 5, 6\}\}$. Si on prend $C' = \{\{2, 3, 4\}, \{1, 5, 6\}\}$, alors on a une couverture exacte : $\bigcup_{c' \in C'} c' = X$ et $\bigcap_{c' \in C'} c' = \emptyset$. Si on avait pris $C' = \{\{1, 2, 3\}, \{2, 4, 5\}, \{2, 5, 6\}\}$, C' n'est pas une couverture exacte, car bien que $\bigcup_{c' \in C'} c' = X$, on a $\bigcap_{c' \in C'} c' \neq \emptyset$.

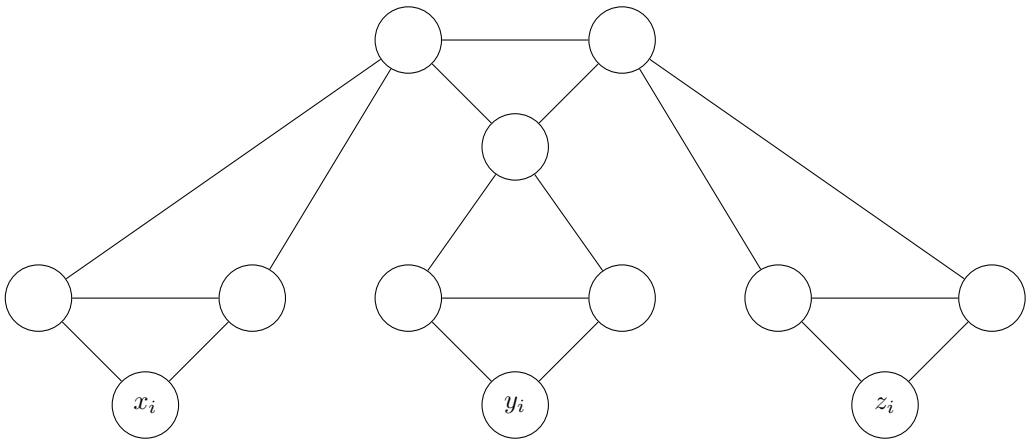
Passons maintenant à Partition en Triangles. Prenons le graphe suivant :



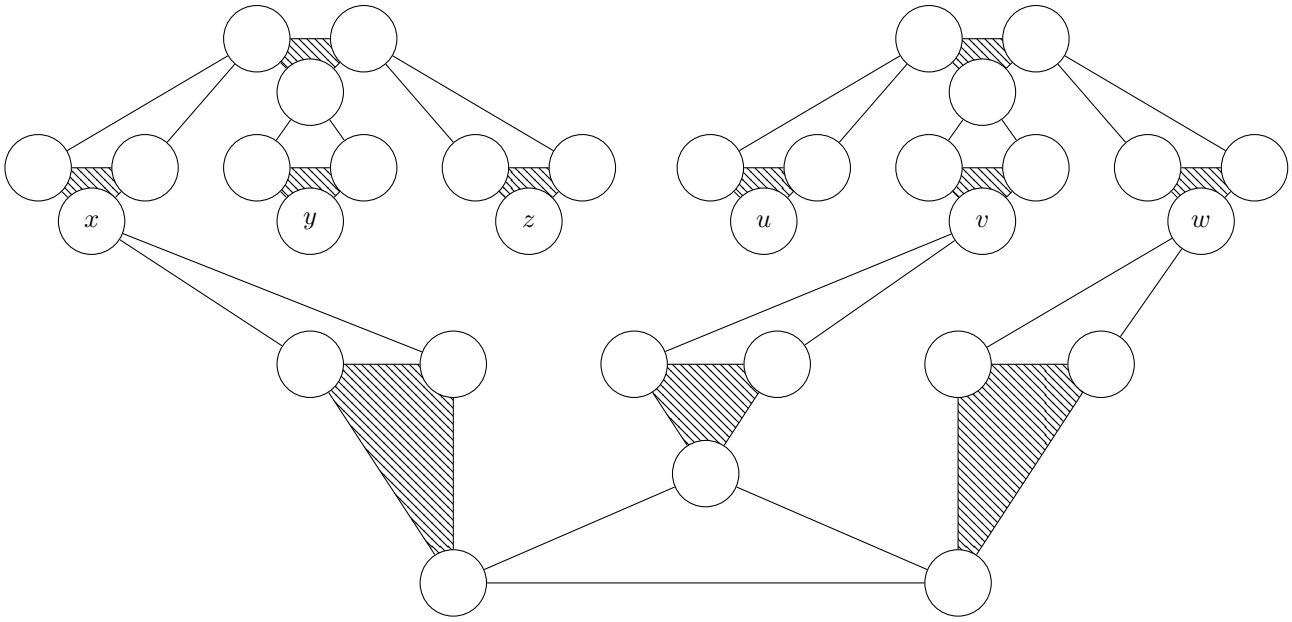
Si on pose les zones grillagées $V_1 = \{v_1, v_2, v_3\}, V_2 = \{v_4, v_5, v_6\}$ et $V_3 = \{v_7, v_8, v_9\}$, on voit bien que $\bigcup_{1 \leq i \leq 3} V_i = V$ et $\bigcap_{1 \leq i \leq 3} V_i = \emptyset$.

— Partition en Triangles est dans $\mathcal{NP}$. Le certificat polynomial positif est l'ensemble des triplets. Il suffit de vérifier que les triplets sont disjoints deux à deux, et qu'il y ait bien des arêtes entre chaque sommet.

— Pour montrer que Partition en Triangles est $\mathcal{NP}$-complet, on réduit Couverture Exacte d'ensemble à Partition en Triangles. Soit C une instance de Couverture Exacte d'ensemble. Soit $G = (V, E)$ l'instance de Partition en Triangles. On obtient G à partir de C grâce à la réduction suivante : pour tout $C_i = \{x_i, y_i, z_i\}$, on construit le gadget suivant :



Plutôt qu'essayer de montrer formellement la correction, je vais faire un dessin avec $X = \{x, y, z, u, v, w\}$ et $C = \{\{x, y, z\}, \{u, v, w\}, \{x, v, w\}\}$:



Dans cette configuration, on doit clairement prendre $\{x, y, z\}$ et $\{u, v, w\}$ pour avoir notre solution. Donnons intuitivement les preuves :

$\Rightarrow$ Supposons qu'on a une solution à Exact-Cover. Ainsi, il existe un nombre fini de gagdgets similaires à ceux de $\{x, y, z\}$ et $\{u, v, w\}$, c'est à dire dont on prend le triangle supérieur et les triangles liés aux littéraux en tant que solution. On peut trouver des triangles pour tous les autres triplets similaires à celui de $\{x, v, w\}$ dans l'exemple ci-dessus. S'il y a une solution à Exact-Cover, alors il y a une solution à Partition en Triangles.

$\Leftarrow$ Supposons qu'on a une solution à Partition en Triangles. Pour reconstruire une solution à Exact-Cover, on prend tous les littéraux dont le triangle supérieur du gadget du triplet est dans la partition. S'il y a une solution à Partition en Triangles, alors il y en a une pour Exact-Cover.

Partition en Triangles est dans $\mathcal{NP}$, et on a montré (pas du tout formellement) que Exact-Cover se réduit à ce problème. Ainsi, ce problème est $\mathcal{NP}$-dur. Donc le problème est $\mathcal{NP}$-complet.

Exercice 66 - Quelques réductions classiques

Nous supposerons que les problèmes suivants sont $\mathcal{NP}$-complets.

Circuit Hamiltonien

Entrée : Un graphe orienté $G = (V, E)$

Question : Existe-t-il un cycle qui visite chaque sommet exactement une fois ?

Montrer que les problèmes suivants sont $\mathcal{NP}$-complets :
Recouvrement de sommets à partir de 3-Satisfaisabilité

Cette réduction est montrée en détails dans l'exercice 62.

Voyageur de Commerce à partir de Cycle Hamiltonien.

Cette réduction est montrée en détails dans l'exercice 61.

Le problème du Chemin Hamiltonien.

Cette réduction est montrée en détails dans l'exercice 64.

Le problème du Plus Long Chemin

Plus Long Chemin

Entrée : Un graphe orienté $G = (V, E)$ et $L \in \mathbb{N}$.

Question : Existe-t-il un chemin simple (sans boucle) de s à t avec au moins L arêtes ?

— Ce problème est dans $\mathcal{NP}$. En effet, avec comme certificat l'ensemble des sommets qui composent le chemin (ordonnés correctement), il suffit de vérifier si la taille de cet ensemble est bien L et que c'est bien un chemin. Ce certificat est polynomial (en $O(|V|)$).

— On peut très clairement restreindre ce problème à Chemin Hamiltonien. En effet, il suffit juste de donner $L = |V| - 1$:

⇒ On suppose qu'il y a une solution à Chemin Hamiltonien, c'est à dire qu'il y a un chemin entre s et t qui passe par tous les sommets du graphe (la taille de ce chemin est donc $|V| - 1$). Il y a donc clairement un chemin simple de s à t, avec exactement $|V| - 1$ arêtes.

⇐ On suppose qu'il y a une solution à Plus Long Chemin de taille L. Comme $L = |V| - 1$ et que le chemin est sans répétition, c'est clairement un Chemin Hamiltonien.

— Plus Long Chemin est dans $\mathcal{NP}$, et est $\mathcal{NP}$-dur car Chemin Hamiltonien se réduit à ce problème avec une réduction constante. Ainsi, Plus Long Chemin est $\mathcal{NP}$-complet.

Le problème du Sac à Dos

Sac à Dos

Entrée : Un ensemble X d'objets à valeurs dans $\mathbb{N}$ et, pour chaque objet x, un volume $c_x \in \mathbb{N}$ et une utilité $a_x \in \mathbb{N}$; une capacité $b \in \mathbb{N}$ et une valeur $v \in \mathbb{N}$.

Question : Existe-t-il un sous-ensemble d'objets X' tel que $\sum_{x \in X'} c_x \leq b$ et $\sum_{x \in X'} a_x \geq v$?

— Ce problème est dans $\mathcal{NP}$. On peut donner le certificat suivant : l'ensemble X'. Comme $X' \subseteq X$, celui-ci est polynomial en la taille de la donnée entrée, et la vérification se fait en temps linéaire (il suffit de vérifier que la somme des poids est bien inférieure ou égale à b et que la somme des valeurs est bien supérieure ou égale à v).

— Il suffit de restreindre Partition à ce problème. En effet, soit $\langle X, \tau \rangle$ une instance de Partition. On applique la réduction suivante :

$$\forall x \in X . c_x = a_x = \tau(x)$$
$$b = v = \frac{1}{2} \sum_{x \in X} \tau(x)$$

Cette réduction est bien polynomiale : elle est linéaire en la taille de l'entrée. Prouvons la correction de cette réduction, c'est à dire montrons qu'il existe une solution à Partition si et seulement s'il existe une solution à Sac à Dos :

⇒ Supposons qu'on a une solution à Partition, c'est à dire qu'on a deux sous-ensembles S_1 et S_2 tels que $S_1 \cap S_2 = \emptyset$, $S_1 \cup S_2 = X$ et $\sum_{s_1 \in S_1} \tau(s_1) = \sum_{s_2 \in S_2} \tau(s_2) = S$. Comme $S = b = v$, en prenant S_1 ou S_2, on a une solution à Sac à Dos : $\sum_{s_i \in S_i} \tau(s_i) = \sum_{s_i \in S_i} a_{s_i} = \sum_{s_i \in S_i} c_{s_i} = b = v$ pour $i = 1, 2$. On a bien $S \leq b$ et $S \geq v$.

⇐ Supposons qu'on a une solution à Sac à Dos. C'est à dire qu'on a trouvé un ensemble X' tel que $\sum_{x \in X'} a_x = \sum_{x \in X'} c_x = b = v$. Posons $S_1 = X'$ et $S_2 = \{x \mid x \in X \wedge x \notin S_1\}$. On a bien une solution à Partition.

— Sac à Dos est dans $\mathcal{NP}$, et est $\mathcal{NP}$-dur car Partition, qui est $\mathcal{NP}$-complet, se réduit à ce problème avec une réduction linéaire. Ainsi, Sac à Dos est $\mathcal{NP}$-complet.

Le problème d'un Plus Court Chemin Sous Contrainte

Plus Court Chemin Sous Contrainte

Entrée : Soit un graphe $G = (V, E)$, deux entiers $c, \tau \in \mathbb{N}$, une longueur d'arc $c_{ij}, \forall (i,j) \in E$ et un temps de traversée $\tau_{ij}, \forall (i,j) \in E$.

Question : Existe-t-il dans le graphe G un chemin d'un sommet s à un sommet t dont la distance entre les sommets est au plus c et dont le temps de traversée est d'au plus τ ?

Idée : Utiliser le problème du Sac à Dos.

— Ce problème est dans $\mathcal{NP}$. En effet, avec le chemin solution comme certificat polynomial, il est facile de vérifier que c'est une solution au problème du Plus Court Chemin Sous Contrainte, il suffit de faire la somme des longueurs d'arcs et du temps de traversée.

— Pour prouver que ce problème est $\mathcal{NP}$-dur, on réduit le problème du Sac à Dos à celui-ci. Tout d'abord, on remarque que les deux contraintes de Plus Cour Chemin Sous Contrainte sont de la forme $\leq$. Soit l'instance $\langle X, b, v \rangle$ de Sac à Dos. On veut trouver un ensemble X' tel que $\sum_{x \in X'} c_x \leq b$ et $\sum_{x \in X'} a_x \geq v$. Ainsi, il va falloir transformer la contrainte de valeur pour qu'elle soit inférieure à τ. On sait que :

$$a \geq b \Leftrightarrow -a \leq -b$$

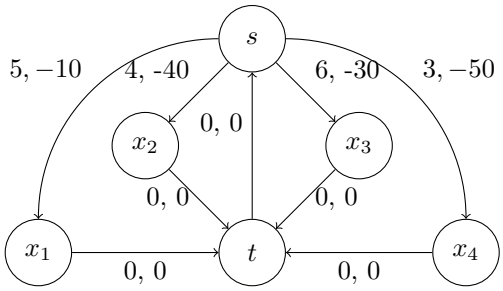
Ainsi, on pose $\langle G = (V, E), c_{xy}, \tau_{xy} c, \tau \rangle$ l'instance de Plus Court Chemin Sous Contrainte, construite à partir de Sac à Dos suivante :

$$\begin{aligned} V &= X \cup \{s, t\} \\ E &= \{(s, x) \mid x \in X\} \cup \{(t, s)\} \\ c_{xy} &= c_y \\ \tau_{xy} &= -a_y \\ c &= b \\ \tau &= -v \end{aligned}$$

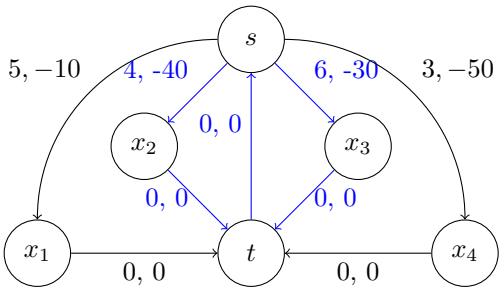
En clair, on crée un graphe avec une source s et un puit t. Chaque sommet correspond à un élément de l'ensemble, et les arêtes pour arriver à ce sommet ont pour distance le poids de cet objet et pour valuation l'opposé de leur valeur a. Faisons un exemple pour bien comprendre la réduction :

$$X = \{(x_1, 5, 10), (x_2, 4, 40), (x_3, 6, 30), (x_4, 3, 50)\}$$

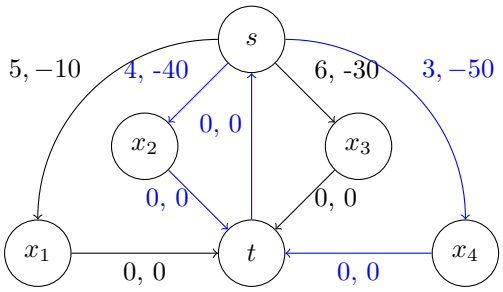
Cet ensemble est composé de triplet (élément, poids, valeur). Pour $b = 10$ et $v = 70$, on se retrouve avec $X' = \{x_2, x_3\}$ ou $X' = \{x_3, x_4\}$ ou $X' = \{x_2, x_4\}$. Transformons maintenant cette instance de Sac à Dos en instance du Plus Court Chemin Sous Contrainte :



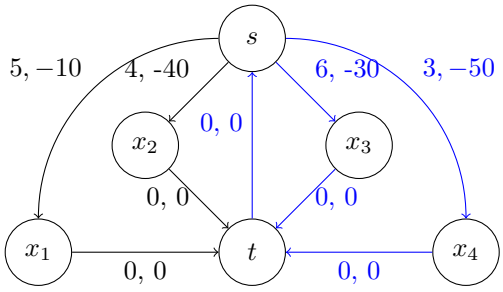
On voit bien que les seuls chemins entre s et t qui satisfont $c \leq 10$ et $v \leq -70$ sont les suivants :



Chemin $\{s, x_2, t, s, x_3, t\}$
$c = 10$ et $v = -70$.



Chemin $\{s, x_2, t, s, , x_4, t\}$
$c = 7$ et $v = -90$.



Chemin $\{s, x_3, t, s, x_4, t\}$
$c = 8$ et $v = -80$.

Montrons que cette réduction est correcte, c'est à dire, prouvons qu'il existe une solution à Sac à Dos si et seulement s'il existe une solution à Plus Court Chemin Sous Contrainte :

⇒ Supposons qu'il existe une solution à Sac à Dos. C'est à dire qu'il existe un ensemble X' tel que $\sum_{x \in X'} c_x \leq b$ et $\sum_{x \in X'} a_x \geq v$. On peut construire une solution de Plus Court Chemin Sous Contrainte : $\{s\} \cup \bigcup_{x \in X'} \{x, t, s\}$, en supprimant le dernier s de la solution. Pour chaque $x \in X'$, on aura (s, x) qui ajoute c_x à la distance totale et $-a_x$ au temps de traversée. Pour toutes les autres arêtes, on aura $0, 0$ ajoutés à ces deux valeurs, on aura donc bien une distance totale égale au poids ($\leq b$) et un temps de traversée total égal à l'opposé de la valeur totale ($\leq -v$).

⇐ Supposons qu'il existe une solution S à Plus Court Chemin Sous Contrainte. Pour reconstruire une solution à Sac à Dos depuis cette solution, il suffit de prendre $X' = S - \{s, t\}$. Comme toutes les arêtes vers s et t sont valuées à $(0, 0)$, on a évidemment $S_b \leq b$ et $-S_v \geq v$.

— Plus Court Chemin Sous Contrainte est dans $\mathcal{NP}$, et comme Sac à Dos $\propto$ Plus Court Chemin Sous Contrainte, ce problème est aussi $\mathcal{NP}$-dur. Ainsi, ce problème est $\mathcal{NP}$-complet.

Le problème de Steiner

Steiner

Entrée : Soit un graphe $G = (V, E)$, et soit $R \subseteq V$, $k \in \mathbb{N}$.

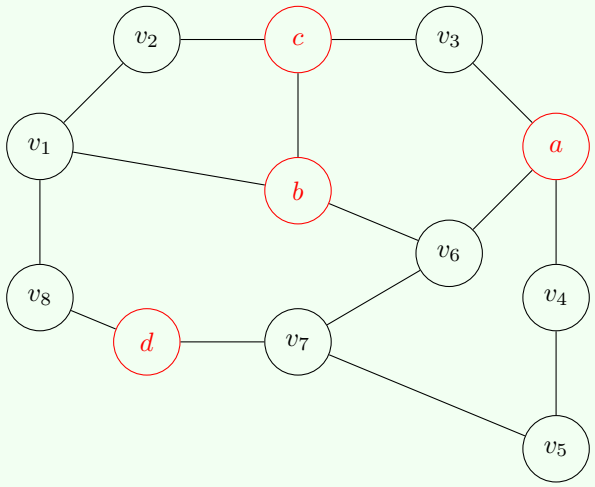
Question : Existe-t-il un arbre couvrant de R dans G avec au plus k arêtes ?

(a) Que retrouvez-vous dans le cas où $R = \{i, j\}$, $i \neq j$?

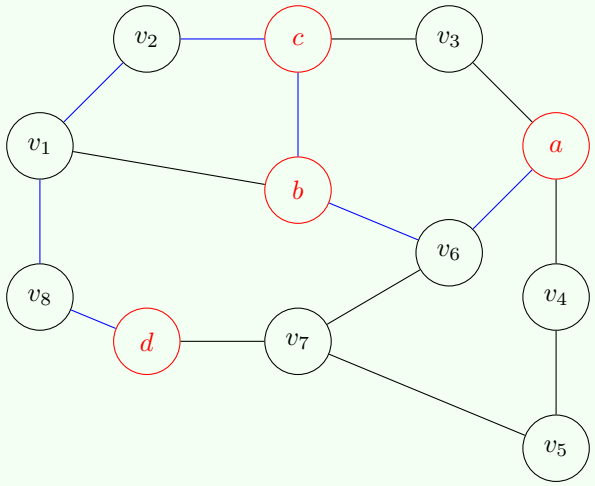
(b) De même pour $R = \{1, 2, \ldots, n\}$?

(c) Montrer que le problème de Steiner est $\mathcal{NP}$-complet.

Pour bien comprendre le problème, posons l'instance suivante : $k = 7$ et $R = \{a, b, c, d\}$, avec G :



Les sommets à atteindre sont en rouge. On peut envisager plusieurs chemins de taille inférieure ou égale à 7, par exemple :



Cette couverture est de taille 7. On voit cependant bien qu'il y a d'autres solutions, plus optimales.

(a) Dans le cas où $R = \{i, j\}$ deux sommets distincts du graphe, il suffit de faire un plus court chemin

entre ces deux sommets (Dijsktra) pour trouver l'arbre de STEINER.

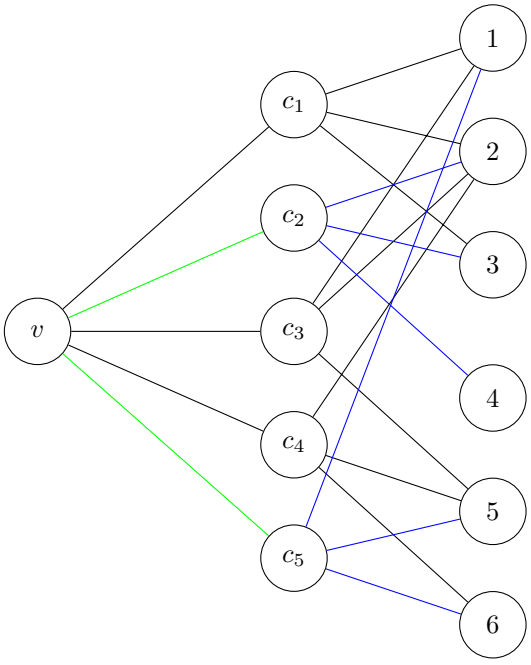
(b) Dans le cas où $R = \{1, 2, \ldots, n\}$, c'est à dire que $R = V$, alors l'arbre couvrant est V.

(c) Montrons que STEINER est $\mathcal{NP}$-complet. Tout d'abord, STEINER est dans $\mathcal{NP}$. Le certificat correspond simplement à la solution, c'est à dire à l'arbre de STEINER. Il est facile de vérifier que c'est un arbre qui contient tous les sommets de R et qu'il parcours au plus k arêtes.

Montrons maintenant que STEINER est $\mathcal{NP}$-dur, en réduisant COUVERTURE EXACTE D'ENSEMBLE à celui-ci. Soit $\langle X, C\rangle$ une instance de COUVERTURE EXACTE D'ENSEMBLE. On a $X = \{x_1, x_2, \ldots, x_{3q}\}$ et $C = \{C_1, \ldots, C_n\}$. Soit $\langle G = (V, E), R, k\rangle$ une instance de STEINER. On transforme $\langle X, C\rangle$ en $\langle G = (V, E), R, k\rangle$ de la manière suivante :

$$
\begin{aligned}
V &= \{v\} \cup \{c_1, \ldots, c_n\} \cup X \\
E &= \{(v, c_1), \ldots, (v, c_n)\} \cup \bigcup_{x_j \in c_i} \{(c_i, x_j)\} \\
R &= \{v, x_1, \ldots, x_{3q}\} \\
k &= 4q
\end{aligned}
$$

En clair, on crée un sommet source v qui relie tous les sommets qui représentent les ensemble dans C. Ensuite, pour chaque sommet représentant un ensemble, on le relie avec les 3 éléments qui le compose. Par exemple, soit $X = \{1, 2, 3, 4, 5, 6\}$ et $C = \{\{1, 2, 3\}, \{2, 3, 4\}, \{1, 2, 5\}, \{2, 5, 6\}, \{1, 5, 6\}\}$. On construit l'instance de STEINER suivante :



La solution à COUVERTURE EXACTE D'ENSEMBLE est $C' = \{\{2, 3, 4\}, \{1, 5, 6\}\}$. Ce sont les arêtes bleues du graphe ci-dessus. La solution à STEINER est l'union des arêtes vertes et bleues. En effet, avec ces arêtes, on a bien un arbre couvrant de R. Le nombre total d'arêtes est 8, qui est exactement $4 \times q$ car $q = 2$. On a vu la réduction sur un petit exemple, montrons maintenant qu'elle est correcte pour n'importe quelle instance de COUVERTURE EXACTE D'ENSEMBLE :

$\Rightarrow$ Supposons qu'on a une solution C' à COUVERTURE EXACTE D'ENSEMBLE. Comme $|X| = 3q$ et que $|C'_i| = 3, \forall i \leq q$, tous les sommets représentant des éléments de l'ensemble sont reliés aux q ensembles de C', avec un nombre total de $3q$ arêtes. De plus, v est relié à $C'_i, \forall i \leq q$, ce qui fait

q arêtes en plus. Il y a en tout $4q$ arêtes dans l'arbre couvrant. C'est donc bien une solution à Steiner.

$\Leftarrow$ Supposons qu'on a une solution à Steiner. C'est à dire qu'on a un ensemble de sommets X' qui donnent un arbre couvrant de R. Pour reconstruire une solution à Couverture Exacte d'ensemble, il suffit de prendre tous les sommets c_i et de leur faire correspondre un ensemble (pour chaque arête (c_i, x_j), ajouter x_j à C'_i).

On a montré que Couverture Exacte d'ensemble se réduit à Steiner. Ainsi, ce problème est $\mathcal{NP}$-dur. De plus, comme ce problème est dans $\mathcal{NP}$, il est $\mathcal{NP}$-complet.

Montrer que les problèmes suivants sont $\mathcal{NP}$-complets (utiliser le problème Chemin Hamiltonien). Soit un graphe $G = (V, E)$, un ensemble $L \subset V$, un entier k. Existe-t-il un arbre couvrant T de G tel que :

(a) L'ensemble des feuilles de T est L ?

(b) Il n'existe pas de feuilles de T à l'extérieur de L ?

(c) T admet k feuilles ?

(d) T admet au plus k feuilles ?

(e) T admet au moins k feuilles ? (utiliser le problème Ensemble Dominant Connecté, décrit dessous).

(f) Les sommets de T admettent un degré au plus k ?

Dans cette question, je vais juste donner l'intuition des réductions et de leur preuve, sans passer par le formel. La plupart des réductions sont abordées précédemment, ou bien sont immédiates par rapport à des problèmes déjà vus, ce n'est donc pas une question très intéressante.

(a) Ce problème est clairement dans $\mathcal{NP}$. Il suffit de prendre l'ensemble T_1 tous les sommets de degré 1 de T et de vérifier que $T_1 = L$.

Ensuite, on peut réduire Chemin Hamiltonien à ce problème. La réduction est triviale : on lui donne le même graphe que Chemin Hamiltonien, et pour L, on prend les sommets de degré 1 du graphe d'origine. On voit bien que s'il existe un Chemin Hamiltonien, c'est un arbre couvrant avec pour feuilles L, et que s'il existe un arbre couvrant, alors c'est un Chemin Hamiltonien.

On a trouvé une réduction polynomiale entre Chemin Hamiltonien et ce problème, de plus, il appartient à $\mathcal{NP}$, il est donc bien $\mathcal{NP}$-complet.

(b) C'est la même réduction, et la même preuve. On veut juste que les feuilles de T soit un sous-ensemble de L, donc on peut prendre n'importe quel ensemble L tant qu'on garde les sommets de degré 1.

(c) Ce problème est évidemment dans $\mathcal{NP}$. La réduction est pratiquement immédiate à partir des questions précédentes : il suffit de donner $k = 2$. Avec ceci, on peut réduire Chemin Hamiltonien à ce problème, et la preuve de correction est immédiate.

(d) Idem.

(e) Ce problème est totalement équivalent à Ensemble Dominant Connecté, je vous invite donc à aller voir la question suivante.

(f) Toujours en utilisant la même réduction de Chemin Hamiltonien, en restreignant k à 2, une instance de ce problème est exactement une instance de Chemin Hamiltonien.

Montrer que le problème Ensemble Dominant Connecté est $\mathcal{NP}$-complet (utiliser le problème Recouvrement de sommets).

Ensemble Dominant Connecté (Connected Dominating Set)

Entrée : $G = (V, E)$ un graphe non orienté, $k \in \mathbb{N}$

Question : Existe-t-il un ensemble dominant S dans G de taille au plus k tel que le sous-graphe de G induit par les sommets de S est connecté ?

Un ensemble dominant d'un graphe $G = (V, E)$ est un sous-ensemble $D \subseteq V$ tel que tout sommet qui n'appartient pas à D possède au moins une arête d'extrémité un sommet de D.

— Ensemble Dominant Connecté est dans $\mathcal{NP}$. Il est facile de vérifier que chaque sommet u de la solution est connecté à un autre sommet v de cette solution, il suffit de regarder si $(u, v) \in E$.

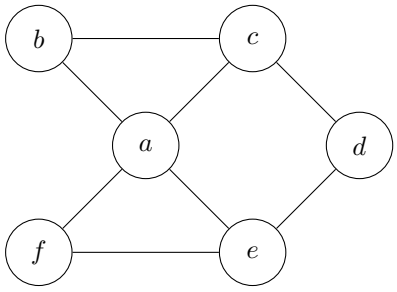
— Prouvons qu'Ensemble Dominant Connecté est $\mathcal{NP}$-dur. Pour ce faire, réduisons Recouvrement de sommets en Ensemble Dominant Connecté. L'idée est de construire des triangles pour chaque arête d'un graphe. Soit $\langle G = (V, E)\rangle$ l'instance de Recouvrement de sommets, transformons la en $\langle G' = (V', E'), k\rangle$ instance de Ensemble Dominant Connecté :

$$V' = \bigcup_{(u,v)\in E} \{uv\} \cup V$$

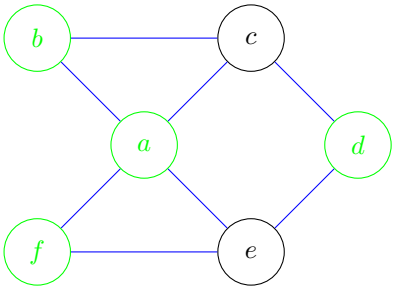
$$E' = \bigcup_{(u,v)\in E} \{(u, uv), (v, vu), (u, v)\} \cup \left(\bigcup_{u\in V} \bigcup_{\substack{v \in V \\ u \neq v}} \{(u, v)\} \right)$$

$$k = |V|$$

En clair, on prend le graphe complet de G et on ajoute des triangles. Prenons par exemple le graphe suivant :

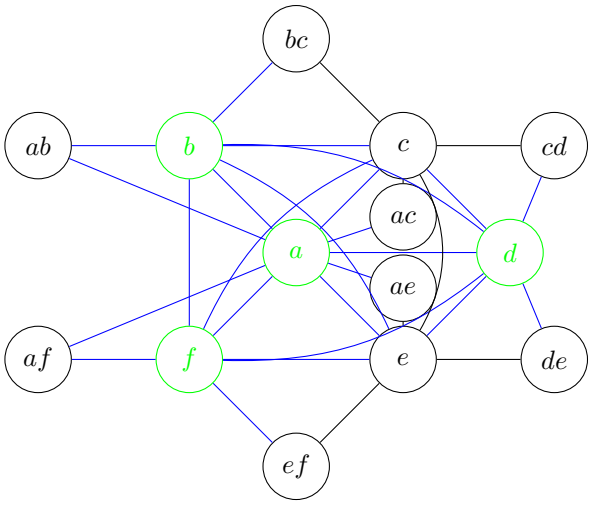


Graphe G.



Recouvrement de G.

Avec la transformation, on construit le graphe suivant :



Maintenant que nous avons l'intuition, montrons que la réduction est correcte :

⇒ Supposons qu'on a une solution S à Recouvrement de sommets de taille k. Cette solution est bien une solution de Ensemble Dominant Connecté. En effet, chaque sommet de G' est connecté avec un sommet de S, car s'il est dans V, alors comme au moins un des deux sommets de chaque arête est dans le Recouvrement, tout sommet de V est couvert par l'ensemble dominant. S'il est dans $V' - V$, c'est que le sommet est un de ceux ajouté pour faire un triangle. Or, comme un des deux sommets dans V du triangle est dans le recouvrement, le sommet triangle est aussi couvert par celui-ci dans l'ensemble dominant. Comme le sous-graphe V dans V' est complet, et que la solution ne possède que des sommets de V, S est connexe, c'est donc bien un Ensemble Dominant Connecté.

⇐ Supposons qu'on a une solution S pour Ensemble Dominant Connecté. Pour n'importe quel triangle (u, v, uv), le sommet uv est dominé par u ou v. De plus, comme uv n'est connecté qu'avec u et v, il faut qu'un des deux soit dans l'Ensemble Dominant Connecté pour qu'uv soit aussi dedans. Si on veut garder le nombre exact de sommets dans la solution, il suffit de remplacer uv par u ou v (celui qui n'a pas été pris dans la solution). Sinon, on peut tout simplement supprimer uv pour garder seulement les sommets de V, qui sera un Recouvrement de sommets : tous les uv doivent être dominés, donc soit u soit v doit être dans la solution, pour toutes les arêtes de E.

Cette preuve est écrite tard, après une (longue) journée de travail, signalez moi si cette preuve n'est pas claire, je pourrai la réécrire en étant plus lucide.

Montrer que le problème de 3-Satisfaisabilité Équilibré est $\mathcal{NP}$-complet.

3-Satisfaisabilité Équilibré

Entrée : Soit une formule logique dans laquelle chaque clause contient 3 littéraux et chaque variable apparaît une fois positivement et une fois négativement.

Question : Existe-t-il une affectation qui satisfasse toutes les clauses ?

Je doute que cette preuve soit possible, car ce problème se résout en temps polynomial : comme chaque littéral apparaît une fois positivement et négativement, il suffit d'appliquer la méthode de résolution, qui est déterministe dans ce cas, et on peut savoir si la formule est satisfiable ou non.

2.7.3 Autour des nombres

Exercice 67 - Partition

On considère que le problème Somme de sous ensemble est $\mathcal{NP}$-complet.

2-Partition

Entrée : Étant donnés n objects $a_i (1 \leq i \leq n)$ de poids entiers $p(a_1), p(a_2), \ldots, p(a_n)$.

Question : Est-il possible de les partager en deux sous-ensembles disjoints de même poids total P ?

Montrer à partir de Somme de sous ensemble que 2-Partition est $\mathcal{NP}$-complet.

2-Partition est dans $\mathcal{NP}$. En effet, avec comme certificat positif les partitions S_1 et S_2, on peut vérifier en temps linéaire que $S_1 \cap S_2 = \emptyset$ et $\sum_{s \in S_1} s = \sum_{s' \in S_2} s' = P$.

Montrons maintenant que 2-Partition est $\mathcal{NP}$-dur. Pour ce faire, soit $\langle X, T \rangle$ une instance de Somme de sous ensemble. Pour avoir une instance $\langle S \rangle$ de 2-Partition depuis $\langle X, T \rangle$, on applique la réduction suivante :

$$S = X \cup \{(\sum_{x \in X} x) - 2 \times T\}$$

Pour se convaincre de la réduction, prenons un exemple. Soit $\langle X, T \rangle = \langle \{3, 34, 4, 12, 5, 2\}, 9 \rangle$. On a donc :

$$\sum_{x \in X} x - 2 \times T = 60 - 18 = 42 \Rightarrow S = \{3, 34, 4, 12, 5, 2, 42\}$$

Cherchons deux partitions S_1 et S_2 de cet ensemble. On veut deux ensembles de somme totale 51 :

$$S_1 = \{5, 12, 34\} \qquad \sum_{s \in S_1} s = 5 + 12 + 34 = 51$$

$$S_2 = \{2, 3, 4, 42\} \qquad \sum_{s \in S_2} s = 2 + 3 + 4 + 42 = 51$$

On a bien une 2-Partition. Pour revenir à notre problème initial de Somme de sous ensemble, il suffit de supprimer 42 de S_2, et on trouve une solution $X' = \{2, 3, 4\}$: $\sum_{x' \in X'} x' = 9 = T$. Prouvons maintenant formellement que la réduction est correcte, c'est à dire prouvons qu'il existe X' tel que $\sum_{x' \in X'} x' = T$ de $\langle X, T \rangle$ une instance de Somme de sous ensemble si et seulement s'il existe S' une solution de 2-Partition :

$\Rightarrow$ Supposons qu'il existe une solution à Somme de sous ensemble. C'est à dire qu'il existe X_1 tel que $\sum_{x_1 \in X_1} x_1 = T$. Prenons l'ensemble $X_2 = \{x \mid x \in X \wedge x \notin X_1\}$. Soit $s = \sum_{x \in X} x$. Comme $\sum_{x_1 \in X_1} x_2 = T$, on a $\sum_{x_2 \in X_2} x_2 = s - T$. En ajoutant $s - 2 \times T$ à X_1, donc $X_1' = X_1 \cup \{s - 2 \times T\}$, la somme totale de cet ensemble sera $T + s - 2 \times T = s - T$. On a $\sum_{x_1' \in X_1'} x_1' = \sum_{x_2 \in X_2} x_2 = s - T$. En prenant par exemple $S' = X_1'$, on a bien une solution à 2-Partition.

$\Leftarrow$ Supposons qu'il existe une solution S' à 2-Partition. Supposons que $s - 2 \times T \in S'$ (sans perte de généralité, car il y a forcément un des deux ensembles qui contient cet entier). Il suffit de supprimer cet élément pour trouver une solution X' à Somme de sous ensemble. En effet, on a $\sum_{s' \in S'} s' = s - T$. En retirant $s - 2 \times T$, on a :

$$(\sum_{s' \in S'} s') - (s - 2T) = (s - T) - (s - 2T) = (s - s) + (2T - T) = T$$

La solution est donc bien $X' = S' - \{s - 2T\}$. On a bien une solution à Somme de sous ensemble.

On a montré que la réduction est correcte, ainsi, Somme de sous ensemble $\propto$ 2-Partition. Comme Somme de sous ensemble est $\mathcal{NP}$-complet, 2-Partition est $\mathcal{NP}$-dur. De plus, comme 2-Partition est dans $\mathcal{NP}$, ce problème est aussi $\mathcal{NP}$-complet.

Exercice 68 - Autour du problème de la 2-Partition

Nous rappelons que le problème suivant est $\mathcal{NP}$-complet.

2-Partition

Entrée : Étant donnés n objets a_i $(1 \leq i \leq n)$ de poids entiers $p(a_1), p(a_2), \ldots, p(a_n)$ de somme $2P$.

Question : Est-il possible de les partager en deux sous-ensembles de même poids total P ?

Montrer que les problèmes suivants sont $\mathcal{NP}$-complets.

2-Partition à valeurs paires

Entrée : Étant donnés n objets a_i $(1 \leq i \leq n)$ de poids entiers à valeurs paires $p(a_1), p(a_2), \ldots, p(a_n)$ de somme $2P$.

Question : Est-il possible de les partager en deux sous-ensembles de même poids P ?

Cette réduction est effectuée sur la question 3 de l'exercice 65.

2-Partition avec nombre pair

Entrée : Étant donnés $2n$ objets a_i $(1 \leq i \leq 2n)$ de poids entiers $p(a_1), p(a_2), \ldots, p(a_{2n})$ de somme $2P$.

Question : Est-il possible de les partager en deux sous-ensembles I_1, I_2 de même poids P ?

— Ce problème est dans $\mathcal{NP}$. En effet, avec les solutions I_1 et I_2, il suffit de vérifier que $I_1 \cap I_2 = \emptyset$, $I_1 \cup I_2 = X$ et $\sum_{i_1 \in I_1} i_1 = \sum_{i_2 \in I_2} i_2 = P$, ce qui se fait en temps polynomial.

— Ce problème est $\mathcal{NP}$-dur : 2-Partition se réduit à lui. Posons $\langle X, P, p \rangle$ une instance de 2-Partition. On obtient $\langle X', P', p' \rangle$ une instance de 2-Partition avec nombre pair grâce à la réduction suivante :

$$X' = X \cup \left(\bigcup_{n < i \leq 2n} \{x_i\} \right)$$

$$p' = \begin{cases} p(x_i) & \text{si } i \leq n \text{ (autrement dit, si } x_i \in X) \\ (2P+1) \times p(x_{i-n}) & \text{sinon} \end{cases}$$

$$P' = P + P(2P+1) = P(2P+2)$$

Cette réduction est polynomiale : elle ajoute n sommets, et n multiplications sont effectuées. Montrons que cette réduction est correcte, c'est à dire, prouvons qu'il existe une solution à 2-Partition si et seulement s'il existe une solution à 2-Partition avec nombre pair :

⇒ Supposons qu'il existe une solution à 2-Partition, c'est à dire qu'il existe I_1, I_2 tels que $\sum_{i_1 \in I_1} i_1 = \sum_{i_2 \in I_2} = P$. On pose I'_1 et I'_2 les solutions à 2-Partition avec nombre pair, construites de la manière suivante : $I'_1 = I_1 \cup \{x_{i+n} \mid x_i \in I_1\}$ et $I'_2 = I_2 \cup \{x_{i+n} \mid x_i \in I_2\}$. C'est bien une solution à 2-Partition avec nombre pair : on a $p'(I'_1) = p(I_1) + (2P+1)p(I_1) = P + (2P+1)P = P(2P+2)$ et $p'(I'_2) = p(I_2) + (2P+1)p(I_2) = P + (2P+1)P = P(2P+2)$. On a montré que si on a une solution à 2-Partition, alors on a une solution à 2-Partition avec nombre pair.

⇐ Supposons qu'il existe une solution à 2-Partition, c'est à dire qu'il existe I_1, I_2 tels que $\sum_{i_1 \in I_1} p'(i_1) = \sum_{i_2 \in I_2} p'(i_2) = 2P$. Pour reconstruire une solution à 2-Partition, il suffit de prendre tous les objets de poids supérieur à $2P$ et de les supprimer. La solution donnée sera forcément solution à 2-Partition : pour atteindre $P + (2P+1)P$, et que tous les éléments qui ne sont pas dans X ont

un poids supérieur à $2P$, le poids P des deux côtés vient des éléments de X. On a montré que si on a une solution à 2-Partition avec nombre pair, alors on a une solution à 2-Partition.

— 2-Partition avec nombre pair est dans $\mathcal{NP}$, et 2-Partition se réduit à ce problème, donc 2-Partition avec nombre pair est $\mathcal{NP}$-complet.

2-Partition équilibré

Entrée : Étant donnés $2n$ objets a_i $(1 \leq i \leq 2n)$ de poids entiers $p(a_1), p(a_2), \ldots, p(a_{2n})$ de somme $2P$.

Question : Est-il possible de les partager en deux sous-ensembles I_1, I_2 de même poids P tel que $|I_1| = n$?

— 2-Partition équilibré est dans $\mathcal{NP}$. Il suffit de vérifier que la somme et la taille des deux ensembles solution est identique.

— Ce problème est $\mathcal{NP}$-dur : 2-Partition se réduit à lui. Posons $\langle X, P, p\rangle$ une instance de 2-Partition. On obtient $\langle X', P', p'\rangle$ une instance de 2-Partition équilibré grâce à la réduction suivante :

$$X' = X \cup \left(\bigcup_{n<i\leq 2_n} \{x_i\} \right)$$

$$p' = \begin{cases} p(x_i) & \text{si } i \leq n \\ 0 & \text{sinon} \end{cases}$$

$$P' = P$$

Cette réduction est polynomiale : elle ajoute n éléments à un ensemble. Montrons que cette réduction est correcte, c'est à dire, prouvons qu'il existe une solution à 2-Partition si et seulement s'il existe une solution à 2-Partition équilibré :

⇒ Supposons qu'il existe une solution à 2-Partition, c'est à dire qu'il existe I_1, I_2 tels que $\sum_{i_1 \in I_1} i_1 = \sum_{i_2 \in I_2} = P$. On pose $I'_1 = I_1$ et $I'_2 = I_2$ les solutions à 2-Partition équilibré. Comme tous les éléments de $X' \setminus X$ ont pour poids 0, il suffit d'équilibrer le nombre d'éléments des ensembles I'_1 et I'_2 : on ajoute $n - |I_1|$ éléments de poids 0 à I'_1 et $n - |I_2|$ éléments de poids 0 à I'_2. On a $|I'_1| = |I_1| + n - |I_1| = n$ et $|I'_2| = |I_2| + n - |I_2| = n$. Le poids de ces deux ensembles ne change pas. On a donc bien une solution à 2-Partition équilibré.

⇐ Supposons qu'il existe une solution à 2-Partition équilibré. On peut reconstruire une solution à 2-Partition en enlevant tous les éléments de poids 0 (qui ne sont pas dans X). C'est évidemment une solution de 2-Partition.

— 2-Partition équilibré est dans $\mathcal{NP}$, et 2-Partition se réduit à ce problème, et donc il est $\mathcal{NP}$-dur. Ainsi, ce problème est bien $\mathcal{NP}$-complet.

2-Partition Paire/Impaire

Entrée : Étant donnés $2n$ objets a_i $(1 \leq i \leq 2n)$ de poids entiers $p(a_1), p(a_2), \ldots, p(a_{2n})$ de somme $2P$.

Question : Est-il possible de les partager en deux sous-ensembles I_1, I_2 de même poids P avec un entier entre a_{j-1} et a_{2j} $(1 \leq j \leq n)$ appartenant à I ?

— Ce problème est dans $\mathcal{NP}$. En effet, il suffit de vérifier que c'est une 2-Partition valide, et qu'il n'y a pas deux éléments d'indice $2i$ et $2i - 1$ dans la solution.

— Ce problème est $\mathcal{NP}$-dur : 2-Partition se réduit à lui. Posons $\langle X, p\rangle$ une instance de 2-Partition. On obtient $\langle X', p'\rangle$ une instance de 2-Partition Paire/Impaire grâce à la réduction suivante :

$$X' = X \cup \left(\bigcup_{n<i\leq 2_n} \{x_i\} \right)$$

$$p' = \begin{cases} 1 & \text{si } i = 2n \\ p'(x_{2i+1}) & \text{si } i \mod 2 = 0 \\ p'(x_{2i}) + p(x_i) & \text{sinon} \end{cases}$$

Posons un exemple pour bien comprendre. Soit $X = \{3, 4, 2, 1, 8\}$. On a bien 2 partitions possibles : $\{3, 4, 2\}$ et $\{1, 8\}$. On construit l'ensemble des poids de X' suivant :

$$p'(X') = \{19, 16, 16, 12, 12, 10, 10, 9, 9, 1\}$$

On prend en solution : $2i - 1$ si $i \in S$, $2i$ sinon. On se retrouve avec les deux ensembles suivants :

$$S_1 = \{19, 16, 12, 9, 1\}$$
$$S_2 = \{16, 12, 10, 10, 9\}$$

On voit bien que la somme des deux ensemble est égale (57). Maintenant que nous avons l'intuition, formalisons la correction :

⇒ Supposons qu'il existe une solution à 2-Partition, c'est à dire qu'il existe I_1, I_2 tels que $\sum_{i_1\in I_1} i_1 = \sum_{i_2\in I_2} = P$. On pose $I'_1 = I_1$ et $I'_2 = I_2$ les solutions à 2-Partition Paire/Impaire. On a $I'_1 = \{x'_{2i-1} \mid x_i \in I_1\} \cup \{x_{2i} \mid x_i \notin I_1\}$ et $I'_2 = \overline{I'_1}$. C'est bien une solution à 2-Partition Paire/Impaire. Deux éléments adjacents d'indice $2i$ et $2i - 1$ ne sont jamais dans une solution, et la somme des poids des éléments de I'_1 est égale à la somme des poids de I'_2.

⇐ Supposons qu'on a une solution à 2-Partition Paire/Impaire. Pour reconstruire une solution à 2-Partition, il suffit de prendre tous les i tels que x'_{2i-1} est dans la solution.

— 2-Partition Paire/Impaire est dans $\mathcal{NP}$, et 2-Partition se réduit à ce problème, donc il est $\mathcal{NP}$-dur. Ainsi, ce problème est bien $\mathcal{NP}$-complet.

3-Partition
Entrée : Étant donnés n objets a_i $(1 \leq i \leq n)$ de poids entiers $p(a_1), p(a_2), \ldots, p(a_n)$ de somme $3P$.
Question : Est-il possible de les partager en trois sous-ensembles de même poids P ?

Cette réduction est effectuée dans l'exercice 69.

Exercice 69 - Tri-partition

3-Partition
Entrée : Étant donnés n objets a_i $(1 \leq i \leq n)$ de poids entiers $(p(a_1), p(a_2), \ldots, p(a_n))$.
Question : Est-il possible de les partager en trois sous-ensembles de même poids total P ?

3-Partition est dans $\mathcal{NP}$. En effet, avec comme certificat positif les partitions S_1, S_2 et S_3, on peut vérifier en temps linéaire que $S_1 \cap S_2 = S_2 \cap S_3 = S_1 \cap S_3 = \emptyset$ et que $\sum_{s_1\in S_1} p(s_1) = \sum_{s_2\in S_2} p(s_2) = \sum_{s_3\in S_3} p(s_3) = P$.

Montrons maintenant que 3-Partition est $\mathcal{NP}$-dur. Soit $\langle X \rangle$ une instance de 2-Partition. Posons $S = \sum_{x \in X} x$. Soit $P = \frac{S}{2}$ (si S n'est pas pair, 2-Partition n'a pas de solution dans tous les cas). Soit $\langle X' \rangle$ une instance de 3-Partition. Pour obtenir X' à partir de X, on applique la réduction suivante :

$$X' = X \cup \{x\} \text{ avec } p(x) = P$$

Montrons que cette réduction est correcte, c'est à dire, montrons qu'il existe une solution à 2-Partition si et seulement s'il existe une solution à 3-Partition :

$\Rightarrow$ On suppose qu'il existe une solution à 2-Partition. Soit S_1 et S_2 ces deux partitions. On a $\sum_{s_1 \in S_1} p(s_1) = \sum_{s_2 \in S_2} p(s_2) = P$. On pose $S_3 = \{x\}$. Comme $p(x) = P$, on a bien $\sum_{s_1 \in S_1} p(s_1) = \sum_{s_2 \in S_2} p(s_2) = \sum_{s_3 \in S_3}$. De plus, comme $S_1 \cap S_2 = \emptyset$, $S_1 \cup S_2 = X$ et que $x \notin X$, on a bien $S_1 \cap S_3 = \emptyset$ et $S_2 \cap S_3 = \emptyset$. Ainsi, on a bien une solution à 3-Partition.

$\Leftarrow$ On suppose qu'il existe une solution à 3-Partition. Soit S_1, S_2 et S_3 ces partitions. On supposera, sans perte de généralité, que $S_3 = \{x\}$ (un des trois ensembles sera forcément composé seulement de x, sachant que $p(x) = P$). La solution a 2-Partition est S_1 et S_2. En effet, $S_1 \cap S_2 = \emptyset$ et $S_1 \cup S_2 = X$. De plus, $\sum_{s_1 \in S_1} p(s_1) = \sum_{s_2 \in S_2} p(s_2) = P$, c'est bien une solution à 2-Partition.

3-Partition est dans $\mathcal{NP}$. De plus, on a prouvé que 2-Partition $\propto$ 3-Partition. Ainsi, comme 2-Partition est $\mathcal{NP}$-complet, 3-Partition est $\mathcal{NP}$-dur. Comme de plus il est dans $\mathcal{NP}$, 3-Partition est $\mathcal{NP}$-complet.

Exercice 70 - Programmation dynamique : algorithme pseudo-polynomial

1. Sur le problème de la partition :

   2-Partition

   Entrée : Étant donnés n objets $a_i (1 \leq i \leq n)$ de poids entiers $p(a_1), p(a_2), \ldots, p(a_n)$.

   Question : Est-il possible de les partager en deux sous-ensembles de même poids total P ?

   (a) Nous allons plonger le problème dans une classe de problèmes dépendant de paramètres et liés par une relation de récurrence. On considère deux entiers i et j avec $1 \leq i \leq n$ et $0 \leq j \leq P$, et l'expression booléenne $T(i, j)$: « étant donnés les i premiers éléments de la famille, il existe un sous-ensemble de ces i éléments de poids j ». On remplit alors ligne par ligne un tableau A, qui contient les valeurs de T dont les colonnes sont indicées par j et les lignes par i.

      i. Donner la formule qui lie la ligne i et $i-1$ et $p(a_i)$.

      ii. Illustrer ce principe avec les données suivantes : $n = 6, p(a_1) = 5, p(a_2) = 9, p(a_3) = 3, p(a_4) = 8, p(a_5) = 2, p(a_6) = 5$.

      iii. Comment avec le tableau rempli obtient-on les éléments de la partition ?

   (b) Donner la complexité de cet algorithme.

2. Le problème du sac à dos :

   Nous considérons le problème du sac à dos sans répétition, c'est à dire les objets seront pris au plus une fois. Pour cela considérons un tableau K à deux dimensions tel que $K_{[j,w]}$ représente la valeur maximale que l'on peut stocker dans un sac de capacité w avec des objets $1, \ldots, j$.

   (a) Donner les formules.

   (b) Illustrer le principe avec les données suivantes : $(w_1, v_1) = (1, 1); (w_2, v_2) = (2, 6); (w_3, v_3) = (5, 18); (w_4, v_4) = (6, 22); (w_5, v_5) = (7, 24)$ et $W = 12$.

   (c) Comment retrouver la solution à partir du tableau ?

   (d) Donner la complexité en temps et en mémoire.

1. (a) i. On a :
   — $T(i,0) = V$ (cas de base de la récurrence).
   — $T(i,j) = \begin{cases} V & \text{si } p(a_i) = j \\ V & \text{si } T(i-1,j) = V \\ V & \text{si } p(a_i) \geq P \text{ et } T(i-1, j-p(a_i)) \\ F & \text{sinon} \end{cases}$

   ii. On construit le tableau A suivant :

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	V					V											
2	V					V				V					V		
3	V			V		V			V	V			V		V		
4	V			V		V			V	V		V	V	V	V		V
5	V		V	V		V		V	V	V	V	V	V	V	V	V	V
6	V		V	V		V		V	V	V	V	V	V	V	V	V	V

   iii. Déjà, il y a bien une 2-PARTITION si $A_{[n][P]} = V$. Pour la reconstruire, il faut suivre les étapes suivantes :
      A. Tant que $A_{[i-1][j]} = V, i = i-1$.
      B. $j = j - p(a_i)$ et $i = i-1$.
      C. Aller à (A).

      On arrive au cas de base si $i = 1$ ou $j = 0$. On peut donc exhiber l'algorithme suivant :

```
set reconstruire(int **A, int i, int j) {
    if (i == 1) return {a_i};
    if (j == 0) return {};
    if (A[i-1][j]) return reconstruire(A, i - 1, j);
    return {a_i} + reconstruire(A, i - 1, j - p(a_i));
}
```

      Cet algorithme nous permet de récupérer une des deux partitions. Il suffit de mettre tous les autres éléments dans l'autre partition.

   (b) La construction du tableau A prend un temps en $O(nP)$. Pour le remplissage, c'est fait en $O(n)$. La complexité de l'algorithme est donc $O(nP)$. Cependant, ce n'est pas un algorithme polynomial, mais pseudo-polynomial en la taille de l'entrée (qui sont des entiers, donc codés sur $O(log(n))$ bits).

2. (a) On a :
   — $K_{[j,0]} = 0$ et $K_{[0,w]} = 0$ (cas de base).
   — $K_{[j,w]} = max\{K_{[j-1][w]}, K_{[j-1][w-w_j]} + v_j \text{ (si } w_j > w)\}$

   (b) On construit le tableau K suivant :

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1	1	1
2	0	1	6	7	7	7	7	7	7	7	7	7	7
3	0	1	6	7	7	18	19	24	25	24	25	25	25
4	0	1	6	7	7	18	22	24	28	29	29	40	41
5	0	1	6	7	7	18	22	24	28	30	31	40	42

   (c) On procède un peu de la même manière que précédemment :

i. tant que $K_{[j][w]} = K_{[j-1][w]}$, $j = j - 1$.
ii. $w = w - w_j$ et $j = j - 1$.
iii. Aller à (i).

On arrive au cas de base si $w = 0$ (ou $j = 0$) :

```
set reconstruire(int **K, int j, int w) {
    if (w == 0 || j == 0) return {};
    if (K[j-1][w]) return reconstruire(K, j - 1, w);
    return reconstruire(K, j - 1, w - w_j) + {j};
}
```

(d) La construction du tableau se fait en $O(nW)$, et la reconstruction en $O(n)$, l'algorithme est donc en $O(nW)$. C'est un algorithme pseudo-polynomial en la taille de l'entrée (qui sont des entiers, donc codés sur $O(log(n))$ bits).

Exercice 71 - Mètre du charpentier

Montrer que le problème du mètre de charpentier est un problème $\mathcal{NP}$-complet.

Mètre du charpentier

Entrée : La longueur de l'étui k et des segments $L = \{l_1, l_2, \ldots, l_n\}$.

Question : Peut-on plier le mètre pour qu'il rentre dans l'étui ?

— Ce problème est dans $\mathcal{NP}$. En effet, il suffit de prendre les indices des plis en certificat pour vérifier que la taille entre deux plis est bien inférieure ou égale à L.

— Ce problème est $\mathcal{NP}$-complet. En effet, on peut réduire Partition à ce problème. Soit $\langle X, P \rangle$ une instance de Partition. Soit $S = \sum_{x \in X} x$. On transforme cette instance de Partition en instance de Mètre du charpentier en faisant :

$$L = X$$
$$k = \frac{S}{2}$$

Montrons que cette réduction est correcte, c'est à dire qu'il existe une Partition de X si et seulement s'il existe un pliage du Mètre du charpentier pour qu'il rentre dans l'étui :

⇒ Supposons qu'il existe une solution à Partition, c'est à dire qu'il existe deux ensembles S_1 et S_2 tels que $S_1 \cap S_2 = \emptyset$, $S_1 \cup S_2 = X$ et $\sum_{s_1 \in S_1} s_1 = \sum_{s_2 \in S_2} s_2 = \frac{S}{2}$. Une solution à Mètre de Charpentier serait de plier à chaque segment qui appartient à s_1 et à s_2. Il est évident que c'est bel et bien une solution. En effet, dans le pire des cas, on peut plier au milieu, car il y a deux partitions.

⇐ Supposons qu'il existe une solution à Mètre de charpentier. On reconstruit une solution à Partition en prenant mettant tout dans un ensemble, puis en alternant les ensembles à chaque fois qu'il y a un pli. Comme la longueur des segments ne peut pas dépasser $\frac{S}{2}$ et que la taille totale des éléments est S, il est évident que c'est une solution à Partition.

— Ce problème est dans $\mathcal{NP}$, et il est $\mathcal{NP}$-dur car Partition se réduit à celui-ci. Ainsi, le problème Mètre du charpentier est $\mathcal{NP}$-complet.

Exercice 72 - Algorithmes pseudo-polynomiaux

Donner deux algorithmes pseudo-polynomiaux pour résoudre le problème du sac-à-dos dont le temps d'exécution est proportionnel au produit d'un polynôme en n (nombre d'objet) et au volume du sac à dos (pour l'un des algorithme) et au poids de l'objet le plus lourd (pour l'autre).

— L'algorithme pseudo-polynomial dont le temps d'exécution est proportionnel au produit d'un polynôme n et du volume du sac à dos est celui décrit dans l'exercice 70.

— Je vois pas du tout comment faire un algo dynamique en fonction du poids de l'objet le plus lourd, si quelqu'un a une idée, je la veux bien.

Ces exercices donnent un bon aperçu de la panoplie de problèmes $\mathcal{NP}$-complets connus. Cependant, il y en a plus de 3000, et ce chiffre augmente tous les jours. Pour ceux qui voudraient aller plus loin dans ce domaine, le livre « Computers and Intractability : A Guide to the Theory of NP-Completeness » de Michael R. Garey et David S. Johnson est une excellente introduction au domaine, bien plus poussée que ce qui a été vu dans ce cours.