TD HAI602I Calculabilité/Complexité

Année 2023-24

Version 1.4

Université de Montpellier
Place Eugène Bataillon
34095 Montpellier Cedex 5

Rodolphe Giroudeau
161, rue Ada
34392 MONTPELLIER Cedex 5
TEL : 04-67-41-85-40
Mail : rodolphe.giroudeau@lirmm.fr

Université de Montpellier
Licence HAI602I

24 janvier 2024
Durée: 30h

Calculabilité/Complexité
TD – Séance n^o 1

---

1 Calculabilité

1.1 Divers

Exercice 1 – Ensemble infini

Hotel d'Hilbert

Entrée : Un hotel possède une infinité de chambres.
Question : Peut-on trouver une chambre libre pour un nouveau client ?

Hotel d'Hilbert suite

Entrée : Un hotel possède une infinité de chambres.
Question : Peut-on trouver une infinité de chambres libres pour une infinité de clients ?

1. Proposer une solution pour Hotel d'Hilbert et Hotel d'Hilbert suite.
2. Modéliser les deux problèmes et proposer pour chaque problème une fonction mathématique de $\mathbb{N}$ dans $\mathbb{N}$.

Correction exercice 1

1. Hotel d'Hilbert : Soit $\phi : \mathbb{N} \to \mathbb{N}$ tel que $\phi(n) = n + 1$.
2. Hotel d'Hilbert suite : Soit $\phi : \mathbb{N} \to \mathbb{N}$ tel que $\phi(n) = 2n + 1$.

Fin correction exercice 1

Exercice 2 – 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-mêmes 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 ! »

Correction exercice 2

1. Classique, le barbier est une femme ou il n'y a pas de barbier.
2. Si le crocodile dévorait l'enfant, la mère aurait deviné juste et le crocodile devrait rendre l'enfant.

   Si le crocodile rendait l'enfant, la mère se serait trompée et le crocodile devrait le dévorer.

   Dans les deux cas, le crocodile ne peut pas tenir parole et se trouve face à un paradoxe.

Fin correction exercice 2

1.2 Variations sur le codage

Exercice 3 – Codage de couples d'entiers

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

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

Correction exercice 3

1. $Rang(4, 5) = \frac{(4+5)(4+5+1)}{2} + 5 = \frac{9\times10}{2} + 5 = 50$. 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 $(1, 2)$.
2. $RangRec : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ tel que $RangRec(0, 0) = 0$, $RangRec(0, y) = RangRec(y - 1, 0) + 1$ et $RangRec(x, y) = RangRec(x - 1, y + 1) + 1$
3. $EnumRat : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ et $EnumRat(n) = (x, y)$ tel que $x + y = \max\{m | \frac{m(m+1)}{2} \leq n\}$ et $x = n - \frac{(x+y)(x+y+1)}{2}$.

Fin correction exercice 3

Exercice 4 – Codage de triplets

Soit c la fonction de codage pour les couples d'entiers vue en cours.

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 $c(z, t) = c(x, y) + 1$. Ecrire la fonction successeur qui prend en paramètre un couple et retourne le couple successeur.

Correction exercice 4

1. Soit $c(x, y) = 67$. Soit t entier le plus grand tel que $t(t + 1) \leq 2 \times 67$ donc $t = 11$. On trouve $x = 67 - \frac{11\times12}{2} = 1$, et donc $y = t - x = 1$ et donc $(x, y) = (1, 10)$.
   On trouve $h(x, y, z) = 67 = c(c(x, y), z) = c(1, 10)$ et donc $c(x, y) = 10$ et $z = 1$.
   Soit $c(x, y) = 10, t(t + 1) \leq 20$ alors $t = 4$. Ainsi $y = 0, x = 4$ ainsi $(x, y, z) = (4, 0, 1)$.
2. `Successeur`(x, y).
   `Si` $x \neq 0$ `alors retourne` $(x - 1, y + 1)$
   `Sinon retourne` $(y + 1, 0)$.

Fin correction exercice 4

Exercice 5 – Codage rationnels

Proposer un codage pour les nombres rationnels. Un nombre rationnel est caractérisé par une paire de naturels a/b, telle que $b \neq 0$ et telle que a et b n'ont pas de facteurs communs.

Correction exercice 5

Les nombres rationnels sont dénombrables. En effet, un nombre rationnel est caractérisé par une paire de naturels a/b, telle que $b \neq 0$ et telle que a et b n'ont pas de facteurs communs. Classons les paires de naturels satisfait ces restrictions suivant l'ordre de la somme $a + b$ et, pour chaque valeur de la somme, par ordre de numérateur croissant. Cela nous donne une bijection avec les naturels

$$\{(0/1, 0), (1/1, 1), (1/2, 2), (2/1, 3), (1/3, 4), (3/1, 5), \ldots\}$$

Fin correction exercice 5

Exercice 6 – Codage des listes 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 prendre l'ordre lexicographique ?

Correction exercice 6

1. Il y a une infinité de listes dont la somme des entiers est égale à une valeur donnée par exemple :

$$(1)\ (1\ 0)\ (1\ 0\ 0)\ (1\ 0\ 0\ 0)\ldots$$

   Donc la liste (1 1) par exemple ne sera jamais codée.
2. De manière similaire la liste (1 1) ne sera pas codée car il y a une infinité de listes de longueur 1

Fin correction exercice 6

Exercice 7 – 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.

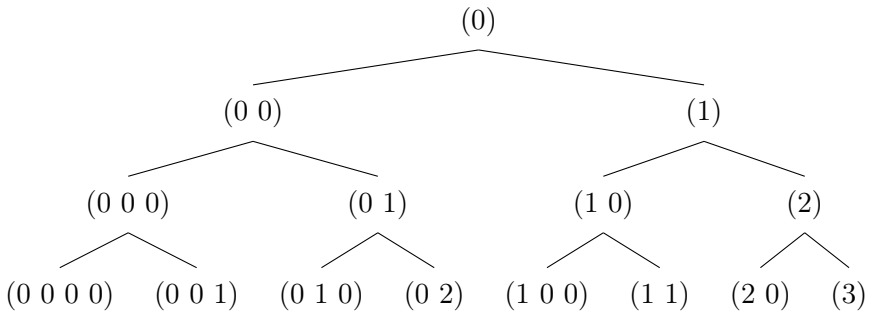
On note U_k l'ensemble des liste l telles que $\sigma(l) = k$ et $u_k = |U_k|$.

1. Donner les ensembles $U_i, i = 0, \ldots, 4$.
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).

Correction exercice 7

1. Voir le détail des ensembles :
   - $U_0 = ()$ liste vide (aucun élément, de longueur nulle)
   - $U_1 = \{(0)\}$
   - $U_2 = \{(0,0),(1)\}$
   - $U_3 = \{(0,0,0),(0,1),(1,0),(2)\}$
   - $U_4 = \{(0,0,0,0),(0,0,1),(0,1,0),(0,2),(1,0,0),(1,1),(2,0),(3)\}$

   Ainsi $U = \{0:();1:(0);2:(00);3:(1);4:(000);5:(01);6:(10);7:(2);8:(0000);9:(001);10:(010);11:(02);12:(100);13:(11);14:(20);15:(3)\}$.
2. Fini car borné par la longueur de k zéros. 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. Premier élément de l'ensemble k zéros, et le dernier $(k-1)$.
4. On a clairement que $u_{k+1} = 2u_k$ et $u_1 = 1$ alors $u_k = 2^{k-1}$..

$$\text{Fonction de codage : } \begin{cases} c(\alpha\ 0) = 2c(\alpha) \\ c(\alpha\ x) = 2c(\alpha\ (x-1)) + 1 \end{cases}$$

Algorithm 1 La fonction de CODAGE (version récursif), CODE($\alpha\ x$).

```
if x = 0 then
  if α est vide then
    RETOURNER 1 ;
  else
    RETOURNER 2∗CODE (α) ;
  end if
else
  RETOURNER 2∗CODE(α (x − 1)) + 1 ;
end if
```

Par exemple, $c(000) = 2c(00) = 2 * 2c((0)) = 2 * 2 * 1$, $c(01) = 2c(00) + 1 = 2 * 2c((0)) + 1 = 2 * 2 * 1 + 1 = 5$.

Pour la version itérative :

Pour aller de la liste (0) à la liste $(x_1\ x_2\ \ldots x_n)$. On va de (0) puis $(x_1\ 0)$ puis $(x_1\ x_2)$ etc,

Fonction de DÉCODAGE donnée par l'algorithme 3.

La fonction fonctionne de manière suivante :

— Si k est pair alors j'écris un zéro en tête de liste,
— Sinon j'incrémente de un la tête de la liste.

Algorithm 2 La fonction de CODAGE($x_1\ x_2\ \ldots x_n$) (version itérative).

```
c = 1 ; m = 1 ;
while m < n do
  for i = 1 à x_m do
    c := 2c + 1 {pour passer de (x_1 x_2 ... x_(m−1) 0) à (x_1 x_2 ... x_(m−1) x_m)}
  end for
  c := 2 ∗ c ; {pour passer de (x_1 x_2 ... x_(i−1) x_i) à (x_1 x_2 ... x_(i−1) x_i 0)}
end while
for i = 1 à x_n do
  c := 2 ∗ c + 1 {pour passer de (x_1 x_2 ... x_(n−1) 0) à (x_1 x_2 ... x_(n−1) x_n)}
end for
```

Algorithm 3 La fonction de DÉCODAGE(k).

```
if k = 0 then
  Retourner Sol:= () ;
end if
Sol:= (0) ;
while k ≥ 2 do
  if k%2 = 0 then
    Sol :=Cons(0,Sol) ;
    k := k/2 ;
  else
    k := (k − 1)/2 ;
    Sol :=Cons(tete[Sol]+1,queue(Sol)) ;
  end if
end while
Retourner Sol ;
```

Fin correction exercice 7

Exercice 8 – Codage d'entiers

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

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

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

1. Donner quelques éléments de $A_i, \forall i \in \{0,1,2,3,4,5,6\}$.
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 $A_4 \cup A_6$.

Correction exercice 8

1. $A_0 = \{1\}$, $A_1 = \{2\}$, $A_2 = \{4\}$ $A_3 = \{8\}$, $A_4 = \{16, 5, 10, 20, 3, 40, 6, 80, 13, 12, 160, \ldots\}$, $A_5 = \{32\}$, A_6 est infini.
2. Dans un premier temps on peut considérer l'algorithme suivant :

   $n = 0$
   `Tant que (true){`
   `  Si` $f(n) = i$ `afficher` n `;` $n++$`;`
   `}`

   ne fonctionne pas a priori (f est-elle définie sur toutes les valeurs ?).

   — Une autre idée $2^i \in A_i$.
   — Si $n \in A_i$ alors $2^n \in A_i$, si n n'est pas une puissance de 2 et $(n-1)/3$ si $(n-1)/3 \in \mathbb{N}$

   Ceci donne l'algorithme suivant :
   `Procédure : Afficher` A_i
   $L = filevide()$`;`
   `Si` $(2^i - 1) \mod 3 = 0$ `alors` $L = ajouterfin((2^i - 1)/3, L)$
   `Sinon` $L = \varnothing$
   `Tant que` L `est non vide`
   `{`
   $n := tete(L)$
   $L := queue(L)$`;`
   $L := ajouterfin(2n, L)$`;`
   $X = \frac{n-1}{3}$`;` `si` X `entier impair alors` $L := ajouterfin((n-1)/3, L)$`;`
   `}`

`Retourner` $L = ajoutertete(2^i, L)$;

3. Affiche A_4 puis Affiche A_6, ne marche pas car A_4 est infini. Il faut modifier l'algorithme $Affiche(A)$ pour avoir le n^{ieme} élément de A_i.

   $n = 1$
   `Tant que vrai {`
   `Afficher` n^{ieme} `de` A_4
   `Afficher` n^{ieme} `de` A_6
   $n++$ }

Fin correction exercice 8

Exercice 9 – Gödelisation

Le but est de donner un numéro entier naturel à chaque objet manipulé (mot, instruction, machine de Turing...) et de remplacer des manipulation de ces objets par des opérations de nature arithmétique sur leurs numéros. Nous commençons par les séquences d'entiers. Le numéro de Gödel d'un n-uplet d'entiers naturels $(x_1, \ldots, x_n)$, noté par $\langle x_1, \ldots, x_n \rangle$, est défini comme suit :

$$A = \langle x_1, \ldots, x_n \rangle = 2^n.3^{x_1}.5^{x_2} \ldots p_n^{x_n} = 2^n \prod_{i=1}^{n} p_i^{x_i}$$

1. Donner $\langle 10, 11 \rangle$.
2. Le numéro de Gödel d'un n-uplet est une opération injective ? Surjective ?
3. Etendre le nombre de Gödel au mot défini par $w = a_{i_1} a_{i_2} \ldots a_{i_l}$.
4. Etendre le nombre de Gödel aux instructions $I : q_i a_j \rightarrow q_k a_l \left\{ \begin{array}{c} \text{-} \\ G \\ D \end{array} \right\}$ pour les machines de Turing. **Rappel :** Les instructions ont la forme suivante :(état, symbole lu) → (état, symbole écrit, déplacement).
5. Etendre le nombre de Gödel au numéro d'une machine de Turing avec $I_1, \ldots, I_n$ les instructions de la machine M.
6. Etendre le nombre de Gödel au numéro d'une configuration $c = (w_G, q_i, w_D)$.

Correction exercice 9

1. $\langle 10, 11 \rangle = 2^2.3^{10}.5^{11}$.
2. injective par construction (Le numéro de Gödel d'un n-uplet est injective puisque la décomposition en facteurs premiers est unique) et non surjective, un numéro n'est pas forcément associé à un nombre de Gödel.
3. Le numéro de w est $\langle w \rangle = \langle i_1, i_2, \ldots, i_l \rangle$. Par exemple $\langle a_3 a_7 a_1 \rangle = \langle 3, 7, 1 \rangle = 2^3.3^3.5^7.7^1$.

4. $\langle I\rangle = \langle i, j, k, l, \left\{ \begin{array}{c} \text{-} \\ G \\ D \end{array} \right\}\rangle$
5. $\langle M\rangle = \langle\langle I_1\rangle, \ldots, \langle I_n\rangle\rangle$
6. $\langle c\rangle = \langle\langle w_G\rangle, \langle i\rangle, \langle I_n\rangle\rangle$

Fin correction exercice 9

Exercice 10 – Etude 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. Pour répondre, il suffit de prouver que l'égalité $f(n) = an$ est vrai pour $n = 0$ et $n = 1$, est vraie $\forall n \in \mathbb{N}$.
3. Vérifier successivement les égalités $f(2) = 2a, f(4) = 4a$ et $f(5) = 5a$.
4. Utiliser les valeurs de $f(4)$ et de $f(5)$ 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

$$\forall m, \text{ on a } \left\{ \begin{array}{l} (2m)^2 + (m-5)^2 = (2m-4)^2 + (m+3)^2 \\ (2m+1)^2 + (m-2)^2 = (2m-1)^2 + (m+2)^2 \end{array} \right.$$

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

Correction exercice 10

1. $f(m^2+n^2) = f(m)^2 + f(n)^2$ avec $m = n = 0$, donc $f(0^2+0^2) = f(0)^2 + f(0)^2$ donc $0 = f(0)(-1+2f(0))$. LA seule solution est $f(0) = 0$. De plus $f(0^2+n^2) = f(0)^2 + f(n)^2$.
2. $f(1^2) = f(0^2+1^2) = f(0)^2 + f(1)^2 = f(1)^2$.
3. $f(2) = f(1^2+1^2) = f(1)^2 + f(1)^2 = 2a^2 = 2a$. $f(4) = f(2^2+0^2) = f(2)^2 + f(0)^2 = 4a^2 = 4a$. $f(5) = f(2^2+1^2) = f(2)^2 + f(1)^2 = 4a^2 + a^2 = 5a$.
4. $25a^2 = f(5)^2 + f(0)^2 = f(5^2+0^2) = f(5^2) = f(4^2+3^2) = f(4)^2 + f(3)^2 = 16a^2 + f(3)^2$

5. $2f(5)^2 = f(5^2+5^2) = f(50) = f(7^2+1^2) = f(7)^2 + f(1)^2$
6. $f(8) = f(4+4) = f(2^2+2^2) = f(2)^2 + f(2)^2$, $f(9) = f(3^2+3^2) = f(3)^2 + f(3)^2$, $f(10) = f(3^2+1^2)$, $f(6)^2 + f(8)^2 = f(6^2+8^2) = f(100) = f(10^2+0^2) = f(10)^2 + f(0)^2$
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 $\forall n, f(n) = an$.

Fin correction exercice 10

1.3 Diagonalisation

Exercice 11 – Diagonalisation

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

Correction exercice 11

Supposons que $[0,1[= \{x_n : n \geq 1\}$. On écrit le développement (propre) en base 10 de x_n

$$x_n = \sum_{k\geq 1} a_{k,n} 10^{-k},$$

$\forall n \geq 1$, définissons

$$\begin{aligned} a_n = 2 \quad &\text{si} \quad a_{n,n} = 1 \\ a_n = 1 \quad &\text{si} \quad a_{n,n} \neq 1 \end{aligned}$$

On note

$$x = \sum_{k\geq 1} a_k 10^{-k} \in [0,1[,$$

Alors $\forall n \geq 1,\ x \neq x_n$ car $a_n \neq a_{n,n}$, ce qui est une contradiction.

Autre version : 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	$\dots$	d_i	$\dots$
x_0	$\boxed{1}$	1	0	$\dots$	1	$\dots$
x_1	1	$\boxed{1}$	0	$\dots$	1	$\dots$
x_2	0	0	$\boxed{0}$	$\dots$	0	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
x_i	1	0	1	$\dots$	$\boxed{1}$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

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.

Fin correction exercice 11

Exercice 12 – Diagonalisation

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

Correction exercice 12

1. On raisonne par l'absurde. Supposons que U soit dénombrable, et soit $(x_n)_{n\in\mathbb{N}}$ une énumération de U.

x	0	1	2	...	j	...
x_0	$x_0(0)$	$x_0(1)$	$x_0(2)$	...	$x_0(j)$	$\vdots$
x_1	$x_1(0)$	$x_1(1)$	$x_1(2)$	...	$x_1(j)$	$\vdots$
x_2		*	*		*	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_i	$x_i(0)$	$x_i(1)$	$x_i(2)$	...	$x_i(j)$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

On procède ensuite par un argument dit de diagonalisation : soit u la suite définie par $\forall n \in \mathbb{N}, u_n = 0$ si $x_n(n) = 1$, sinon $u_n = 1$ si $x_n(n) = 0$. Alors u ne peut être à aucun des x_n, puisque $u_n \neq x_n(n), \forall n$

Ceci nous prouve aussi que $\mathcal{P}(\mathbb{N})$ n'est pas dénombrable puisque $\mathcal{P}(\mathbb{N})$ est en bijection avec U : il faut pour cela associer à chaque associer à chaque partie de A de $\mathbb{N}$ la suite u défini par la fonction caractéristique χ_A de A, c'est à dire $u_n = \chi_A(n), \forall n \in \mathbb{N}$.

Fin correction exercice 12

1.4 Dénombrabilité

Exercice 13 – 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.

Correction exercice 13

L'ensemble des entiers est infini parce qu'on peut le mettre en relation bijective avec le sous-ensemble des entiers pairs ; en effet, la relation $\{(n, 2 \times n) \backslash n \in \mathbb{N}\}$ est bijective.

Fin correction exercice 13

Exercice 14 – Dénombrabilité

Montrez que l'ensemble F des parties finies de $\mathbb{N}$ est dénombrable.

Correction exercice 14

A toute partie finie non vide A de $\mathbb{N}$, on lui associe la suite binaire $x_0, x_1, x_2, \ldots, x_n$ définie

par $x_k = 1$ si l'entier k appartient à A et 0 sinon, avec la contrainte que $x_n = 1$ (autrement dit n est le plus grand élément de A). Le nombre entier strictement positif codé en binaire par $x_n, x_{n-1}, x_{n-2}, \ldots, x_2, x_1, x_0$ caractérise entièrement l'ensemble A. Il suffit ensuite de coder l'ensemble vide par 0 et l'on vient de décrire une bijection de F dans $\mathbb{N}$, ce qui prouve que F est dénombrable. Par exemple la partie $\{2, 3, 5, 8\}$ est associée tout d'abord au mot binaire 100101100 qui représente l'entier $256 + 32 + 8 + 4 = 300$. Réciproquement l'entier $25 = 1 + 8 + 16$, soit 1101 en binaire, code l'ensemble $\{0, 2, 3\}$.

Fin correction exercice 14

1.5 Fonctions (non)-calculables

Exercice 15 – 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 $g_1 : \mathbb{N} \to \mathbb{N}$ totale, croissante et non calculable à partir de f.
3. Construire une fonction $g_2 : \mathbb{N} \to \mathbb{N}$ totale, strictement croissante et non calculable à partir de f.
4. Est-il vrai que toute fonction $f : \mathbb{N} \to \mathbb{N}$ qui est non bornée est également non calculable ?

Correction exercice 15

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. Une solution est donne par $g(n) = \sum_{i=0}^{n} f(i)$. g n'est pas calculable, car on a , $\forall n > 0$

$$f(n) = g(n) - g(n-1)$$

ce qui permet de calculer f à partir de g.
3. $g(n) = f(n) + 2n$, pour obtenir une fonction strictement croissante.
4. Faux en prenant $f(n) = n$ qui non bornée mais calculable.

Fin correction exercice 15

Exercice 16 – Calculabilité

1. Une fonction $\mathbb{N} \to \mathbb{N}$ totale, non calculable et croissante, peut-elle être bornée ?

2. Montrer que toute fonction totale, croissante et bornée est calculable. Que se passe-t-il si une fonction est totale décroissante ?

Correction exercice 16

1. Non, sinon elle est ultimement constante (on travaille sur les entiers), donc calculable ; ce qui signifie qu'une procédure pour la calculer existe, pas forcément qu'on est capable de la fabriquer
2. De même une fonction décroissante est ultimement constante, il suffit cette de ne pas oublier qu'on travaille sur des entiers naturels.

Fin correction exercice 16

Exercice 17 – Calculabilité

1. Montrer que l'inverse d'une fonction f calculable et bijective est calculable.
2. Si f n'est surjective, quel est le problème rencontré avec la procédure précédente ? Même chose pour une fonction surjective mais partielle ?
3. Si f n'est pas injectif, que calcule f^{-1} ?

Correction exercice 17

1. On peut remarque que si f n'est pas surjective (contrairement à l'hypothèse de l'énoncé), cette procédure boucle si y n'appartient pas à l'image de f, et calcule donc une fonction partielle.
```
int f (int x) {...}
int g (int y) {
     int x;
     for(x = 0;;x++)
          if (f(x) == y) return x;
}
```
2. Enfin si f est surjective mais partielle, cette procédure échoue en général à trouver un x tel que $f(x) = y$, car elle boucle dès qu'elle rencontre une valeur pour laquelle f n'est pas définie.
3. Si f n'est pas injective (contrairement à l'hypothèse de l'énoncé), cette procédure reste intéressante, elle calcule :
$$\min\{x | f(x) = y\}$$

Fin correction exercice 17

Exercice 18 – 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.

Correction exercice 18

1. si f est totale et calculable, voici une procédure qui calcule la fonction caractéristique χ du graphe G : `int f (int` x`) {`$\ldots$`}`
   `int` χ `(int` x`, int` y`) {`
   `return` $y == f(x)$`;`
   } il est essentiel que f soit calculable pour que la procédure ci-dessus calcule bien la fonction caractéristique de G.
2. Réciproquement si χ est calculable, voici une procédure qui calcule f :

```
 int χ (int x, int y) {...}
int f (int x) {
     int y;
     for (y = 0;; y++)
          if (χ(x, y)) return y;
}
```

Fin correction exercice 18

Exercice 19 – Calculabilité

Soient E un ensemble et ϕ une fonction telle que $\phi(n)$ est égale au nombre d'éléments de E strictement inférieur à n.

1. Montrer que ϕ est calculable si et seulement si E est décidable.

Correction exercice 19

1. Si E est décidable, sa fonction caractéristique χ est, par définition, calculable, et voici la procédure (évidente) qui calcule ϕ

```
 int χ (int x){...}
int φ (int n) {
     int x, k = 0;
     for (x = 0; x < n; x++)
          If (χ(x)) k++
return k;
}
```

   Inversement, si ϕ est calculable, voici la procédure qui calcule χ :

```
 int φ (int x) {...}
int χ (int x) {
     return φ(x + 1) − φ(x);
}
```

Fin correction exercice 19

1.6 Problèmes indécidables

Exercice 20 – Une preuve incorrecte

Nous considérons la fonction suivante donné par l'algorithme 4 :

Algorithm 4 La fonction de Collatz

while $n \neq 1$ **do**
  if $n = 0 \mod 2$ **then**
    $n := n/2$
  else
    $n := 3 \times n + 1$
  end if
end while

Actuellement nous ne savons pas si cette fonction termine $\forall n$.

Est-ce que vous êtes 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 »

Correction exercice 20

Cette preuve est incorrecte car elle confond l'ignorance d'une possibilité avec la connaissance d'une impossibilité.

Ne pas savoir si ce programme termine n'est pas de même nature que savoir que le problème de l'arrêt est indécidable. Dans le cas de la fonction de Collatz, nous sommes dans le domaine de l'ignorance : aujourd'hui nous ne savons pas, mais demain ? Dans le cas du problème de l'arrêt nous sommes dans le domaine de la connaissance : nous savons depuis 70 ans que ce problème est indécidable sous une certaine définition de ce que sont les fonctions, les programmes et les calculs ; cette connaissance est définitive.

Fin correction exercice 20

Exercice 21 – Problème de Post

Né en Pologne en 1897, Emil Post émigre pour les Etats-Unis en 1904, où il mène de brillantes

études à l'université de Columbia. Malgré une maladie très handicapante, il conduit des recherches en logique au City College de New York qui l'amènent à définir en 1946 le problème qui porte aujourd'hui son nom, dont il démontre l'indécidabilité. Il meurt en 1954.

Le problème de Problème de Post est un problème de décision.

Problème de Post (Post)

Entrée : Soit un alphabet. Une suite finie $(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)$ de couples de mots.

Question : Peut-on aligner des dominos de sorte que les mots résultant sur chaque moitié sont les mêmes : existe-t-il une suite $k_1, k_2, \ldots, k_p$ d'indices à prendre dans $[1, n]$ (on peut choisir plusieurs fois le même indice) telle que les mots $u_{k_1} u_{k_2} \ldots u_{k_p}$ et $v_{k_1} v_{k_2} \ldots v_{k_p}$ soient identiques ?

Le problème est indécidable en général.

1. $(1, 111); (10111, 10); (10, 0)$
2. $(10, 101); (011, 11); (101, 011)$
3. $(ab, aa); (a, baa); (bba, bb)$

Correction exercice 21

1. $(1, 111); (10111, 10); (10, 0)$

   On commence par $(10111, 10)$, puis $(1, 111)$, puis $(1, 111)$ et pour finir $(10, 0)$. (Indices $2, 1, 1, 3$).
2. $(10, 101); (011, 11); (101, 011)$.

   On commence par le domino $(10, 101)$. On continue nécessairement par $(101, 011)$ et ensuite par $(101, 011)$ et encore indéfiniment.
3. (bba, bb) puis (ab, aa), (bba, bb) et pour finir (a, baa)

Fin correction exercice 21

Exercice 22 – Equations diophantiennes

On appelle équation diophantienne est une équation du genre $P(x, y, z, \ldots,) = 0$, où P est un polynôme à cœfficients entiers. Résoudre une telle équation, c'est chercher les solutions sous forme d'entiers.

1. Quelle est la solution de $x^2 + y^2 - 1 = 0$?
2. Trouver la solution de $x^2 - 991y^2 - 1 = 0$ autre que $(1, 0)$?
3. Retrouver une équation connue depuis l'antiquité.

Correction exercice 22

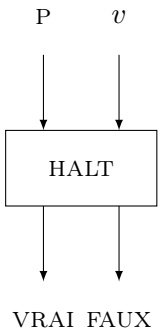


FIGURE 1 – Illustration graphique du programme HALT. P est un programme et v des données.

1. $x = 1$ et $y = 0$ ou $x = 0$ et $y = 1$.
2. $x = 379\ 516\ 400\ 906\ 811\ 930\ 638\ 014\ 896\ 080$ et $y = 12\ 055\ 735\ 790\ 331\ 359\ 447\ 442\ 538\ 767$
3. $x^n + y^n = z^n$

Fin correction exercice 22

Exercice 23 – Variantes du problème l'arrêt

1. Comment dériver les programmes SELF-HALT et ANTI-SELF-HALT définis ci-après à partir de HALT.
   (a) 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 appliqué à des données v.
   (b) ANTI-SELF-HALT : le programme ANTI-SELF-HALT(p) s'arrête si et seulement si p ne s'arrête pas.
   (c) Quel impact sur le status des problèmes SELF-HALT et ANTI-SELF-HALT ?
2. Monter que les problèmes suivants sont indécidables.
   (a) $\text{HALT}_\exists$: le problème de l'arrêt existentiel, existe-t-il une entrée pour laquelle le programme s'arrête ?
   (b) $\text{HALT}_\forall$: le problème de l'arrêt universel, le programme s'arrête-t-il pour toutes les entrées ? La réduction est l 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.

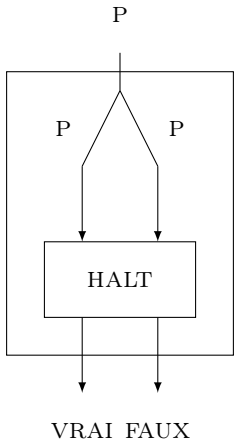


FIGURE 2 – SELF-HALT.

Correction exercice 23

1. Sur la dérivation des problèmes
   (a) SELF-HALT : voir la figure 2
   (b) ANTI-SELF-HALT : la solution est donnée par la figure 3.
   (c) Sachant que HALT est indécidable alors SELF-HALT et ANTI-SELF-HALT
2. Monter que les problèmes suivants sont indécidables.
   (a) $\text{HALT}_{\exists}$:

   Prenons un programme P, une donnée v et formons le nouveau programme P' défini par P'(x) =P$(v); x$, où « ; » dénote la séquence (P' exécute P'(v), puis rend x). Ainsi P'$=$ *vrai* si et seulement si HALT(P,v) est vrai.
   (b) $\text{HALT}_{\forall}$:

   Voir figure 4.
   (c) NEGVAL :

   La solution est donnée par la figure 5.
   (d) EQUIV : Certaines optimisations changent le comportement du programme. Par exemple, on sait que la multiplication par 0 rend toujours zéro ; il est donc tenant de remplacer tous les $0 * x$ par 0. Cependant, si x est une expression complexe dont l'évaluation peut boucler, cette optimisation changera le comportement. Cette réponse n'a pas de réponse automatique.

   La solution est donnée par la figure 6.
   (e) RETURN_0 : Cette question rejoint des questions de conformité des valeurs rendues à des spécifications. Par exemple, concernant un afficheur de vitesse, la vitesse affichée est-elle toujours dasn un intervalle $[V_{\min}, V_{\max}]$?

   La solution est donnée 7.

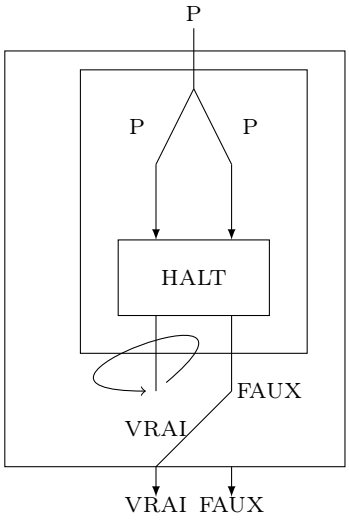


FIGURE 3 – ANTI-SELF-HALT.

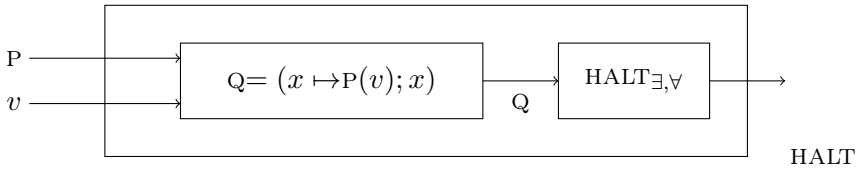


FIGURE 4 – Réduction HALT $\propto$ $\mathrm{HALT}_{\exists}$ ou Réduction HALT $\propto$ $\mathrm{HALT}_{\forall}$.

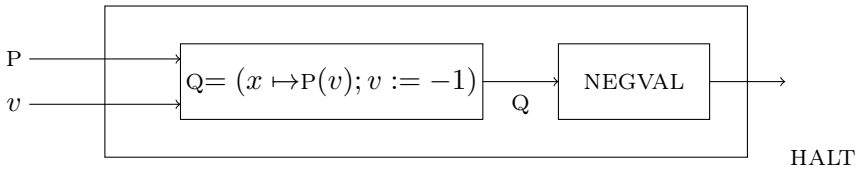


FIGURE 5 – Réduction HALT $\propto$ NEGVAL.

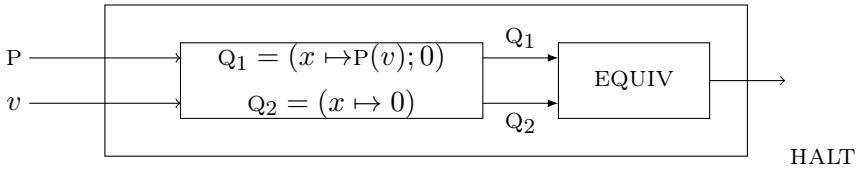


FIGURE 6 – Réduction HALT $\propto$ EQUIV.

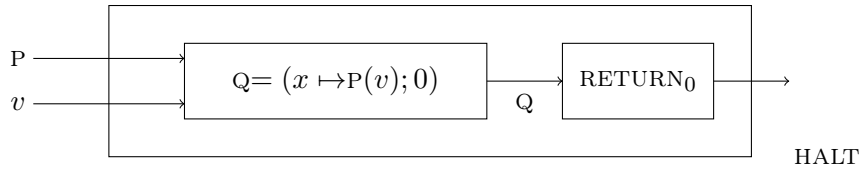


FIGURE 7 – Réduction HALT $\propto$ RETURN$_0$.

Fin correction exercice 23

Exercice 24 – Retour sur le problème du barbier

Formaliser le problème du barbier en utilisant le prédicat $R(x, y)$. Comparer avec la preuve de l'indécidabilité du problème K.

Correction exercice 24

Soit

$$R(x,y) = \begin{cases} 1 \text{ si } x \text{ se rase } R(y,y) = 0 \\ 0 \text{ sinon} \end{cases}$$

Considérons $R(x, x)$

— Si $R(x, x) = 0$ alors $R(x, x) = 1$
— Si $R(x, x) = 1$ alors $R(x, x) = 0$.

Fin correction exercice 24

Exercice 25 – Indécidabilité du problème de Post

Nous allons montrer que le problème de Post est indécidable. Pour cela nous allons procéder à une réduction à partir du problème PROBLÈME DE POST MODIFIÉ défini ci-dessous :

PROBLÈME DE POST MODIFIÉ (Post Modifié)

Entrée : Soit un alphabet. Une suite finie $(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)$ de couples de mots.

Question : Peut-on aligner des dominos de sorte que les mots résultant sur chaque moitié sont les mêmes : existe-t-il une suite $k_1, k_2, \ldots, k_p$ d'indices à prendre dans $[1, n]$ (on peut choisir plusieurs fois le même indice) telle que les mots $u_{k_1} u_{k_2} \ldots u_{k_p}$ et $v_{k_1} v_{k_2} \ldots v_{k_p}$ soient identiques.

Le problème de PROBLÈME DE POST MODIFIÉ impose que le premier indice est 1.

On supposera que PROBLÈME DE POST MODIFIÉ est indécidable (la preuve se fait à partir de HALT$_0$ (le problème de l'arrêt sur la valeur 0)).

1. Montrer qu'il existe une réduction de problème PROBLÈME DE POST MODIFIÉ vers PROBLÈME DE POST.

— On introduit une nouvelle lettre \$, et pour $a_1, \dots a_k \in A$, soient $p(a_1 \dots a_k) = \$a_1 \dots \$a_k$ et $s(a_1 \dots a_k) = a_1\$ \dots a_k\$$.
— Soit $(u_1, v_1), \dots, (u_n, v_n)$ une instance de Problème de Post Modifié.
— Soient les $2n+1$ mots suivants :

$$\begin{aligned} x_i &= p(u_i) && \text{et } y_i = s(v_i) \\ x_{n+i} &= p(u_i)\$ && \text{et } y_{n+i} = s(v_i) \\ x_{2n+1} &= p(u_1) && \text{et } y_{2n+1} = \$s(v_1) \end{aligned}$$

Appliquer cette construction pour l'instance $(bba, bb), (ab, aa), (a, baa)$.

2. Montrer que Problème de Post Modifié est vrai pour l'instance $((u_l, v_l))_{1 \le l \le n}$ a une solution si et seulement si le Problème de Post sur l'instance $((u_l, v_l))_{1 \le l \le 2n+1}$.
3. Proposer une réduction du problème de Problème de Post vers le problème de Problème de Post Modifié.
4. Dans Problème de Post-simple on a les restrictions suivantes :

— u_i et v_i ont la même longueur, $\forall, 2 \le i \le n-1$ (on suppose $n \neq 1$).
— La solution de Problème de Post doit commencer par l'indice 1 et terminer par l'indice n. De plus, ces indices ne peuvent pas être utilisés au milieu.

Donner une solution pour l'instance $(u_1, v_1) = (ab, a), (u_2, v_2) = (aa, ba)$ et $(u_3, v_3) = (a, aa)$,
5. Proposer une réduction de Simple-PCP au problème d'accessibilité.

Correction exercice 25

1. Facile
2. On commence par le couple (x_{2m+1}, y_{2m+1}) puis le même ordre que PCPM
3. Si on a un algorithme pour résoudre PCPM, on a un algorithme pour résoudre PCP. Il suffit de résoudre n PCPM différents, selon le mot avec lequel on commence.
4. La séquence 1, 2, 3 est une solution de Simple-PCP. Pour qu'une solution existe, il faut que $|u_1| - |v_1| = |v_n| - |u_n|$. Soit donc $k = |u_1| - |v_1| = |v_n| - |u_n|$ et supposons que $k > 0$. Pour chaque séquence $1 = i_1, i_2, \dots, i_k$ où $i_2, \dots, i_k \in \{2, \dots, n-1\}$ on a $|u_{i_1} \dots u_{i_k}| - |v_{i_1} \dots v_{i_k}| = k$.
5. On peut chercher une solution pour une instance I de Simple-PCP dans un graphe orienté fini G_I : les sommets sont les mots de longueur k ; on a un arc de u vers v s'il existe un couple (u_j, v_j) tel que $uu_j = v_j v$. Le sommet de départ est le mot w tel que $u_1 = v_1 w$ et le sommet d'arrivée est w' tel que $w'u_n = v_n$. L'instance I de Simple-PCP a une solution si et seulement si il existe un chemin dans G_I de w à w'.

   On a donc réduit Simple-PCP au problème d'accessibilité dans les graphes orientés. Comme ce dernier problème est décidable, Simple-PCP l'est aussi. C'est possible de réduire aussi dans le sens inverse, du problème d'accessibilité à Simple-PCP.

Fin correction exercice 25

Exercice 26 – Indécidabilité pour l'analyse syntaxique

Montrer que le problème de savoir si une grammaire donnée est ambiguë est indécidable. Pour cela procéder à une réduction à partir du problème de Post.

Correction exercice 26

1. A un problème de Post $(u_1, v_i)_{i\in[1,n]}$ sur l'alphabet $A = \{a, \ldots, z\}$, associons les grammaires suivantes :

$$\begin{array}{rcl} S_1 & \to & u_1\#S_1\#miroir(v_1) \\ & \to & u_2\#S_1\#miroir(v_2) \\ \ldots & & \\ & \to & u_N\#S_1\#miroir(v_N) \\ & \to & u_1\#\$\#miroir(v_1) \\ & \to & u_2\#\$\#miroir(v_2) \\ \ldots & & \\ & \to & u_N\#\$\#miroir(v_N) \end{array}$$

$$\begin{array}{rcl} S_2 & \to & \$ \\ & \to & \#S_2 \\ & \to & T \\ T & \to & T\# \\ & \to & U \\ U & \to & aS_2a \\ \ldots & & \\ & \to & zS_2z \end{array}$$

$$\begin{array}{rcl} S & \to & S_1 \\ & \to & S_2 \end{array}$$

La grammaire S_1 génère les mots de la forme $u_{i_1}\# \ldots u_{i_k}\#\$\#miroir(v_{i_k})\# \ldots \#miroir(v_{i_1})$ et n'est pas ambiguë.

La grammaire S_2 génère les mots qui sont des palindromes pairs non vides sur A avec un \$ au centre et des # parasites n'importe où. S_2 n'est pas ambiguë.

Dire que S est ambiguë, c'est donc dire qu'il y a un mot qui est généré à la fois par S_1 et par S_2, c'est-à-dire qui répond aux deux caractéristiques, ce qui est équivalent à dire que le problème de Post a une solution.

Par exemple, la grammaire associée à l'exemple ci-dessus est ambiguë, en effet le mot

$$a\#baabbaba\#a\#\$\#aaba\#bb\#bb\#aaba$$

Peut être obtenu des deux façons différentes.

On obtient donc que le problème de Post a une solutions si la grammaire est ambiguë. On construit alors un programme qui transforme un problème de Post en la grammaire associé, et teste si la grammaire est ambiguë. On obtient un programme qui répond au problème de Post, ce qui est absurde. Donc le problème de l'ambiguïté des grammaires est indécidable.

Fin correction exercice 26

Exercice 27 – Indécidabilité de l'intersection de deux langages algébriques

On veut montrer que le problème de savoir si l'intersection de deux langages algébriques est vide est indécidables. La réduction se fait à partir de Problème de Post.

Construction : Soit $P = \{(u_1, v_1), \ldots, (u_n, v_n)\}$ un problème Problème de Post indécidable sur l'alphabet A. Soient n nouveaux symboles $c_1, \ldots, c_n$ qu'on ajoute à l'alphabet A. Soit G_1 la grammaire algébrique définie par

$$\begin{aligned} S_1 &\to iEu_i, i = 1, \ldots, n \\ E &\to iEu_i, i = 1, \ldots n \\ E &\to \Delta \end{aligned}$$

Et soit G_2 la grammaire algébrique définie par :

$$\begin{aligned} S_2 &\to iFv_i, i = 1, \ldots, n \\ F &\to iFv_i, i = 1, \ldots n \\ F &\to \Delta \end{aligned}$$

1. Appliquer la construction précédente avec $(ab, aa); (a, baa); (bba, bb)$.
2. Conclure de manière générale.

Correction exercice 27

Soit $P = \{(u_1, v_1), \ldots, (u_n, v_n)\}$ un problème PROBLÈME DE POST indécidable sur l'alphabet A. Soient n nouveaux symboles $c_1, \ldots, c_n$ qu'on ajoute à l'alphabet A. Soit G_1 la grammaire algébrique définie par

$$\begin{aligned} S_1 &\rightarrow iEu_i, i = 1, \ldots, n \\ E &\rightarrow iEu_i, i = 1, \ldots n \\ E &\rightarrow \Delta \end{aligned}$$

Par exemple, $123\Delta u_3 u_2 u_1$ avec Δ un séparateur. Cette grammaire engendre des mots de la forme $ij \ldots k\Delta u_k \ldots u_j u_i$. La partie à droite du Δ est une suite de mots, et la partie à gauche est la suite inversées des index de ces mots dans l'énoncé du problème de Post. Remarquons que toutes les séquences de u_i peuvent être produites.

On montre comme que $L(G_1) \cap L(G_2) \neq \varnothing$ ssi P admet une solution, ce qui est indécidable avec le choix P.

Si le mot est engendré par G_1 et par G_2, il sera de la forme $ij \ldots k\Delta u_k \ldots u_j u_i$ vu par G_1 et de la forme $ij \ldots k\Delta v_k \ldots v_j v_i$ vu par G_2. Donc la séquence $y_k \ldots u_j u_i$ sera égale à la séquence $v_k \ldots v_j v_i$, et la suite des indices $k \ldots ji$ sera une réponse au problème PROBLÈME DE POST.

Fin correction exercice 27

1.7 Fonctions caractéristiques

Exercice 28 – Exemples et propriétés

Soit E un ensemble et $\mathbb{A}$ un sous-ensemble de E. La fonction indicatrice (ou caractéristique) de $\mathbb{A}$, notée $1_{\mathbb{A}}$, est la fonction de E dans $\{0, 1\}$ définie par :

$$1_{\mathbb{A}} : \; E \rightarrow \{0, 1\}$$
$$1_{\mathbb{A}}(x) = \begin{cases} 1 \text{ si } x \in \mathbb{A} \\ 0 \text{ sinon} \end{cases}$$

Donner les solutions de $1_{\mathbb{A}}(x)$ pour les ensembles suivants :

1. $1_{\mathbb{Q}}(\sqrt{2})$, $1_{\mathbb{Q}}(\frac{2}{3})$
2. $(A \subseteq B) \Longleftrightarrow ?$.
3. $1_{\mathbb{A} \setminus \mathbb{B}}$.
4. $1_{\mathbb{A}\Delta\mathbb{B}}$
5. $$f(x) = \begin{cases} e^{-x} \text{ si } x \geq 0 \\ 0 \text{ sinon} \end{cases}$$
6. Est-ce que la fonction indicatrice est bijective ?
7. Fonction caractéristique et prédicats.
   (a) Le prédicat ZÉRO qui n'est vrai que pour l'entier 0.
   (b) Le prédicat SG qui n'est vrai que pour l'entier strictement positif.

(c) Le prédicat PETIT qui est vrai si $n < m$.
(d) Le prédicat ÉGALITÉ qui teste si $n = m$.
(e) Le prédicat PAIR qui teste si n est pair.
(f) Le prédicat P tel que $P(x) \iff x \neq 0$

Correction exercice 28

1. 0, 1
2. $(A \subseteq B) \iff 1_{\mathbb{A}} \leq 1_{\mathbb{B}}$
3. $1_{\mathbb{A}\backslash\mathbb{B}} = 1_{\mathbb{A}}(1 - 1_{\mathbb{B}})$
4. $1_{\mathbb{A}\Delta\mathbb{B}} = 1_{\mathbb{A}} + 1_{\mathbb{B}} - 21_{\mathbb{A}}1_{\mathbb{B}} = (1_{\mathbb{A}} - 1_{\mathbb{B}})^2 = |1_{\mathbb{A}} - 1_{\mathbb{B}}|$
5. $f(x) = e^{-x}1_{x\geq 0}$ ou $f(x) = e^{-x}1_{[0,\infty[}(x)$
6. Montrons que la fonction indicatrice est bijective :
   — Injective : $\forall A, B \in E^2, A \neq B \rightarrow 1_{\mathbb{A}} \neq 1_{\mathbb{B}}$. Si $A \neq B$ alors il existe un élément x appartenant à l'ensemble $A \cup B$ qui n'appartient pas à $A \cap B$.
   Supposons $x \in A$ et $x \notin B$ alors $1_{\mathbb{A}}(x) \neq 1_{\mathbb{B}}(x)$ et donc $1_{\mathbb{A}} \neq 1_{\mathbb{B}}$.
   — Surjective : Soit ϕ l'application de E dans $\{0, 1\}$, montrons qu'il existe un ensemble A avec $A \in \mathcal{P}(E)$ tel que $\phi = 1_{\mathbb{A}}$.
   A est bien une partie de E, et on a $\forall x \in A, \phi(x) = 1$ et $\forall x \in E\backslash A, \phi(x) = 0$ donc $\phi = 1_{\mathbb{A}}$.
7. Fonction caractéristique et prédicat
   (a)
   $$\text{ZÉRO}(x) = \begin{cases} 1 \text{ si } x = 0 \\ 0 \text{ sinon} \end{cases}$$
   (b)
   $$\text{SG}(x) = \begin{cases} 1 \text{ si } x > 0 \\ 0 \text{ sinon} \end{cases}$$
   (c) $\text{PETIT}(x, y) = \text{SG}(y - x)$
   (d) $\text{ÉGAL}(x, y) = 1 - (\text{SG}(y - x) + \text{SG}(x - y))$
   (e) $\text{PAIR}(2n) = 1$ et $\text{PAIR}(2n+1) = 0$. On peut écrire aussi : $\chi_{pair}(2n) = 1$ et $\chi_{pair}(2n+1) = 0$.
   (f) La fonction caractéristique d'un prédicat P tel que $P(x) \iff x \neq 0$. Est
   $$\chi_{P(x)} = \begin{cases} 1 \text{ si } x > 0 \\ 0 \text{ si } x = 0 \end{cases}$$
   On voit aussi que $\chi_{P(x)} = \text{SG}(x)$

Fin correction exercice 28

1.8 Théorème de Rice

Exercice 29 – Calculabilité

1. 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
   (a) une fonction totale
   (b) une fonction injective
   (c) une fonction croissante
   (d) une fonction à valeurs bornées

Correction exercice 29

1. Prédicat P_1 qui exprime qu'une procédure p calcule une fonction totale :

$$P_1(p) := \forall x.p(x) \text{ est défini}$$

   On peut construire, en utilisant le principe de la récursion, une procédure contradictoire γ, en supposant par l'absurde que le prédicat est calculable :

   Si P_1 affirme que γ_1 calcule une fonction totale, celle-ci calcule la fonction jamais définie ; sinon γ_1 calcule l'identité, qui est une fonction totale :

```
 int P1 (int p(int)){...}
int γ1 (int n) {
     If P1 (γ1) then While (1)
     else return n;
}
```

   Ceci prouve qu'il n'existe pas de procédure $P1$ qui calcule le prédicat P_1.

   Autre vision : Soit T le prédicat définit de la manière suivante :

$$\begin{cases} T(f) = 1 \text{ si } f \text{ est totale} \\ T(f) = 0 \text{ sinon} \end{cases}$$

   On pose

$$g(x) = \begin{cases} x \text{ si } T(g) = 0 \\ 0 \text{ sinon} \end{cases}$$

   Nous arrivons à une absurdité.
2. Prédicat P_2 qui exprime qu'une procédure p calcule une fonction injective :

$$P_2(p) := \forall x_1 \forall x_2.p(x_1) = p(x_2) \rightarrow x_1 = x_2$$

   si la fonction est partielle, on peut préciser

$$(p(x_1) \text{ défini } \wedge p(x_2) \text{ défini } \wedge p(x_1) = p(x_2)) \rightarrow x_1 = x_2$$

Si P_2 affirme que γ_2 calcule une fonction injective, celle-ci calcule une fonction constante ; sinon γ_2 calcule l'identité, qui est une fonction injective :

```
 int P2 (int p(int)){...}
int γ2 (int n) {
     If P2 (γ2)) then return 15
     else return n;
}
```

Ceci prouve qu'il n'existe pas de procédure $P2$ qui calcule le prédicat P_2.

Autre vision : Soit I le prédicat définit de la manière suivante :

$$\begin{cases} I(f) = 1 \text{ si } f \text{ est injective} \\ I(f) = 0 \text{ sinon} \end{cases}$$

On pose

$$g(x) = \begin{cases} x \text{ si } I(g) = 0 \\ 0 \text{ sinon} \end{cases}$$

Nous arrivons à une absurdité.

3. Prédicat P_3 qui exprime qu'une procédure p calcule une fonction croissante :

$$P_3(p) := \forall x_1 \forall x_2 . x_1 \leq x_2 \rightarrow p(x_1) \leq p(x_2)$$

si la fonction est partielle, on peut préciser

$$(x_1 \leq x_2 \wedge p(x_1) \text{ défini } \wedge p(x_2) \text{ défini }) \rightarrow p(x_1) \leq p(x_2)$$

Si P_3 affirme que γ_3 calcule une fonction croissante, celle-ci calcule une fonction périodique (valeurs successives $0, 1, 2, 0, 1, 2, \ldots$; sinon γ_3 calcule l'identité, qui est une fonction croissante :

```
 int P3 (int p(int)){...}
int γ3 (int n) {
     If P3 (γ3)) return n%3
     else return n;
}
```

Ceci prouve qu'il n'existe pas de procédure $P3$ qui calcule le prédicat P_3.

Autre vision : Soit C le prédicat définit de la manière suivante :

$$\begin{cases} C(f) = 1 \text{ si } f \text{ est croissante} \\ C(f) = 0 \text{ sinon} \end{cases}$$

On pose

$$g(x) = \begin{cases} -x \text{ si } C(g) = 1 \\ x \text{ sinon} \end{cases}$$

Nous arrivons à une absurdité.

4. Prédicat P_4 qui exprime qu'une procédure p calcule une fonction bornée :

$$P_4(p) := \exists b \forall x. p(x) \leq b$$

si la fonction est partielle, on peut préciser

$$p(x) \text{ défini } \rightarrow p(x) \leq b$$

On peut remarquer aussi qu'une fonction $f : \mathbb{N} \rightarrow \mathbb{N}$ est bornée si et seulement si $Image(f)$ est un ensemble fini (car les valeurs de f sont des entiers naturels ; sinon cette équivalence est évidemment fausse).

Si P_4 affirme que γ_4 calcule une fonction bornée, celle-ci calcule l'identité qui n'est pas bornée sinon γ_4 calcule une fonction croissante :

```
 int P4 (int p(int)){...}
int γ4 (int n) {
     If P2 (γ4)) return n
     else return 12;
}
```

Ceci prouve qu'il n'existe pas de procédure $P4$ qui calcule le prédicat P_4.

Autre vision : Soit B le prédicat définit de la manière suivante :

$$\begin{cases} B(f) = 1 \text{ si } f \text{ est borné} \\ B(f) = 0 \text{ sinon} \end{cases}$$

On pose

$$g(x) = \begin{cases} x \text{ si } B(g) = 1 \\ 12 \text{ sinon} \end{cases}$$

Nous arrivons à une absurdité.

Fin correction exercice 29

Exercice 30 – Théorème de Rice

Prouver qu'on ne peut pas décider si deux procédures P et Q calculent la même fonction.

Correction exercice 30

S'il existait un algorithme pour décider si deux procédures P et Q calculent la même fonction, on pourrait décider par exemple si P calcule l'identité, en appliquant l'algorithme au cas où Q est la procédure return x. En détail :

Si, par l'absurde, il existe une procédure PMF (MF est l'abréviation de même fonction) qui décide si P et Q calculent la même fonction, alors on peut construire la procédure PID, comme ci-dessus, pour décider si P calcule l'identité.

Algorithm 5 Algorithme qui décide si deux procédures P et Q.

```
int PMF (int P (int), int Q (int)) {...}
int ID (int x ) {RETURN x}
int PID (int P (int )) {RETURN PMF((P, ID);}
```

Note : on peut résoudre aussi cet exercice en construisant directement un couple de procédures contradictoires GAMMA1 et GAMMA2, qui calculent la même fonction si et seulement si PMF (GAMMA1, GAMMA2) dit le contraire (c'est-à-dire vaut 0). La méthode employée ci-dessus est plus rapide et intéressante : elle consiste en une réduction d'un problème A (la procédure P calcule-t-elle l'identité), à un problème B (les procédures P et Q calculent-elles la même fonction) ; si B est décidable, la réduction fournit un algorithme pour A, qui est à son tour décidable ; donc inversement (c'est le cas ici), si A est indécidable, B l'est aussi !

Fin correction exercice 30

1.9 Décidabilité et récursivement énumérable

Exercice 31 – Calculabilité

1. Est-ce que l'ensemble de décimales de π est récursivement énumérable ?
2. Est-ce que le problème de vérifier si la séquence 327 appartient-elle à l'écriture décimale d'un nombre transcendant tel que e est récursivement énumérable ?

Correction exercice 31

1. l'ensemble de décimales de π est récursivement énumérable. On sait engendrer ces décimales, mais elles constituent une suite infinie non périodique, π n'étant pas rationnel.
2. On peut engendrer les n premiers décimales de e, et voir facilement, dasn un temps fonction de n, si 327 appartient à cette séquence. Une réponse affirmative à cette question est une réponse affirmative à la première. Par contre, une réponse négative sur les n premiers chiffres suggère simplement de pousser les tests au-delà du $n$ème chiffre. L'appartenance de 327 à l'écriture décimale d'un nombre transcendant tel que e peut être considérée comme typiquement semi-décidable : s, à un moment donné « oui » n'est pas atteint, il faut poursuivre pour savoir si oui ou non. Si bien que « non » ne peut être atteint en une durée finie. Semi-décidable

Fin correction exercice 31

Exercice 32 – Récursivement énumérable

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

Correction exercice 32

Soit A un programme qui dresse la liste de tous les éléments de A. On va décrire un programme, disons B, qui dresse la liste de tous les éléments de B. On initialise le programme B avec une liste vide. Pour chaque élément a de la liste produite par A, on lance le programme P. Si $P(a) = Vrai$, le programme B rajoute l'élément a à sa liste. Sinon, il ne fait rien et passe à l'élément suivant. On énumère ainsi tous les éléments de A qui satisfont le prédicat encodé par le programme P, c'est-à-dire, tous les éléments de l'ensemble B.

Fin correction exercice 32

Exercice 33 – Calculabilité

Soit E l'ensemble $val(f)$ où f est calculable partielle.

1. Montrer que E est récursivement énumérable (inspirez-vous du fait que l'arrêt en t unités de temps est décidable)

Correction exercice 33

1. Algorithme d'énumération en version abrégée :

   `pour tout couple` (x, t)
   `Si` $h(f, x, t)$ `alors afficher` $f(x)$

   Chaque élément de $Val(f)$ est affiché une infinité de fois, ce qui est autorisé. L'exercice suivant montre comment éviter ces répétitions si on le souhaite.

Fin correction exercice 33

Exercice 34 – 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 ?

Correction exercice 34

1. Oui si f est totale

2. Voici la fonction :

```
 int fca (int x) {
Return fcb(f(x));
}
```

Donc B décidable alors A est décidable si f est totale. Sinon on montre que A est récursiment énumérable en utilisant le temps :

$\forall(x,t)$ `si` $h(f,x,t)$ `alors si` $fcb(x)$ `afficher` x;
}

Fin correction exercice 34

Exercice 35 – 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.

Correction exercice 35

1. Si f est strictement croissante, la fonction caractéristique de $E = Val(f)$ est calculable :

```
int f (int ) {...}
int χ (int x)
{
int n, y;
for (n = 0;; n++)      {
     y = f(n);
     if (x == y) return 1;
     if (x < y) return 0;
     }
}
```

Le résultat reste vrai en supprimant le mot strictement, mais alors il faut distinguer le cas où E est fini, avec les complications qui s'en suivent.

2. Pour extraire un ensemble décidable infini d'un ensemble récursivement énumérable infini, énuméré par f, il suffit d'extraire une sous-suite strictement croissante de la suite des $f(n)$:

```
int f (int ) {...}
int y, max = 0;
for (n = 0;; n++) {
     y = f(n);
     if (max < y)
     {
          max = y;
```

```
        afficher f(n)
    }
}
```

L'ensemble A énuméré par cette procédure est manifestement un sous-ensemble de E ; si E est infini, il n'est pas borné, donc on passe une infinité de fois « à l'intérieur» du if, et A est infini ; enfin par construction, A est énuméré en ordre croissant, donc est décidable.

Fin correction exercice 35

Exercice 36 – Calculabilité

1. Montrer que tout ensemble récursivement énumérable peut-être énuméré par une fonction sans répétition.
2. Quelle est la différence entre un ensemble dénombrable et un ensemble récursivement énumérable

Correction exercice 36

1. Voici l'algorithme d'énumération en version abrégée :

```
 int f (int ) {...}
int nouveau( int n)
{
int i, x = f(n);
for (i = 0; i < n; i++)
     if (x == f(i)) then return 0;
     return 1;
}
for (n = 0;; n++)
if (nouveau (n)) afficher f(n)
```

Si l'on modifie la procédure *nouveau* pour qu'elle filtre les éléments supérieurs à tous les précédents (au lieu des éléments différents de tous les précédents), en remplaçant le test d'égalité (==) par un test de comparaison (<=) :

```
 int nouveau int n
{
int i, x = f(n);
for (i = 0; i < n; i++)
     if (x <= f(i)) then return 0;
     else return 1;
}
```

l'ensemble A énuméré par cet algorithme est un sous-ensemble de E énuméré en ordre (strictement) croissant, comme précédemment (A est décidable, et si E est infini, A l'est aussi).

2. L'informatique ne manipule que le dénombrable. Mais une propriété plus exigeante est utile. Un ensemble est fini ou dénombrable quand il est vide ou l'image d'une fonction définie sur N, donc d'une certaine façon « énumérable » par une fonction définie sur les entiers. Pour qu'un ensemble soit récursivement énumérable on demande que de plus cette fonction soit calculable (au sens calculable par une machine). Il existe des ensembles dénombrables qui ne sont pas récursivement énumérables. Un exemple est l'ensemble des entiers qui codent une machine de Turing qui ne s'arrête pas pour une entrée donnée, cet ensemble ne peut être récursivement énumérable d'après l'indécidabilité du problème de l'arrêt.

Fin correction exercice 36

Exercice 37 – Calculabilité

Soient A et B deux ensembles décidables :

1. Est-on sûr que le complémentaire de A est décidable ?
2. Est-on sûr que l'union de A et B est décidable ?
3. Est-on sûr que l'intersection de A et B est décidable ?
4. Est-on sûr que $A - B$ est décidable ?
5. Même question en remplaçant décidables par récursivement énumérables. Proposer une solution en utilisant les fonctions semi-caractéristiques et les fonctions contradictoires..

Correction exercice 37

1. Il suffit de poser $\chi_{\bar{A}} = 1 - \chi_A$.
2. Il suffit de poser $\chi_{A\cup B} = \chi_A + \chi_B - \chi_A \cdot \chi_B$.
3. Il suffit de poser $\chi_{A\cap B} = \chi_A \cdot \chi_B$.
4. Il suffit de poser $A - B = A \cap (\mathbb{N} - B) = \chi_A(1 - \chi_B)$
5. Non, le complémentaire d'un ensemble récursivement énumérable n'est pas forcément récursivement énumérable (voir le cours).

   Si E et son complémentaire sont tous deux récursivement énumérables, alors ils sont décidables.

   En effet,

```
 int f int n {}
int g int n {}
int fc (int x) {
int n;
for (n = 0;; n++){
if (f(n) == x) return 1;
if (g(n) == x) return 0;
}
}
```

   Cette fois l'exécution de la procédure termine quel que soit x, car x appartient soit à E soit à

son complémentaire ; elle calcule donc bien la fonction caractéristique χ de E, autrement dit E est décidable.

On énonce parfois cette preuve sous une forme abrégée, en utilisant deux "écrans" : sur l'écran de gauche s'affichent les éléments de E, sur celui de droite les éléments du complémentaire. Pour décider si un entier x appartient à E, on attend de le voir apparaître sur l'un des deux écrans : il faut être éventuellement très patient, mais on est sûr de ne pas attendre pour rien.

6. Oui, l'union de deux ensembles récursivement énumérables est récursivement énumérable ; voici une procédure d'énumération de l'union de deux ensembles énumérés respectivement par f et g :

```
 int f (int n) {...}
int g (int n) {...}
int enumUnion (int) n{
if (n div 2 == 0)
return f(n/2)
else return g(n/2)
```

   On a que $f(0), g(0), f(1), g(1), \ldots$.

   ou plus simplement, `for (i = 0;;i++){`
   `afficher f(i); afficher g(i); }`
7. Oui, l'intersection de deux ensembles récursivement énumérable est récursivement énumérable : voici une procédure simplifiée d'affichage de l'intersection de deux ensembles énumérés respectivement par f et g. `pour tout couple` (i, j)
   `Si` $f(i) = g(j)$ `alors afficher` $f(i)$
   On peut utiliser les fonctions semi-caractéristiques :

```
 int scA (int x) {...}
int scB (int x) {...}
int ScIntersection (int) x{
int y = ScA;
return scB(y);
```

   L'intersection de A et B est donc semi-décidable, et par conséquent récursivement énumérable.
8. Pour $A - B$ c'est délicat de trouver $\mathbb{N} - B$.

Fin correction exercice 37

Exercice 38 – Calculabilité

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

Correction exercice 38

1. Voici une procédure qui énumère les éléments de E :

   `pour tout couple` (x, y)
   `Si` $(x, y) \in A$ `alors afficher` x

   On a le droit d'écrire une procédure qui utilise le test $(x, y) \in A$ car A est décidable.

   ou bien voici une procédure qui calcule la fonction semi-caractéristique de E, en utilisant la fonction caractéristique de A :

   ```
   int FcA (int x, int y) {...}
   int sc (int x) {
   int y;
   for (y = 0;; y++)
   if (FcA(x, y)) return 1;
   ```

2. E est énuméré par une fonction calculable totale f, c'est donc la (seconde) projection di graphe G de f à savoir :

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

   et G est décidable : tester si (x, y) appartient à G équivaut à tester si $y = f(x)$ (si f n'est pas totale, ce test peut boucler, mais ici f est totale).

   On peut prendre pour f $f(n) =$ le nième élément affiché.

Fin correction exercice 38

Exercice 39 – 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 $A \propto B$) si il existe une fonction totale (récursive) totale f telle que pour tout $x \in \mathbb{N}$, $x \in A$ ssi $f(x) \in B$.

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 .

Correction exercice 39

1. On a $\chi_A(x) = \chi_B \circ f(x)$

   Autre vision :

   Il existe une fonction totale h telle que $\forall x, A(x) \iff B(h(x))$. D'autre part, par définition de décidabilité χ_B est totale. Soit χ_A la fonction caractéristique de A. Donc,

   $$\chi_A(x) = \begin{cases} 1 \text{ si } A(x) \\ 0 \text{ si } \neg A(x) \end{cases} = \begin{cases} 1 \text{ si } B(h(x)) \\ 0 \text{ si } \neg B(h(x)) \end{cases} = \chi_B(h(x)).$$

Donc χ_A est totale puisque elle est obtenue par la composition de χ_B et h qui son totales. Alors A est décidable.

2. Soit b une fonction récursive partielle de domaine B. La fonction $a = b \circ f$ pour domaine A ce qui montre que A est récursivement énumérable.

Fin correction exercice 39

2 Complexité

2.1 Rappel

Exercice 40 – 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 sont vérifiées 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 : $f_0 = 0, f_1 = 1 = f_2, n > 1, f_n = f_{n-1} + f_{n-2}$. 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 en 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 ?

Correction exercice 40

1. La complexité est polynomial en $O(x^4) + O(x^2) = O(x^4)$.

   y est de valeur inférieure à x^2, et sachant que $ph(y)$ est linéaire en y donc en $O(y)$ on arrive à $O(x^2)$.
2. On peut rien dire ce n'est peut-être pas le programme optimal.

3. oui
4. L'ensemble se fait en temps polynomial si chaque appel de fonction est polynomial.
5. Soit $t(n)$ le nombre d'addition au rang n. On obtient $t(n) = 1 + t(n-1) + t(n-2)$. La borne supérieure est donné par la solution de l'équation de récurrence : $t_n = 2+2t_{n-1}$, donc la valeur est 2^n. Pour la borne inférieure, on a $t_n = 2 + 2t_{n-2}$. On trouve $\sum_{i=1}^{n/2-1} 2^i$

   $\sum_{i=0}^{n} 2^i = \frac{2^{n+1}-1}{2-1}$

   Sachant que $\sum_{i=0}^{n} 2^i = 1 + \sum_{i=1}^{n/2-1} 2^i + 2^{n/2} + \sum_{n/2+1}^{n} 2^i$

   Soit $Y = \sum_{n/2+1}^{n} 2^i = \sum_{j=0}^{n/2-1} 2^j 2^{n/2+1}$. On a $\sum_{i=0}^{n/2-1} 2^i = \frac{2^{n/2}-1}{2-1}$ et donc $Y = 2 * 2^{n/2}(2^{n/2} - 1)$.

   Ainsi on trouve $\sum_{i=1}^{n/2-1} 2^i = 2^{n+1} - 1 - 1 - 2^{n/2} - 2^{n+1} + 2 * 2^{n/2} = 2^{n/2}$.
6. En mémorisant les calculs intermédiaire.
7. En utilisant un calcul matriciel avec

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix}$$

   et un algorithme logarithmique en n pour calculer A^n.

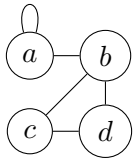
Fin correction exercice 40

Exercice 41 – Sur le codage d'un graphe

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

Correction exercice 41

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.

— On code n le nombre de sommets puis les m arêtes comme un couple de sommets : $(2m+1)\log(n+1)$ bits

— On code n le nombre de sommets puis la matrice d'adjacence $\log(n+1) + n^2$ bits.

Fin correction exercice 41

Exercice 42 – 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 connait 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.

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 P ? Quel serait l'impact si ce problème était dans P ?

Correction exercice 42

Non car pour coder n, il faut $\log_2 n$ bits et la taille de la donnée est donc $k = \log_2 n \Rightarrow n = 2^k$ et $\sqrt{n} = 2^{k/2}$. Donc l'algorithme n'est pas polynomial. Heureusement car résoudre ce problème permettrait de mettre à mal le système de codage des cartes bleues.

Fin correction exercice 42

Exercice 43 – 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 $1s$?.

Correction exercice 43

La puissance des machines est multipliée par $2^{3^{10}} = 2^{30} = 10^9$. Avec l'algorithme en n^3 la taille est multiplie par deux tous les 4 ans donc par 2^{10} en 40 ans, donc les problèmes traitables sont de l'ordre de 10000. Avec l'algorithme en $O(2^n)$ en ajoute trois tous les 4 ans donc la taille des problèmes traités est 40.

1. Nous avons de manière classique que

$$\begin{aligned} \frac{T_A(n_1)}{n_1^3} &= \frac{T_A(n_0)}{n_0^3} \\ T_A(n_0) \times n_1^3 &= T_A(n_1) \times n_0^3 \\ T_A(n_0) \times n_1^3 &= 10^9 \times T_A(n_0) \times n_0^3 \\ n_1^3 &= 10^9 n_0^3 \\ n_1 = 1000 n_0 = 10000 & \end{aligned}$$

2. l'autre c'est $2^{n_1} = 10^9 2^{n_0}$ ainsi on trouve $n_1 \log 2 = 9 \log 10 + n_0 \log 2$ et donc $n_1 = 9/\log 2 + n_0 = 40$

Fin correction exercice 43

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

Exercice 44 – 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 deux instances suivantes :
   — $\phi = (x_1 \vee x_2) \wedge (x_3 \vee \bar{x}_4) \wedge (x_1 \vee x_4) \wedge (x_1 \vee x_3) \wedge (x_4 \vee \bar{x}_2)$.
   — $\phi = (x_1 \vee x_3) \wedge (x_2 \vee \bar{x}_4) \wedge (x_1 \vee x_4) \wedge (x_4 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3)$.
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 ?

Correction exercice 44

1. Evident
2. Pour la première formule c'est oui. Pour la seconde c'est non.

   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).

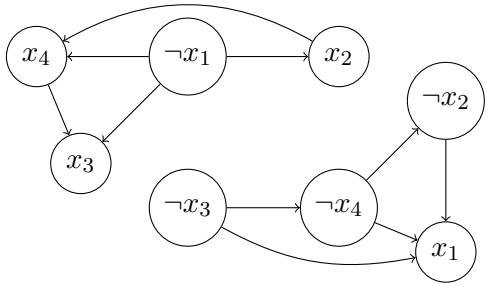
Le but de l'algorithme est d'abord de trouver des contradictions : si dans une composante fortement connexe du graphe se trouvent a et $\neg a$, alors on a une formule du style $a \Leftrightarrow \neg a$, ce qui est absurde.

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 :

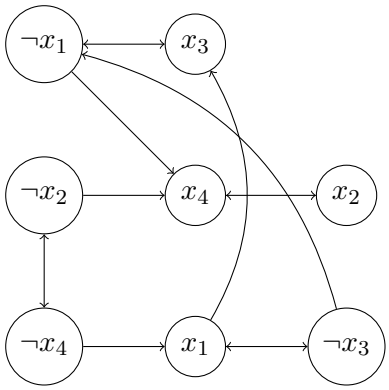


Dans ce graphe, il y a 2 composantes connexes, et chaque composante fortement connexe est composée d'un seul sommet (il n'y a pas de cycle). On peut en déduire que la formule est satisfiable. Une affectation possible est $x_1 = \top$, $x_2 = \top$, $x_3 = \top$, $x_4 = \top$.

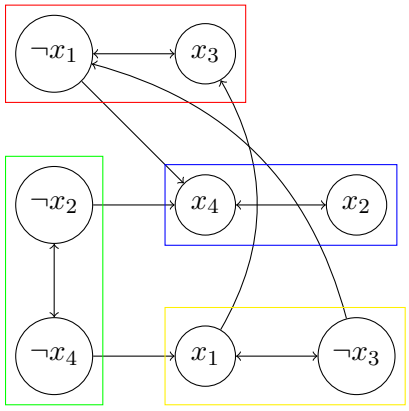
Le cas de ϕ_2 est plus intéressant :

$$\begin{aligned} \phi_2 \quad = \quad & ((\neg x_1 \Rightarrow x_3) \vee (\neg x_3 \Rightarrow x_1)) \wedge ((\neg x_2 \Rightarrow \neg x_4) \vee (x_4 \Rightarrow x_2)) \wedge ((\neg x_1 \Rightarrow x_4) \vee (\neg x_4 \Rightarrow x_1)) \\ & \wedge ((\neg x_4 \Rightarrow \neg x_2) \vee (x_2 \Rightarrow x_4)) \wedge ((x_1 \Rightarrow \neg x_3) \vee (x_3 \Rightarrow \neg x_1)) \\ & \wedge ((x_1 \Rightarrow x_3) \vee (\neg x_3 \Rightarrow \neg x_1)) \end{aligned}$$

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. On peut étudier Max-2-SATISFAISABILITÉ

Fin correction exercice 44

Exercice 45 – Algorithme non-déterministe pour le problème de 3-COLORATION

Proposer un algorithme linéaire ($O(n)$) pour le problème 3-COLORATION.

Correction exercice 45

Algorithm 6 Algorithme non-déterministe pour 3-COLORATION.

```
Coloriable := vrai ;
i = 0 ;
créer un tableau Couleur ;
while i ≤ |V| ∧ Coloriable do
    i := i + 1 ;
    Guess Couleur(i)
    if Couleur(i) ∈ Couleur({voisins de i}) then
        Coloriable := faux ;
    end if
end while
```

Fin correction exercice 45

Exercice 46 – Classification dans $\mathcal{NP}$ ou dans $\mathcal{P}$

Problème P1

Entrée : $G = (V, E)$ un graphe non orienté.
Question : Existe t'il un cycle de longueur égale à $\lfloor \frac{|V|}{2} \rfloor$?

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 simple chemin 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 mois de k feuilles ?

1. Classer les problèmes en fonction de $\mathcal{P}$ et $\mathcal{NP}$.

Correction exercice 46

1. Problème P1

   Le problème $P1$ est dans $\mathcal{NP}$car étant donnée une suite de sommets s, on peut vérifier en temps polynomial si s est un cycle de de longueur égale à $\lfloor \frac{|V|}{2} \rfloor$ dans G (en $O(n^2)$ opérations).
2. Problème P2

   Le problème $P2$ est dans $\mathcal{NP}$car étant donnée une suite de sommets s, on peut vérifier en temps polynomial si s est un cycle de de longueur égale à 4 dans G (en $O(n)$ opérations). Le problème P2 est dans $\mathcal{P}$. Considérons l'algorithme suivant. Pour tout 4-uplets de sommets (x, y, z, t) vérifier s'il est un cycle de de longueur égale à 4. Si c'est un cas, pour un seul 4-uplets, alors répondre oui sinon répondre non. La complexité de cet algorithme est $O(n^5)$ (il y a $O(n^4)$ 4-uplets et tester s'il est un cycle de de longueur égale à 4 nécessite $O(n)$ opérations.
3. Problème P3

   Le problème P2 est dans $\mathcal{NP}$car étant donnée une suite de sommets P,on peut vérifier en temps polynomial si P est un simple chemin entre u et v de longueur inférieure ou égale à k (en $O(n)$ opérations).

   Le problème P2 est dans $\mathcal{P}$. Considérons l'algorithme suivant. Il suffit simplement de calculer le plus court chemin entre u et V. Si la longueur du chemin est plus grande que k, ou si il n'existe pas de chemin entre u et v alors répondre oui sinon répondre non.
4. Problème P4 $\mathcal{NP}$-complet (réduction avec chaîne Hamiltonienne en prenant $k = 2$).

Fin correction exercice 46

2.3 Réduction polynomiale

Exercice 47 – 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 qui est plus grand que n, et B le problème de décision $B = \{n | n \text{ est premier } \}$.

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

Correction exercice 47

1. Facile
2. L'algorithme suivant est une réduction de A à B :

```
Nextprime(n)
k = n + 1;
While k ∉ B do
    k++;
endWhile
Return k;
```

Fin correction exercice 47

Exercice 48 – 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 soient 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 : Soit $a \in \mathbb{N}$.
Question : Peut-on élever au carré a ?

Correction exercice 48

Supposons que l'on veuille faire des calculs sur les entiers et que l'addition, la soustraction et la division soient des opérations autorisées. On veut savoir si faire une multiplication (problème Problème de la multiplication) est plus ou moins difficile que d'élever un nombre au carré (problème Problème du carré d'un entier). Un sens est facile : si l'on sait faire une multiplication, on peut élever un nombre au carré en le multipliant par lui-même, donc Problème du carré d'un entier $\propto$ Problème de la multiplication. Mais l'autre inégalité Problème de la multiplication $\propto$ Problème du carré d'un entier est vraie aussi, grâce à la formule :

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

Les deux problèmes sont aussi difficiles l'un que l'autre (ils peuvent être réduits l'un à l'autre).

Fin correction exercice 48

Exercice 49 – Réduction entre deux problèmes polynomiaux

2-Satisfaisabilité

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

Question : Existe-t-'il une affectation de valeurs de vérité aux variables qui satisfasse ϕ ?

2-coloration

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

Question : Existe-t'il une 2-coloration valide, *i.e.* une fonction totale $f : \mathbb{N} \to \{1, 2\}$ tel $f(u) \neq f(v), \forall (u, v) \in E$?

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

Correction exercice 49

1. Nous procédons à une réduction entre 2-coloration et 2-Satisfaisabilité.

   Construction 1 — $\forall v \in V$, *on a une variable* x_v.
   — $\forall (u, v) \in E$ *nous créons les deux clauses :* $(x_v \vee x_u)$ *et* $(\bar{x}_v \vee \bar{x}_u)$

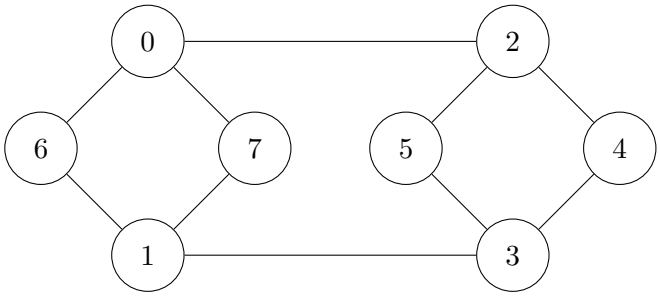
   Les deux clauses ainsi créées exprime le fait que les deux variables ne peuvent pas être simultanément vrai ou simultanément faux.

   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 ($\alpha(u) \neq \alpha(v)$).

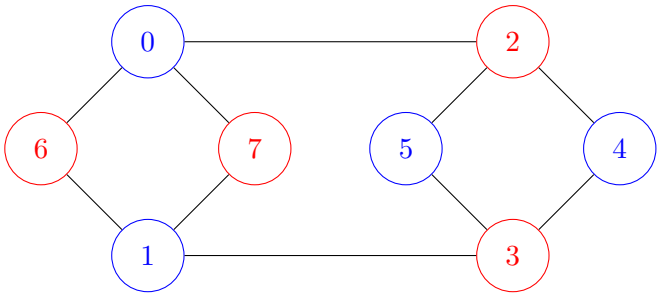
La réduction est alors assez évidente : il suffit, pour chaque arête, de faire un xor. On se retrouve donc avec la formule φ suivante :

$$\varphi = \bigwedge_{(u,v)\in E} (u \vee v) \wedge (\overline{u} \vee \overline{v})$$

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 :

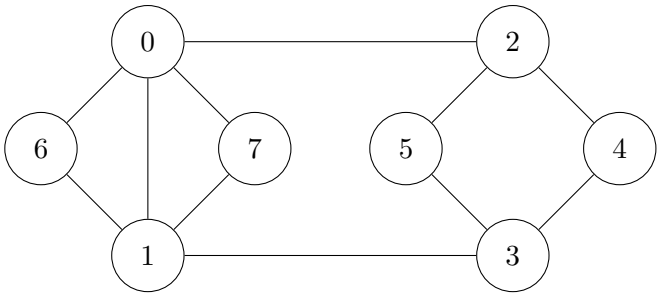
$$\begin{aligned}\varphi = (x_0 \vee x_2) \wedge (\neg x_0 \vee \neg x_2) \wedge (x_0 \vee x_6) \wedge (\neg x_0 \vee \neg x_6) \wedge (x_0 \vee x_7) \wedge (\neg x_0 \wedge \neg x_7)\\ \wedge (x_1 \vee x_6) \wedge (\neg x_1 \vee \neg x_6) \wedge (x_1 \vee x_7) \wedge (\neg x_1 \vee \neg x_7) \wedge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

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 bicolorié que j'ai donné un peu plus haut en exemple.

2. Le problème 2-Satisfaisabilité est polynomial.

Fin correction exercice 49

Exercice 50 – Problèmes équivalent 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 (Minimum spanning tree)
   Entrée : Un graphe $G = (V, E)$.
   But : Trouver un sous-graphe connexe de poids minimum.

   Arbre couvrant de poids maximum (Maximum spanning tree)
   Entrée : Un graphe $G = (V, E)$.
   But : Trouver un sous-graphe connexe de poids maximum.

2. Arborescence des plus courts chemins et Plus long Chemin

   Arborescence des plus courts chemins (Shortest Path)
   Entrée : $G = (V, E)$ un graphe orienté.
   But : Trouver une arborescence des plus court chemins.

   Un plus long chemin (Longuest Path)
   Entrée : $G = (V, E)$.
   But : Trouver un plus long chemin sans répétition de sommets.

3. Coupe de valeur maximale et Coupe de valeur minimale

Coupe de valeur maximale (Maximum Cut)

Entrée : $G = (V, E)$ un graphe orienté.
But : Trouver une coupe de valeur maximale.

Coupe de valeur minimale (Minimum Cut)

Entrée : $G = (V, E)$.
But : Trouver une coupe de valeur minimale.

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

   Couplage maximum de valeur maximum

   Entrée : $G = (V, E)$ un graphe, et une valuation sur les arêtes.
   But : Trouver un couplage maximum de poids maximum.

   Couplage maximum de valeur minimum

   Entrée : $G = (V, E)$ un graphe, et une valuation sur les arêtes.
   But : Trouver un couplage maximum de poids minimum.

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

   Voyageur de Commerce de coût minimum (Maximum TSP)

   Entrée : $G = (V, E)$ un graphe complet, et une valuation sur les arêtes.
   But : Trouver un chemin Hamiltonien de poids minimum.

   Voyageur de Commerce de coût maximum (Minimum TSP)

   Entrée : $G = (V, E)$ un graphe complet, et une valuation sur les arêtes.
   But : Trouver un chemin Hamiltonien de poids maximum.

Correction exercice 50

1. Les deux problèmes sont équivalents, il suffit de poser $-w(e)$ pour le minimum.
2. Arborescence des plus courts chemins est polynomial tandis que Un plus long chemin est $\mathcal{NP}$-complet (équivalent à la Chemin Hamiltonien).
3. Coupe de valeur minimale admet un algorithme polynomial (Ford-Fulkerson) tandis que Coupe de valeur maximale est $\mathcal{NP}$-complet.
4. Les deux problèmes sont polynomiaux en posant $-w(e)$ selon les cas.
5. Les deux problèmes sont $\mathcal{NP}$-complets car on a montré que Voyageur de Commerce avec $= k$ est $\mathcal{NP}$-complet.

Fin correction exercice 50

Exercice 51 – 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 connaitre la distance entre deux sommets disjoints dans un graphe ;
3. de connaitre la longueur de la chaîne maximum dans un graphe pondéré.

Correction exercice 51

1. Problème de décision de savoir s'il existe un chemin entre deux sommets disjoints

   Existence d'un chemin entre deux sommets

   Entrée : Un graphe orienté $G = (V, E)$, deux sommets distincts u et v.
   Question : Existe-t'il un chemin entre u et v dans G ?

   La taille de l'instance est $O(V^2)$ car il faut coder le graphe ($O(V^2)$) et les étiquettes de deux sommets (O(logV)).
2. Problème de décision de connaitre la distance entre deux sommets dans un graphe :

   Existence d'un chemin entre deux sommets de longueur k

   Entrée : Un graphe orienté $G = (V, E)$, deux sommets distincts u et v et un entier k.
   Question : Existe-t'il un chemin entre u et v dans G telle que sa longueur soit inférieure à k ?

   La taille de l'instance est $O(V^2)$ car il faut coder le graphe, les étiquettes de deux sommets, l'entier k qui est $k \leq |V|(O(logV))$.
3. Problème de décision de connaitre la longueur de la chaîne maximum dans un graphe.

   Existence d'une chaîne maximum

   Entrée : Un graphe $G = (V, E)$, une fonction $w : E \to \mathbb{N}$ deux sommets distincts u et v et un entier k.
   Question : Existe-t'il une chaîne entre u et v dans G telle que sa longueur soit supérieure à k ?

   Nous allons calculer la taille de l'instance. Il faut coder le graphe, les étiquettes de deux sommets, l'entier k qui est $k, \leq |V|$. Cela nécessite $O(V^2)$ bits. Il faut coder en plus la fonction de poids. Soit $w_{max} = \max\{w(e) : e \in E\}$. Coder la fonction de poids nécessitent de coder $O(V^2)$ entiers qui sont inférieure à w_{max}. La taille de l'instance est $O(V^2 \log w_{max})$ si les entiers sont codés en binaire, sinon $O(V^2 * w_{max})$.

Fin correction exercice 51

Exercice 52 – Optimisation versus décision

Soit le problème du stable de taille k :

Stable (Stable)

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

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 (MaxStable)

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

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, lui 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.

Correction exercice 52

1. il est facile de voir que la solution de stabilité maximum donne le plus grand k, noté k^* pour lequel le problème est résoluble. Il suffit que comparer k et k^*.
2. Recherche dichotomique du fait de la monotonie.
3. Par contre reste encore ouverte la question de l'existence d'un problème dont la version optimisation est plus difficile à résoudre que sa contrepartie décisionnelle. Le résultat suivant montre que la réponse à cette question pourrait être affirmative. Si $\mathcal{P} \neq \mathcal{NP} \cap co-\mathcal{NP}$, alors il existe un problème tel que sa version décisionnelle est polynomial tandis que sa version décisionnelle est polynomial tandis que sa version optimisation ne l'est pas.

   Stable de taille $k \Rightarrow$ stable de taille $k-1$.

Fin correction exercice 52

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

2.4.1 Certificat

Exercice 53 – 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=$\{< G, s, t, > |G$ est un graphe orienté possédant un chemin de s à $t\}$

Correction exercice 53

1. 2-Partition. 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 7 $\pi_{\text{2-Partition}}(S_1, S_2)$

```
if |S1| ≠ |S2| then
  Return(non)
else if Σ_{s1∈S1} s1 ≠ Σ_{s2∈S2} s2 then
  Return(non)
else
  Return(oui)
end if
```

2. Circuit Hamiltonien

   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 ($O(|V| \times |C| + |E| \times |C|)$) :

3. Satisfaisabilité

   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 $O(|\phi|)$) :
4. Clique 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) :
5. 3-coloration : Dans ce cas, 3-coloriable signifie qu'en prenant trois couleurs, 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 8 $\pi_{\text{Circuit\_Hamiltonien}}(G = (V, E), C)$

```
if C_[0] ≠ C_[-1] ou (C_[0], C_[0]) ∉ E then
  Return(non)
end if
seen := ∅
for i = 0 à |C| − 1 do
  if (C_[i], C_[i+1]) ∉ E then
    Return(non)
  end if
  seen := seen ∪ {C_[i], C_[i+1]}
end for
for v ∈ V do
  if  then

  end if
end for
Return(oui)
```

Algorithm 9 $\pi_{\text{Clique}}(G, S)$

```
for s ∈ S do
  for (u, v) ∈ E do
    if u est s then
      marquer v
    else if v est s then
      marquer u
    else if ∃v ∈ S tel que v n'est pas marqué then
      Return(non)
    end if
  end for
end for
Return(oui)
```

Algorithm 10 $\pi_{\text{3-Coloration}}(G = (V, E), C)$

```
for (u, v) ∈ E do
  if C_[u] = C_[v] then
    Return(non)
  end if
end for
Return(oui)
```

— quelle est la taille de la donnée du problème ? Taille $|G| = \theta(|S| + |A|)$
— quelle est la taille du certificat ? $\theta(|S|)$

Tout ceci est vérifiable en temps polynomial :

Algorithm 11 Algorithme de vérification de complexité $O(|A|)$ (liste d'adjacente)

```
for Chaque arête (i, j) de A do
  if T[i] = T[j] then
    retourner faux ;
  end if
end for
Retourner vrai ;
```

(a) Un algorithme en $O(mn)$ (n variables avec m clauses) permet de vérifier si une affectation u satisfait la formule f

(b) Si G est k-coloriable ssi $\exists$ une coloration valide. Une coloration valide c admet une taille polynomiale. Donc, avec G et c on peut vérifier en $O(m)$ si G admet une coloration c réalisable.

(c) Soient n entiers $A_1, A_2, \ldots, A_n$ et $T \in \mathbb{N}$, est-ce qu'il existe un sous-ensemble de nombres dont la somme est au plus T ? $\in NP$

(d) Un parcours en largeur en $O(n + m)$ suffit.

1. 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.
2. 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.
3. 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.
4. Pour le co-problème de la Clique, il faudrait aussi tester la combinaison de tous les sommets, ce qui n'est pas polynomial.
5. Pour le co-problème de la 3-Coloration, il faudrait énumérer tous les cas, ce qui n'est pas polynomial non plus.

Fin correction exercice 53

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

Exercice 54 – Propriétés des classes $\mathcal{NP}$ et $\mathcal{NP}$-complet

Soient A et B deux langages.

Prover 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 $A \cup B$ et ni $A \cap B$ peuvent être $\mathcal{NP}$-complet. Donner un exemple de problème A et B.

Correction exercice 54

1. Si A et B sont dans $\mathcal{NP}$, alors on a $A \cup B \in \mathcal{NP}$ et $A \cap B \in \mathcal{NP}$.

   Si $A \in \mathcal{NP}$(resp. $B \in \mathcal{NP}$, alors il existe une machine de Turing (ou certificat) V_A (resp. V_B) tel que $x \in A$ si et seulement si $\exists y, |y| \leq p(|x|)$ et V_A accepte le couple $\langle x, y\rangle$.

   — Pour $A \cup B$: nous définissons la machine $V_{A\cup B}$, qui calcule les deux certificats V_A et V_B sur l'entrée $\langle x, y\rangle$ et qui renvoie oui si le deux certificats V_A et V_B acceptent $\langle x, y_A\rangle$ (resp. $\langle x, y_B\rangle$). Pour $x \in A \cup B$, il y a une chaîne de caractères y_A tel que V_A accepte $\langle x, y_A\rangle$ ou il y a une chaîne de caractères y_B tel que V_B accepte $\langle x, y_B\rangle$. En posant $y = y_A$ ou $y = y_B$, $V_{A\cup B}$ acceptera $\langle x, y\rangle$.

   — Pour $A \cap B$: de manière similaire , nous pouvons définir $V_{A\cap B}$, tel que l'entrée $\langle x, y_A, y_B\rangle$ est acceptée si et seulement si V_A (resp. V_B) accepte $\langle x, y_A\rangle$ (resp. $\langle x, y_B\rangle$).

2. Si A et B sont $\mathcal{NP}$-complet, alors on a $A \cup B \in \mathcal{NP}$-complet et $A \cap B \in \mathcal{NP}$-complet.

   — Regardons l'intersection : Soit L un langage $\mathcal{NP}$-complet. Nous allons définir les deux langages suivants :

   $$A = 0L = \{0x | x \in L\}$$

   et

   $$B = 1L = \{1x | x \in L\}$$

   Nous pouvons conclure que A et B sont $\mathcal{NP}$-complets. En effet, n'importe quelle réduction à partir de SATISFAISABILITÉ à L peut-être convertie en une réduction pour A en ajoutant un 0 à la sortie, (resp. pour B). Il est facile de voir que les deux appartiennent à $\mathcal{NP}$ si $L \in \mathcal{NP}$. Cependant $A \cap B = \varnothing$ qui ne peut-être $\mathcal{NP}$-complet.

   — Regardons l'union : si A et B sont $\mathcal{NP}$-complets alors $\overline{A}$ et $\overline{B}$ sont *co*$\mathcal{NP}$-complets alors en montrant que $A \cup B$ ne peut appartenir à la classe $\mathcal{NP}$-complets, et c'est équivalent à montrer que $\bar{A} \cap \bar{B}$ n'est pas dans *co*$\mathcal{NP}$-complet.

   Ainsi, pour un problème $\mathcal{NP}$-complet L, on peut prendre $\overline{A} = 0\overline{L}$ et $\overline{B} = 1\overline{L}$.

   Ceci donne

   $$A = \overline{0\overline{L}} = (1\{0,1\}^*) \cup \{0x | x \in L\}$$

   et

   $$B = \overline{1\overline{L}} = (0\{0,1\}^*) \cup \{1x | x \in L\}$$

   Alors, notons que les réductions pour L nous pouvons modifier ces réductions en des réductions pour A et B, en ajoutant 0 et 1 respectivement au début. Alors A et B sont $\mathcal{NP}$-complets. Pourtant $A \cup B = \{0,1\}^*$ ce qui ne peut-être $\mathcal{NP}$-complet.

Autre manière : Pour vérifier si un mot w appartient à $L_1 \cap L_2$, il suffit de vérifier s'il appartient à L_1 , et à L_2, et de combiner les résultats obtenus. Si L_1 et L_2 sont tous deux dans $\mathcal{P}$ou dans $\mathcal{NP}$, $L_1 \cap L_2$ et $L_1 \cup L_2$, sont donc aussi respectivement dans $\mathcal{P}$ ou $\mathcal{NP}$.

Pour l'appartenance à $\mathcal{NP}$-complet, la situation est différente. En effet, l'appartenance à $\mathcal{P}$ ou $\mathcal{NP}$indique une limite supérieure sur les ressources nécessaires à reconnaître un langage, tandis que l'appartenance à $\mathcal{NP}$-complet indique que le langage est au moins aussi difficile que tout autre langage de $\mathcal{NP}$. L'intersection ou l'union peuvent radicalement changer cette situation. En effet, il est parfaitement possible que deux langages de $\mathcal{NP}$-complet aient le langage vide comme intersection. Il suffit que, par exemple pour une question d'encodage, les mots de L_1 ne puissent jamais être des mots de L_2.

L'union de deux langages de $\mathcal{NP}$-complet peut aussi être langage plus simple. Par exemple, le problème SATISFAISABILITÉ est $\mathcal{NP}$-complet. Le problème de déterminer s'il existe une fonction d'interprétation qui rend une formule booléenne générale fausse est aussi dasn $\mathcal{NP}$-complet. Comme pour toute formule il existe forcément une fonction d'interprétation qui la rend vraie ou fausse, l'union de ces problèmes se réduit à déterminer si le mot fourni est bien l'encodage d'une formule booléenne, ce qui est soluble en temps polynomial. Cet exemple montrer que l'union de deux problèmes de $\mathcal{NP}$-complet peut être un problème de $\mathcal{P}$.

Exemple :

STABLE et RECOUVREMENT DE SOMMETS sont des candidats ?

Fin correction exercice 54

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

Exercice 55 – Définition de $co\mathcal{NP}$ par les langages formels

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

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

Correction exercice 55

Soit $A \in co\mathcal{NP}$: par définition, $\bar{A} \in \mathcal{NP}$, donc d'après la caractérisation de $\mathcal{NP}$ (un langage A est dans $\mathcal{NP}$ si et seulement si un polynôme $p(n)$ et un langage $B \in \mathcal{P}$ tels que $x \in A \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)}(x,y) \in B$), il existe $B' \in \mathcal{P}$et un polynôme $p(n)$ tels que

$$x \in {}^cA \Leftrightarrow \exists y \in \{0,1\}^{p(|x|)}(x,y) \notin B'$$

On en déduit que

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

Il suffit de prendre $B = {}^cB'$ (qui est bien un langage de $\mathcal{P}$) pour obtenir notre caractérisation

Fin correction exercice 55

Exercice 56 – 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. $A \cap B \in co\mathcal{NP}$.

Correction exercice 56

On a par définition que $A \in \mathcal{NP}$implique que $\bar{A} \in co\mathcal{NP}$et $B \in co\mathcal{NP}$implique que $\bar{B} \in \mathcal{NP}$

1. $\bar{A} \cup B \in co\mathcal{NP} \Rightarrow$ oui
   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. $A \cap \bar{B} \in co\mathcal{NP}$. Sachant que $A \in \mathcal{NP}$et $\bar{B} \in \mathcal{NP}$alors l'intersection de deux problèmes $\mathcal{NP}$ reste dans $\mathcal{NP}$, alors $co\mathcal{NP}$=$\mathcal{NP}$
   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 $B = \varnothing$ 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. $A \cup B \in co\mathcal{NP}$. On a $\overline{A \cup B} = \bar{A} \cap \bar{B}$ Pad d'idée mais je pense que c'est faux. Par exemple $A = 0L$ et $\bar{B} = 1\bar{L}$ $A \in$ NP si $L \in \mathcal{NP}$ et $\bar{B} \in co\mathcal{NP}$car $\bar{L} \in co\mathcal{NP}$. Alors $\bar{A} = 0\bar{L}$. Donc $\bar{A} \cap \bar{B} = \varnothing$ et donc $A \cup B = \{0,1\}^*$.
   Essayons de trouver un contre-exemple : soit $\varnothing$ le langage vide. Posons $B = \varnothing$. On sait que $\varnothing \in \mathcal{NP}$, 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 $B = \varnothing$ 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. $A \cap B \in co\mathcal{NP}$.

   C'est faux en général. Si tu prends A un problème $\mathcal{NP}$-complet, l'intersection avec B est vide. 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 $\varnothing \in \mathcal{NP}$, on a Σ^* (qui est le complémentaire de $\varnothing$) dans $co\mathcal{NP}$. On a donc $A \cap B = A \in \mathcal{NP}$, donc $A \cap B \notin co\mathcal{NP}$.

Fin correction exercice 56

Exercice 57 – Absence de certificat positif

Considérons le problème suivant :

CO-VOYAGEUR DE COMMERCE (COTSP)

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-il aucun cycle Hamiltonien de distance inférieure ou égale à k ?

Est-ce que ce problème appartient à la classe $\mathcal{NP}$?

Correction exercice 57

Il y a peu de chance pour que le problème CO-VOYAGEUR DE COMMERCE appartienne à la classe $\mathcal{NP}$. En effet, il faut vérifier tous les sous-ensembles donc la complexité n'est pas polynomiale.

Fin correction exercice 57

Exercice 58 – 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 $co\mathcal{NP}$=$\mathcal{NP}$?

Correction exercice 58

1. Sachant que π est $\mathcal{NP}$-complet et $\bar{\pi} \in \mathcal{NP}$ alors $\bar{\pi} \propto \pi$ et également $\pi \propto \bar{\pi}$. Pour n'importe quel autre problème $\pi_1 \in \mathcal{NP}$, nous avons que $\pi_1 \propto \pi \propto \bar{\pi}$. Sachant que $co\mathcal{NP}$est fermé par la réduction polynomial $\propto$, alors $\pi_1 \in co\mathcal{NP}$ et $\mathcal{NP} \subseteq co\mathcal{NP}$.

   De manière similaire, on peut prouver que $co\mathcal{NP} \subseteq \mathcal{NP}$(rappelons selon la propriété des réductionnismes, le complément $\bar{\pi}$ d'un problème $\mathcal{NP}$-complet π est $co\mathcal{NP}$-complet.

2. Sachant que $\mathcal{P}$=$co\mathcal{P}$, et supposons que $co\mathcal{NP}\neq\mathcal{NP}$et $\mathcal{P}\neq\mathcal{NP}$. Dans ce cas les deux résultats suivant serait valide $co\mathcal{NP}$=$\mathcal{NP}$et $co\mathcal{NP}\neq\mathcal{NP}$.
3. Il est possible d'avoir $\mathcal{P}\neq\mathcal{NP}$ et $co\mathcal{NP}$=$\mathcal{NP}$bien que cela peut probable.

Fin correction exercice 58

Exercice 59 – 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 : Etant donné une formule booléenne
   Question : Existe-t-il une assignation de ses variables qui la rend vraie ?
2. Chemin Hamiltonien (HC)
   Entrée : $G = (V, E)$
   Question : Existe-t'il un chemin Hamiltonien ?
3. Clique (Clique)
   Entrée : $G = (V, E)$, $k \in \mathbb{N}^*$
   Question : Existe-t'il une clique de taille k ?
4. Montrer que les problèmes dual ont $co\mathcal{NP}$-complets.

Correction exercice 59

1. UNSAT (UNSAT)
   Entrée : Etant donné une formule booléenne
   Question : Est-elle vraie que la formule est insatisfiable pour toutes les assignations de ses variables ?
2. Co-Hamiltonien (Co-HC)
   Entrée : $G = (V, E)$
   Question : Est-il vrai qu'il n'existe aucun chemin Hamiltonien ?
3. Co-Clique (Co-Clique)
   Entrée : $G = (V, E)$, $k \in \mathbb{N}$
   Question : Est-il vrai que G ne possède pas de clique de taille k ?
4. Prouvons la $co\mathcal{NP}$-complétude des problèmes duals.
   (a) Montrons l'appartenance à $co\mathcal{NP}$:
      — Pour UNSAT : ¬ UNSAT =Satisfaisabilité
      — Pour Co-Hamiltonien : Chemin Hamiltonien est $\mathcal{NP}$
      — Pour Co-Clique : Clique est $\mathcal{NP}$
   (b) Montrons la complétude :
      — Pour UNSAT : Soit $A \in co\mathcal{NP}$. Nous montrons $A \propto$ UNSAT. Pour une entrée w transformes w en une formule ϕ, $w \in \neg A \Rightarrow \phi \in$ Satisfaisabilité, alors $w \notin A \Rightarrow \phi \notin$ UNSAT. Et $w \notin \neg A \Rightarrow \phi \notin$ Satisfaisabilité, alors $w \in A \Rightarrow \phi \in$ UNSAT.

— Pour Co-Hamiltonien : Utiliser le théorème que Chemin Hamiltonien est $\mathcal{NP}$-complet
— Pour Co-Clique : Utiliser le théorème que Clique est $\mathcal{NP}$-complet

Fin correction exercice 59

Exercice 60 – $co\mathcal{NP} \cap \mathcal{NP}$

Considérons le problème Problème du flot maximum.

Problème du flot maximum
Entrée : $G = (V, E, c)$ une source s et 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 Problème du flot maximum $\in \mathcal{NP} \cap co\mathcal{NP}$.

Correction exercice 60

— Si G est une instance positive, alors la preuve est juste le flot de de valeur d'au moins K.
— Si G est une instance négative, il suffit de calculer la valeur de la coupe minimale.

Coupe de valeur minimale
Entrée : $G = (V, E, c)$ une source s et t, et $K \in \mathbb{N}$.
Question : Est-il vrai que G possède une coupe d'au plus K entre s et t ?

Now by the max flow-min cut theorem, the network has a flow of value at least K iff it does not have a cut of capacity $K - 1$ or less.

Fin correction exercice 60

Exercice 61 – $co\mathcal{NP} \cap \mathcal{NP}$

Considérons le problème Problème du couplage parfait.

Problème du couplage parfait
Entrée : $G = (U \cup V, E)$, un graphe biparti.
Question : Est-il vrai que G possède un couplage parfait ?

Montrer que Problème du couplage parfait $\in \mathcal{NP} \cap co\mathcal{NP}$.

Correction exercice 61

— Si G est une instance positive, alors la preuve est juste le couplage parfait.

— Si G est une instance négative, alors le théorème de Hall, il existe un sous-ensemble de sommets $A \subseteq U$ tel que $|N(A)| \leq |A|$ avec $N(A)$ l'ensemble des voisins de U.

Fin correction exercice 61

2.7 Classe $\mathcal{NP}$-complétude

2.7.1 Autour de la Satisfaisabilité

Exercice 62 – Satisfaisabilité optimale

Nous considérons des problèmes de satisfaisabilité maximum o nous cherchons une affectation de valeurs 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 : Etant 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 ?

1. Montrer Satisfaisabilité $\propto$ Satisfaisabilité Maximale. En déduire une preuve pour k-Satisfaisabilité $\propto$ k-Satisfaisabilité Maximale.
2. Nous voulons prouver maintenant que 2-Satisfaisabilité Maximale est $\mathcal{NP}-$difficile.
   (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 $|C_i| = 3, \forall i$. 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 de 10 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.
3. La variante décisionnelle de la version générale de Satisfaisabilité Minimale est définie comme suit :

   Etant 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 défini 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 :

$$(\bar{a}_i \vee y_i), (\bar{b}_i \vee \bar{y}_i)$$

La formule ϕ' est obtenue par la conjonction de $2m$ 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 satisfaite dans ϕ' si et seulement si au moins K_{max} clauses sont satisfaites dans ϕ..

Correction exercice 62

1. Il est dans $\mathcal{NP}$. Il est aussi $\mathcal{NP}$-difficile car s'il était polynomial, il suffirait de la résoudre sur une instance ϕ de Satisfaisabilité à m clauses pou décider en temps polynomial, si les m clauses de ϕ sont simultanément satisfiables. (si la valeur de 2-Satisfaisabilité Maximale sur ϕ vaut m, alors elle est satisfiable ; si cette valeur est inférieure à m, alors ϕ n'est pas satisfiable.
2. Pour 2-Satisfaisabilité Maximale il dire en premier que ϕ' est obtenue en temps polynomial $\max\{n, m\}$.

   La formule ϕ' est obtenue par la conjonction fr $10m$ clauses ainsi construites. Le passage de ϕ en ϕ' se fait en temps polynomial en $\max\{m, n\}$. Enfin, nous posons $K = 7m$.

   Démontrons maintenant que ϕ est satisfaite ssi, dans ϕ', au moins $7m$ sont satisfiables. En effet,

   — si C_i n'est pas satisfaite, aucun littéral ne vaut 1, et alors au maximum 6 clauses sont satisfaites dans $\phi', (y_i = 0)$.
   — Si C_i est satisfaite et un littéral vaut 1, alors au maximum 7 clauses sont satisfaites dans $\phi', (y_i = 0)$.
   — Si C_i est satisfaite et deux littéraux vaut 1, alors au maximum 7 clauses sont satisfaites dans $\phi', (y_i = 0)$ ou $y_i = 1$.
   — Si C_i est satisfaite et trois littéraux vaut 1, alors au maximum 7 clauses sont satisfaites dans $\phi', (y_i = 1)$.

   Par conséquent, si ϕ est satisfaite (c'est à dire, m clauses sont satisfaites), alors $7m$ clauses de ϕ' sont satisfaites, sinon au plus $m - 1$ clauses (il y a une au moins un clause non satisfaites) de ϕ sont satisfaites et donc au plus $7m - 1$ clauses $(7m - 7 + 6)$ de ϕ' sont satisfiables.
3. Pour 2-Satisfaisabilité Minimale il dire en premier que ϕ' est obtenue en temps polynomial $\max\{n, m\}$.

   Considérons une instance de 2-Satisfaisabilité Maximale, notée $(\phi, K_{\max}$, où ϕ définie sur les variables $x_1, x_2, \ldots, x_n$ et les clauses $C_1, \ldots, C_m, i = 1, \ldots, m$. On lui associe la formule ϕ'

définie sur les variables $x_1, x_2, \ldots, x_n, y_1, , \ldots, y_m$ (on ajoute m nouvelles variables) et sur un ensemble de $2m$ clauses $C'_1, \ldots, C'_{2m}$ construites comme suit : à chaque clause $c_i = (l_1 \vee l_2)$, nous associons l'ensemble E_i de deux nouvelles clauses suivantes : $(\bar{l}_1 \vee y_i)$ et $(\bar{l}_2 \vee \bar{y}_i)$. La formule ϕ' est obtenue par la conjonction des $2m$ clauses ainsi construites. Le passage de ϕ en ϕ' se fait en temps polynomial en $\max\{m, n\}$. Enfin nous posons $K_{\min} = 2m - K_{\max}$

Fin correction exercice 62

Exercice 63 – Autour de Satisfaisabilité

Non égal Satisfaisabilité (NAESAT)

Entrée : Etant donnée une formule conjonctive ϕ sur n variables et m clauses

Question : Existe-t-'il une affectation de valeurs de vérité aux variables qui satisfasse ϕ tel que chaque clause à une littéral à vrai et un à faux ?

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

Correction exercice 63

On ajoute une nouvelle variable z, et à chaque clause on ajoute z.

Ainsi si j'ai une affectation consistante des variables de vérité pour le problème Satisfaisabilité alors on obtient une affectation valide pour le problème Non égal Satisfaisabilité en mettant z à faux.

Réciproquement, si on a une affectation consistante pour Non égal Satisfaisabilité ; si z est à faux alors on a une affectation consistante pour Satisfaisabilité. Si $z = vrai$ alors on inverse les toutes les valeurs pour se ramener à $z = faux$

Fin correction exercice 63

Exercice 64 – Autour de Satisfaisabilité (suite)

Coupe maximum (CUT)

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. Conclure.

Remarque : vous verrez en master que coupure min est polynomiale même si les arêtes ont des poids.

Correction exercice 64

On construit à partir de formule F un graphe G pondéré et un entier m tel que

F est NON ÉGAL 3-SATISFAISABILITÉ $\leftarrow (G, \omega)$ admet une coupe de taille au moins m.

Soit t clauses $F = C_1 \wedge C_2 \wedge \ldots \wedge C_t$. Construisons une pondération $(G, \omega, 5t)$ pour le problème COUPE MAXIMUM ssi F est satisfiable.

On suppose sans perte de généralité qua dans l'instance de NON ÉGAL 3-SATISFAISABILITÉ :

— une clause n'a pas une variable et son complémentaire (sinon on peut supprimer la clause).
— deux clauses ont au plus un littéral en commun.

La construction :

— G possède $2n$ sommets v et $\bar{v}$ pour chaque variable v. Le poids sur cette arête de poids n_i avec n_i la somme des occurrences de x_i et $\bar{x}_i$.
— Pour chaque clause $(\lambda_1, \lambda_2, \lambda_3)$, on crée un triangle (si une arête existe alors, le poids augmente d'une unité).
— On a $3m + n$ arêtes. Les arêtes $\{v, \bar{v}\}$ et les arêtes $\{l_1, l_2\}$ si les littéraux (l_1, l_2) appartiennent à la même clause (on forme ainsi des triangles)
— Remarquons que la somme de $\sum_i n_i$ est la somme des occurrences des littéraux dans F est égale à $3t$.

Le problème COUPE MAXIMUM $\in \mathcal{NP}$.

— On remarque que le nombre d'arêtes dans une coupure correspondant à une affectation de NON ÉGAL 3-SATISFAISABILITÉ est $(2m + n)$ (tous les littéraux à vrai sont séparés des littéraux à faux). Une partition avec des valeurs positives et l'autre avec des valeurs négatives.
— Réciproquement une coupure avec $(2m + n)$ arêtes coupe forcément chaque triangle en deux arêtes (on en coupe forcément zéro ou deux), et chaque arête du couplage $\{v, \bar{v}\}$ .On en déduit donc une solution pour NON ÉGAL 3-SATISFAISABILITÉ .

  Plus précisément, on suppose que G possède une coupe (A, B) de valeur au moins $5t$.

  1. Assumons le fait que les variables positives et négatives sont dans deux partitions différentes. En effet si les littéraux x_i et $\bar{x}_i$ appartiennent à la même coupe cela contribue ensemble à au plus de n_i dans la coupe (chaque apparence contribue à une unité de valeur dans la coupe).

     Donc on augmente la valeur de la coupe si on met par exemple $x_i \in A$ et $\bar{x}_i \in B$.
  2. Les variables positives et négatives contribuent à hauteur de "$t = \sum_i n_i$ à la coupe. Le reste $\geq 2t$ viennent des triangles.

     Un triangle contribue soit 0 ou 2 à la coupe. 0 si toutes les sommets sont des la même partition (resp. 2 si coupe).

     $\exists t$ triangle, on le coupe en deux de valeur est au moins de $2t$.
  3. Sélectionner les affectations qui satisfasse tous les littéraux de A. Les affectations sont consistantes (x_i et $\bar{x}_i$) ne sont pas dans la même partition. Sachant que nous coupons les triangles alors i Ilya au moins un positif et un négatif.

En conclusion, l'instance de NON ÉGAL 3-SATISFAISABILITÉ revient à résoudre à COUPE MAXIMUM avec G et $k = 2m + n$.

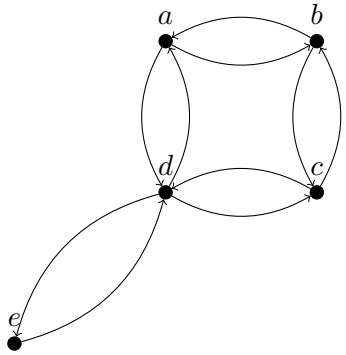


FIGURE 8 – Exemple pour le FEEDBACK VERTEX SET .

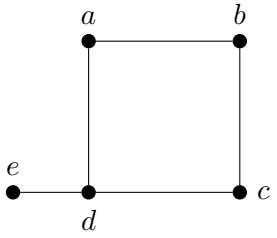


FIGURE 9 – Instance initiale pour le RECOUVREMENT DE SOMMETS.

Fin correction exercice 64

2.7.2 Problèmes autour des graphes

Exercice 65 – Problème du FEEDBACK VERTEX SET

Le problème FEEDBACK VERTEX SET défini ci-dessous est un problème fondamentale en base de données. Dans les systèmes d'exploitation, ce problème joue un rôle important dans l'étude de la récupération de données après des crashs/blocages. Dans le graphe des processus d'un système d'exploitation, chaque cycle dirigé correspond à une situation d'impasse. Afin de résoudre toutes les impasses, certains processus bloqués doivent être abandonnés. Un FEEDBACK VERTEX SET dans le graphe correspond à un nombre minimum de processus qu'il faut abandonner.

FEEDBACK VERTEX SET (FVS)

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

Question : Existe-t'il un ensemble $S \subset V$ de k sommets tel que $G \backslash S$ est un graphe sans circuit (tous les arcs sont orientés dans le même sens) ?

1. Donner l'ensemble S pour le graphe orienté donné par la figure 8.
2. Proposer une réduction à partir de RECOUVREMENT DE SOMMETS.
3. Appliquer votre réduction sur le graphe donné par la figure 9.

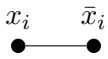


FIGURE 10 – Graphe T_i.

Correction exercice 65

1. $S = \{d, b\}$.
2. Soit G un graphe non orienté. On double les arêtes pour former deux arcs. Soit G' ce nouveau graphe. Il existe un RECOUVREMENT DE SOMMETS de taille k dans G ssi il existe un FEEDBACK VERTEX SET de même taille dans G'. C'est le même ensemble.

   FEEDBACK VERTEX SET $\in NP$.

   — Tout d'abord, supposez qu'il y a une couverture de taille k dans G. Maintenant, retirez ces sommets k de G avec les arêtes incidentes. Remarquez que pour tout arc $(u, v) \in G'$, au moins un des deux sommets u ou v doit avoir été enlevé, car l'un de u ou v doit être dans notre couverture. Ainsi, après avoir enlevé ces sommets k et leurs arcs incidents de G', aucune paire de sommets n'es relié. Il ne peut y avoir aucun cycle.

   — Inversement, supposons qu'il existe un ensemble S de sommets de taille k de G' dont la suppression casse tous les cycles. Par construction, chaque paire de sommets u, et v relié dans G admet un cycle $(u, v), (v, u)$ dans G' . Étant donné que la suppression de l'ensemble S a interrompu tous les cycles de G', pour chaque arête $\{u, v\} \in G$, l'ensemble S doit contenir u ou v (ou les deux). Ainsi, S est une couverture de sommet de G.

Fin correction exercice 65

Exercice 66 – Problème de la coloration

3-COLORATION

Entrée : Soit G un graphe, $k \in \mathbb{N}$
Question : Existe-t'il une $k = 3$-coloration valide pour G ?

1. Montrer que 3-COLORATION est $\mathcal{NP}$-complet (réduction à partir de 3-SATISFAISABILITÉ).

   Nous proposons la transformation polynomiale suivante à partir de 3-SATISFAISABILITÉ : soit une instance $I \in$ 3-SATISFAISABILITÉ constitué d'un ensemble $B = \{C_1, \ldots, C_m\}$ de m clauses sur un ensemble X de n variables $\{x_1, \ldots, x_n\}$. Nous allons construire, à partir de la formule B, un graphe $G = (S, A)$ de manière que B soit satisfiable si et seulement si G admet une coloration en trois couleurs.

   — $\forall x_i \in X$, on pose $T_i = (S_{i,1}, A_{i,1})$ où $S_{i,1} = \{x_i, \bar{x}_i\}$ et $A_{i,1} = \{x_i\bar{x}_i\}$ (voir figure 10).
   — $\forall C_j \in B$, on pose $V_j = (S_{j,2}, A_{j,2})$ où $S_{j,2} = \{y_{j,1}, y_{j,2}, y_{j,3}, y_{j,4}, y_{j,5}, y_{j,6}\}$ et $A_{j,2} = \{y_{j,1}y_{j,2}, y_{j,1}y_{j,4}, y_{j,2}y_{j,4}, y_{j,4}y_{j,5}, y_{j,5}y_{j,6}, y_{j,5}y_{j,3}, y_{j,3}y_{j,6}\}$ (voir figure 11).
   — V_j est isomorphe à $G_0 - \{u_0, u_1, u_2\}$.

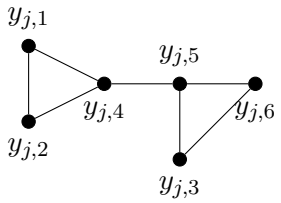


FIGURE 11 – Graphe V_j.

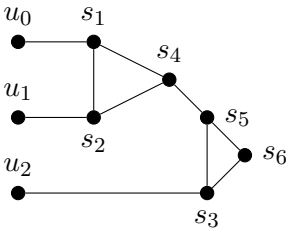


FIGURE 12 – Graphe G_0.

— On pose $T = (S_3, A_3)$ avec $S_3 = \{v_1v_2\}$ et $A_3 = \{v_1v_2\}$; pour chaque variable x_i de X, $A_{i,4} = \{v_2x_i, v_2\bar{x}_i\}, \forall 1 \leq i \leq m, A_{j,5} = \{v_1y_{j,6}, v_2y_{j,6}\}$; enfin $A_{j,6}$ est un ensemble de trois arêtes reliant, pour chaque clause C_j, les trois sommets de type x_i ou $\bar{x}_i$ correspondant à la première, la deuxième et la troisième littéraux de C_j.

Le graphe G_0 est donné par la figure 12.

Donner le nombre de sommets et le nombre d'arêtes. Appliquer la construction sur l'instance $X = \{x_1, x_2, x_3, x_4, x_5\}$, $B = \{(x_1 \vee \bar{x}_3 \vee \bar{x}_4), (\bar{x}_4 \vee x_2 \vee \bar{x}_1)\}$.

2. Montrer que le problème 4-COLORATION est $\mathcal{NP}$-complet (réduction à partir de 3-COLORATION).

Correction exercice 66

1. Le nombre de sommets est de $2|X| + 6m + 2$ donné par $(\bigcup_{x_i \in X} S_{i,1}) \cup (\bigcup_{1 \leq j \leq m} S_{j,2}) \cup S_3$ et $3|X| + 12m + 1$ arêtes données par $A = (\bigcup_{x_i \in X}(A_{i,1} \cup A_{i,4})) \cup (\bigcup_{1 \leq j \leq m}(A_{j,2} \cup A_{j,5} \cup A_{j,6})) \cup A_3$

   Cette transformation est clairement faite en temps polynomial.

   Pour l'exemple (voir figure 13).

   — Remarque 1 : Une coloration des quatre sommets u_0, u_1, u_2 et s_6 en trois couleurs (ou moins) telle que s_6 reçoive la même couleur que l'un (ou plus) des trois autres sommets peut toujours être complétée en une 3-COLORATION.

   — Remarque 2 : Dans une 3-COLORATION de G_0 où u_0u_1, u_2 reçoivent la même couleur, nécessairement s_6 a aussi cette couleur.

   — En effet, si 3-SATISFAISABILITÉ admet une solution, en appelant f la coloration recherchée des sommets de G en trois couleurs 1, 2, et 3, on commence par colorier par 1 (resp.

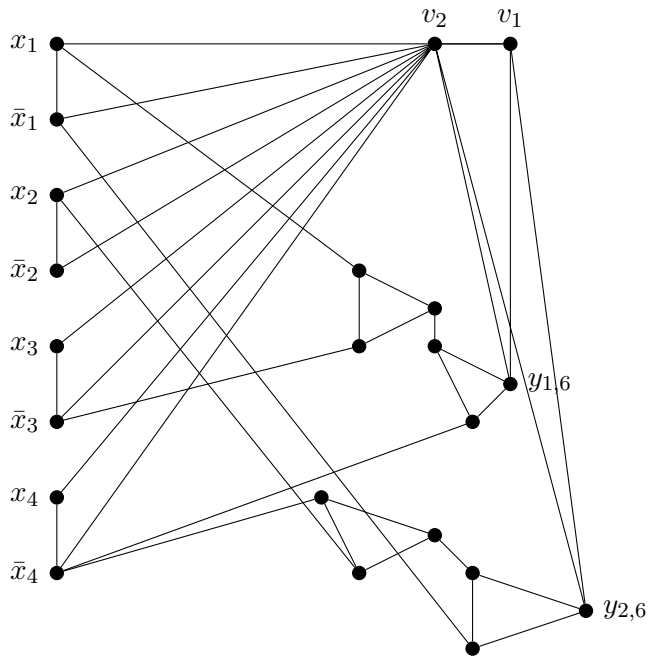


FIGURE 13 – Illustration de la construction pour l'instance $X = \{x_1, x_2, x_3, x_4, x_5\}$, $B = \{(x_1 \vee \bar{x}_3 \vee \bar{x}_4), (\bar{x}_4 \vee x_2 \vee \bar{x}_1)\}$ pour le problème 3-COLORATION.

2) les sommets de type x_i ou $\bar{x}_i$ correspondant aux littéraux qui ont mises à vrai (resp. à faux) pour satisfaire B :

— si $x_i = vrai$, $f(x_i) = 1$ et $f(\bar{x}_i) = 2$;
— si $x_i = faux$, $f(x_i) = 2$ et $f(\bar{x}_i) = 1$;
— Posons ensuite $f(v_1) = 2, f(v_2) = 3$ et $f(y_{j,6}) = 1, 1 \leq j \leq m$.

Jusqu'ici aucune paire e sommets adjacents n'a été coloriée avec la même couleur. La remarque 1 Montre alors qu'on peu compléter f sur les composantes V_j de G pour avoir une 3-COLORATION de G, puisqu'au moins un des trois sommets de type x_i ou $\bar{x}_i$ reliés à une composante V_j a la même couleur 1 que $y_{j,6}$.

— Réciproquement, si f est une 3-COLORATION de G, avec par exemple (san perte de généralité) $f(v_2) = 3, f(v_1) = 2$, alors $\forall x_i, f(x_i) = 1$ ou 2, $f(\bar{x}_i) = 1$ ou 2, avec $f(x_I) \neq f(\bar{x}_i)$, donc, en posant $x_i = vrai$ si et seulement si $f(x_i) = 1$, on obtient une assignation valide des variables de X. Par ailleurs, les sommets $y_{6,j}$ qi sont reliés à v_1 et v_2, ont tous la couleur 1 : $f(y_{j,6}) = 1; 1 \leq j \leq m$.

Supposons alors qu'une clause C_u de B ne soit pas satisfaite : les trois sommets de type x_i ou $\bar{x}_i$ qui sont reliés au graphe V_u ont pour couleur 2. Alors, d'après la remarque préliminaire (remarque 2), $f(y_{6,j}) = 2$, ce qui est une contradiction.

2. Nous procédons à la réduction simple : tout graphe G, instance quelconque de 3-COLORATION est transformé en un graphe G' constitué du graphe G complété par un sommet supplémentaire, relié à tous les sommets de G. Il est immédiat de vérifier que G est 3-COLORATION si et

seulement si G' est 4-COLORATION.

Fin correction exercice 66

Exercice 67 – 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-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 t'il si on autorise l'inégalité triangulaire $\forall i, j, k, c_{ik} \leq c_{ij} + c_{jk}$?

Correction exercice 67

Nous construisons un graphe complet orienté $G' = (V, E')$ avec $c_{ij} = 1$ si $(i, j) \in E$ $c_{ij} = 2$ sinon. Nous définissons $k^* = n$. Avec cette définition le tour $\sum_{(i,j)\in W} c_{ij} \leq k^*$ est satisfaite si et seulement si W appartient à E et ainsi W est cycle Hamiltonien sur le graphe G. Par conséquent, le voyageur de commerce possède une réponse positive si et seulement si G contient un tour de longueur n, et ceci apparaît si G possède un cycle Hamiltonien.

La preuve marche également avec l'inégalité triangulaire par construction.

Fin correction exercice 67

Exercice 68 – 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 .

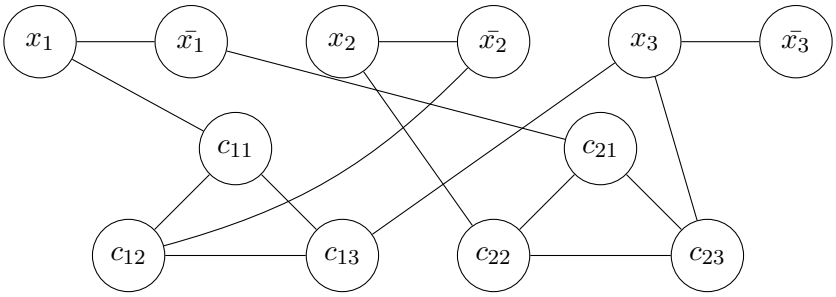
Aide pour la transformation polynomiale : Considérons les variables $x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n$ et n arêtes $(x_i, \bar{x}_i), \forall i = 1, \ldots, n$. Nous considérons m triangles constitués des littéraux. Pour une clause C_i nous notons $c_{i_1}, c_{i_2}, c_{i_3}$ et nous relions le sommet x_i à un sommet d'un triangle noté $C_{j_k}, k = 1, 2, 3$ si la variable x_i apparaît dans la clause C_j à la position k. littéraux

Correction exercice 68

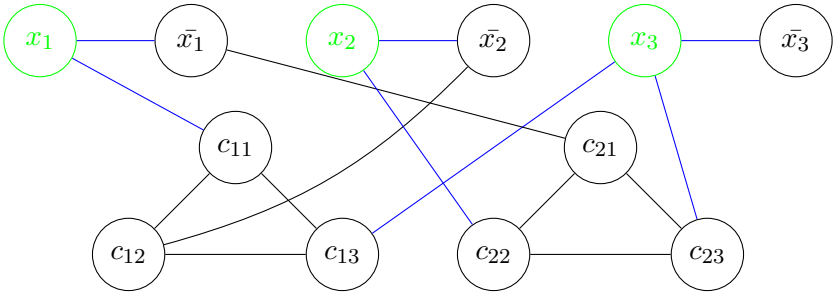
La transformation polynomiale est donnée dans l'énoncé, illustrons-la avec la formule suivante :

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

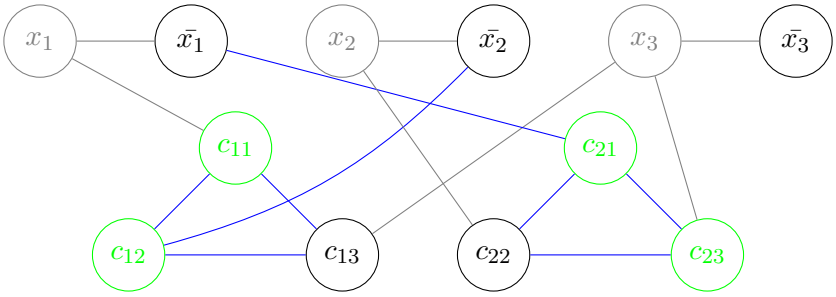
Cette formule est clairement satisfiable, il suffit que la valeur de vérité de x_3 soit à $\top$. Le graphe construit grâce à la transformation polynomiale est le suivant :



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.

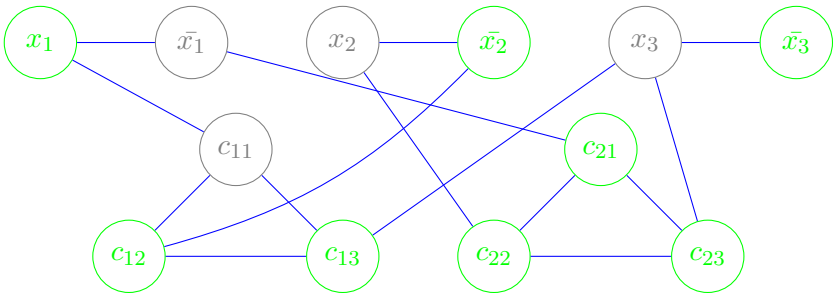
— $\alpha(l_i^{k_1}) = \alpha(l_i^{k_2}) = \top$ et $\alpha(l_i^{k_3}) = \bot$ avec $k_1 \neq k_2 \neq k_3$. Deux des trois littéraux sont à vrai, dans ce cas, on prend dans notre couverture de sommets les littéraux valués à $\top$: $l_i^{k_1}$ et $l_i^{k_2}$, ainsi que $\overline{l_i^{k_3}}$. De plus, 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.

⇐ 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.

Fin correction exercice 68

Exercice 69 – Recouvrement de sommets (suite)

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

Correction exercice 69

Instance I1 de départ : (G, k) de Recouvrement de sommets.

Instance I2 créée : On peut déjà remarquer qu'il y a un nombre pair de sommets de degré impair dans G car $\sum deg(v) = 2|E|$.

On construit une instance $(G_2, k+2)$ en créant trois nouveaux sommets x, y et z. Ensuite on relie x à chaque sommet de G de degré impair, ainsi qu'à y et z, et on relie y et z (on a ainsi formé un triangle xyz) Equivalence : S'il existe une couverture C dans V de cardinal plus petit que k pour G, alors il existe une couverture de cardinal plus petit que $k+2$ (par exemple $C \cup \{x, y\}$ pour G_2. Réciproquement, s'il existe une couverture C_2 de cardinal plus petit que $k+2$ pour G_2, alors C_2 contient forcément au moins deux sommets dans $\{x, y, z\}$ car c'est un triangle qui n'est relié au reste du graphe que grâce à x. Donc $C = C_2 \backslash \{x, y, z\}$ est une couverture de G de cardinal plus petit que k.

Fin correction exercice 69

Exercice 70 – Plus long Chemin

Montrer que le problème Plus long Chemin (défini ci-après) reste $\mathcal{NP}$-complet même si tous les sommets sont de degrés pairs.

Plus long Chemin (Plus Long Chemin)

Entrée : Un graphe orienté $G = (V, E)$, $L \in \mathbb{N}$
Question : Existe t'il un chemin simple (sans boucle) de s à t avec au moins L arcs ?

Correction exercice 70

Si $L = n - 1$, un chemin de de longueur L du sommet s à t est un chemin Hamiltonien. Ce problème est $\mathcal{NP}-$complet..

Fin correction exercice 70

Exercice 71 – Réductions autour de Hamiltonisme

Montrer que les problèmes sont tous 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

Correction exercice 71

1. Chaîne Hamiltonienne $\propto$ Cycle Hamiltonien. Il suffit de rajouter un sommet x à G et de relier x à tous les sommets (de manière orienté ou pas).
2. Cycle Hamiltonien $\propto$ Chaîne Hamiltonienne. A toute arête de G on construit le graphe G_e qui consiste à supprimer $e = \{x, y\}$ et à ajouter deux sommets x_1 et y_1 relié respectivement à x et y. G possède un Cycle Hamiltonien ssi l'un des G_e possède une Chaîne Hamiltonienne.
3. Chaîne Hamiltonienne (resp. Cycle Hamiltonien) dans les bipartis se réduit à Chaîne Hamiltonienne (resp. Cycle Hamiltonien) dans les graphes non orienté (les graphes bipartis sont des cas particulier)
4. Chaine (resp. ccle) dans les graphes non-orienté se réduit à chaîne (resp. cycle) dans les bipartis. Soit $[x; y]$ une arête, elle est transformée en $4*2$ sommets d_x, x, x', f_x (resp. d_y, y, y', f_y relié en chemin et les sommets d_x et f_y (resp. f_x et d_y) sont reliés.
5. Cycle Hamiltonien non orienté réduit à Cycle Hamiltonien orienté : à chaque arête $[x, y]$ on crée deux arcs (x, y) et (y, x).
6. Cycle Hamiltonien orienté vers Cycle Hamiltonien non-orienté. A chaque arc (x, y), on crée un chemin de longeur trois x_d, x, x_f (resp. y_d, y, y_f) et on ajoute l'arête (x_f, y_d).
7. Circuit Hamiltonien (resp. Chemin Hamiltonien) orienté se réduit à Cycle Hamiltonien orienté. Soit $G = (V, E)$ on transforme en $G' = (V \cup \{x\}, E \cup \{(x, t)|\forall t \in V\})$. G' admet un cycle Hamilotnien ssi G possède une chaîne Hamiltonien
8. Chaîne Hamiltonienne non-orienté se réduit à Chaîne Hamiltonienne orienté. Pour chaque $[x, y]$ on crée deux arcs (x, y) et (y, x).

Fin correction exercice 71

2.7.3 Autour des nombres

Exercice 72 – Autour du problème de la 2-Partition

Nous rappelons que le problème suivant est $\mathcal{NP}$-complet.

2-Partition (Partition)

Entrée : Etant 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.

1. 2-Partition à valeurs paires (Partition à valeurs paires)

   Entrée : Etant donné 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 total P ?
2. 2-Partition avec nombre paire (Partition avec nombre pair)

   Entrée : Etant donné $2n$ 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 de même poids total P ?
3. 2-Partition équilibré (Partition équilibré)

   Entrée : Etant donné $2n$ 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 et $\bar{I}$ de même poids total P tel que $|I| = n$?
4. 2-Partition (Partition impair/paire)

   Entrée : Etant donné $2n$ 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 et $\bar{I}$ tel que $\sum_{i \in I} a_i = \sum_{i \in \bar{I}} a_i$, avec un entier entre a_{2j-1} et a_{2j} appartenant à I ?

Correction exercice 72

1. Il suffit de poser $p'(a_i) = 2 * p(a_i)$.
2. Soit I_1 une instance de 2-Partition avec les objets $a_1, \ldots, a_n$. Si n est paire, on pose $I_1 = I_2$. Sinon soit $I_2 = I_1 \cup \{2P, 2P, 4P\}$ (afin de garder la parité). On pose $P' = 5P$
3. Soit I_1 une instance de 2-Partition avec les objets $a_1, \ldots, a_n$. A partir de I_1, on crée une instance I_2 incluant les objets $a_1 + 1, \ldots, a_n + 1$ et n objets de taille un.

   Si I_1 admet une solution I, alors on obtient une partition de I_2 de valeurs $P + n$ et de taille n en prenant les éléments $\{a_i + 1\}_{i \in I}$ et $n - |I|$ éléments de taille un, avec $\sum_{i=1}^{n} a_i = P$.

   Réciproquement, toute solution I' de I_2 tel que $\sum_{i \in I'}(a_i + 1) + n - |I'| = P/2 + n$, alors $|I'|$ doit être égal à n. Ainsi, $\sum_{i \in I'} a_i = P/2$ et I' défini une solution de I_1.
4. Soit I_1 une instance de 2-Partition équilibré incluant $2n$ entiers $a_1, \ldots, a_{2n}$. On construit une instance I_2 de 2-Partition :$I_2 = (a_1, B, a_2, B, \ldots, a_{2n}, B)$ avec $B > \sum_{i=1}^{2n} a_i$. On pose $P' = P + nB$

L'équivalence est évidente sachant que dans n'importe quelle solution de I_2, il y a exactement n éléments égaux à B dans chaque partition. Il est important de prendre B très grand, ainsi si on ne met pas n objets de taille B dans une partition, alors l'autre côté tu as $(n+1)B > P'$.

Fin correction exercice 72

Exercice 73 – Tri-partition

3-Partition (3-Partition)

Entrée : Etant 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 ?

Montrer que 3-Partition est $\mathcal{NP}$-complet ?

Correction exercice 73

Given an instance of Equal S. Create an instance of TriPartition, S' where $S' = S \cup \{h\}$ where $h = \sum_{s \in S} s/2$.

— If S can be divided into two equal sets then S' can be divided into 3 equal sets (the same 2 plus the single item h).
— If S' can be divided into three equal sets then one of them must consist of only h. The other two sets are a partition of S into two equally sized sets.

Fin correction exercice 73

Exercice 74 – Programmation dynamique : algorithme pseudo-polynomial

1. Sur le problème de la partition :

   2-Partition (Partition)

   Entrée : Etant 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 ?

   (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

3. Nous considérons le problème du sac à dos avec répétition. On note $K[w]$ la valeur maximale pouvant être stockée dans un sac de capacité $w \in \{0, 1, 2, \ldots, W\}$. Remarquons que si une solution optimale pour réaliser $K[w]$ contient au moins un objet i, alors

   — sans cette occurrence de l'objet i, le contenu du sac est optimal pour le poids $w - w_i$, et
   — sa valeur est alors $K[w - w_i]$.

   (a) On pose $K[w] = \max(\{0\} \cup \{K[w - w_i] + v_i | w_i \leq w\})$. Montrer que $K[w]$ est la valeur maximale d'un sac de capacité $w \in \{0, 1, \ldots, W\}$. La preuve se fait par récurrence.
   (b) Donner l'algorithme. Donner la complexité temporelle et en mémoire.
   (c) Appliquer l'algorithme sur l'instance : $(w_1, v_1) = (2, 6), (w_2, v_2) = (1, 1), (w_3, v_3) = (5, 18), (w_4, v_4) = (6, 22), (w_5, v_5) = (7, 24)$, et $W = 12$.

4. Sur l'algorithme de Held-Karp pour le problème du Voyageur de Commerce.

   (a) Soit une instance du problème Voyageur de Commerce, c'est-à-dire la donnée d'une matrice $n \times n$ des poids P d'un graphe $G = (X, E)$ à n sommets, supposés numérotés de 0 à $n-1$. Pour toute partie S de X contenant le sommet 0, et tout sommet i non dans cette partie, on considère le problème suivant : déterminer une plus courte chaîne du sommet 0 au sommet i passant une fois et une seule fois par tout sommet de S et n'utilisant pas de sommet non dans S en dehors de i : appelons $C(S, i)$ la longueur d'une telle chaîne. Si S ne contient que le sommet 0, on voit qu'on a, pour tout $i \neq 0$; $C(S, i) = P(0, i)$ ce qui nous sera utile pour initialiser la récurrence. Sinon, on a

$$C(S, i) = \min_{k \in S - \{0\}} \{C(S - \{k\}, k) + P(k, i)\} \text{ pour } i \notin S$$

   On a donc établi une relation de récurrence sur la taille de S liant les $C(S, i)$. Quant à notre problème lui-même, il suffit pour le résoudre de déterminer la valeur de $\min_{i \in X - \{0\}} \{C(X - \{i\}, i) + P(i, 0)\}$. Nous avons bien ici appliqué les principes de la méthode, en plongeant le problème dans une classe plus générale que nous résolvons en utilisant la relation de récurrence.

(b) Résoudre le problème Voyageur de Commerce avec la matrice des distances suivantes :

$$C = \begin{pmatrix} 0 & 2 & 9 & 10 \\ 1 & 0 & 6 & 4 \\ 15 & 7 & 0 & 8 \\ 6 & 3 & 12 & 0 \end{pmatrix}$$

(c) Calculer la complexité.

Correction exercice 74

1. Le problème de la partition :

   (a) Il faut que la somme soit paire

   (b) Le plongement du problème :

      i. Soient $i, j \in \mathbb{N}$ tels que $1 \leq i \leq n$ et $0 \leq j \leq P$, et $T(i,j) \in \{0,1\}$. On considère le problème : « étant données les i premiers éléments de la famille, il existe un sous-ensemble de ces i éléments de poids j » . On a la relation de récurrence suivante entre deux lignes du tableau :

      — Si $i = 1$

      $$T(1,j) = \begin{cases} 1 \text{ si } j = \omega(a_1) \\ 0 \text{ sinon} \end{cases}$$

      — Si $i > 1$

      $$\begin{cases} T(i,j) = T(i-1,j) \\ T(i,j+\omega(a_i)) = T(i-1,j) \text{ si } j+\omega(a_i) \leq P \end{cases}$$

      ii. On obtient

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0
3	1	0	0	1	0	1	0	0	1	1	0	0	1	0	1	0	0
4	1	0	0	1	0	1	0	0	1	1	0	1	1	1	1	0	1
5	1	0	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1
6	1	0	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1

      iii. Une fois le tableau calculé, il faut reconstruire la solution à partir de celui-ci. Pour cela, on applique l'algorithme 12

   (c) La largeur du tableau est P, le remplissage du tableau à une complexité de $O(nP)$. Or en posant w_{max} le poids le plus grand, on a $P = \frac{\sum_{i=1}^{n} \omega(a_i)}{2} \leq \frac{\sum_{i=1}^{n} w_{max}}{2} = nw_{max}/2$. LA reconstruction de la solution se fait en $O(n)$ étapes. $O(n^2 w_{max})$.

2. Le problème du sac à dos :

Algorithm 12 Reconstruction de la solution

$i := n$; $j = P$, $S = \varnothing$;
while $i > 0$ et $T[i][j] \neq 0$ **do**
  $i - -$
  $S := S \cup \{a_{i+1}\}$;
  $j := j - \omega(a_{i+1})$;
  $i := n$
end while
Retourner S

(a) $K[j, w] = \max(K[j-1, w], K[j-1, w-w_j] + v_j)$ et $K[0, w] = 0$ et $K[j, 0] = 0$, et la solution est donnée par la valeur $K[n, W]$ (tous les objets sont autorisés et le poids maximum est W

(b) La solution est donnée par la tableau

$j \backslash w$	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	25	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) La complexité en temps est en $O(n.W)$ et en mémoire $O(n.W)$.

(d) Pour la reconstruction de la solution on applique le même principe que pour le problème de la partition.

3. Nous considérons le problème du sac à dos avec répétition,

(a) Par induction sur w. Si $K[w] = 0$: alors il n'existe pas d'objet de poids $\leq w$, donc la valeur maximale d'un sac de capacité w est 0.

Si $K[w] > 0$: prenons un sac S optimal pour w. Supposons que ce sac contienne les objets $L = \{i_1, i_2, \ldots, i_k\}$. Soit i un de ces i.

La valeur de S est donc : $v_i + \sum_{j \in L \setminus \{i\}} v_j$.

Et le poids de S sans l'objet i, est $\sum_{j \in L \setminus \{i\}} w_j$ et sa valeur est $\sum_{j \in L \setminus \{i\}} v_j$. On a bien $K[w - w_i] = \sum_{j \in L \setminus \{i\}} v_j$ car sinon S ne serait pas optimal.

La complexité en temps $O(Wn)$. Et en mémoire $O(W)$.

(c) La solution est donnée par le tableau 1

0	1	2	3	4	5	6	7	8	9	10	11	12
0	1	6	7	12	18	22	24	28	30	36	40	44

TABLE 1 – Solution donnée pour le problème du sac à dos sans répétition.

4. Programmation dynamique pour le VOYAGEUR DE COMMERCE.

(a) Voici le tableau :

Algorithm 13 Algorithme

(b)
Soit K un tableau de taille $W+1$.
$K[0]=0$.
for $w=1$ to W **do**
  $M=0$
  for $(w_i,v_i)\in T$ **do**
    if $w_i\leq w$ **then**
      $M=\max(M,K[w-w_i]+v_i)$
    end if
  end for
  $K[w]=M$
end for
Retourner $K[W],K$

i	$\{0\}$	$\{0,1\}$	$\{0,2\}$	$\{0,3\}$	$\{0,4\}$	$\{0,1,2\}$	$\{0,1,3\}$	$\{0,1,4\}$	$\{0,2,3\}$
1	1		5	6	0				6
2	2	4		3	1		8	2	
3	1	6	4		4	6		5	
4	0	1	3	5		5	6		4

i	$\{0,2,4\}$	$\{0,3,4\}$	$\{0,1,2,3\}$	$\{0,1,2,4\}$	$\{0,1,3,4\}$	$\{0,2,3,4\}$
1	3	5				4
2		6			7	
3	3			4		
4			6			

(b) La recherche du minimum me coute $l-1$. Le coût des éléments $i\notin S$ me coute $n-l$. De plus, le coût pour la calcul est C_{n-1}^{l-1}. L'intervalle pour l ezst 2 à $n-1$. Ainsi nous avons $\sum_{l=2}^{n-1}(l-1)(n-l)C_{n-1}^{l-1}$. Donc $0(n^2 2^n)$.

Fin correction exercice 74

Exercice 75 – Mètre du charpentier

Montrer que le problème du mètre de charpentier est un problème $\mathcal{NP}$-complet

Mètre du charpentier (MC)

Entrée : La longueur de l'étui L et des segments l_i (i de 1 à n).
Question : Peut-on plier le mètre pour qu'il rentre dans l'étui ?

Correction exercice 75

La réduction se fera à partir de 2-Partition.Les segments de chaque partition sont rangés dans le même sens. Donner un exemple pour expliquer la construction.

Prendre un étui de taille $B = \sum p(a_i)$ et un mètre dont le segment i est de longueur égal à $p(a_i)$, ajouter au début deux segments de taille B et $B/2$ et à la fin deux segments de taille $B/2$ et B.

Fin correction exercice 75

Exercice 76 – Algorithmes pseudo-polynomiaux

Donner deux algorithmes pseudo-polynomiaux pour résoudre la 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).

Correction exercice 76

1. $f(i, j)$ = poids le plus lourd pour un volume i avec les j premiers objets.
2. $g(i, j)$ = volume le plus petit pour le poids i avec les j premiers objets.

Fin correction exercice 76