Chapitre sept

Classes de complexité

MPI/MPI*, lycée Faidherbe

Résumé

Ce chapitre est le prolongement du précédent ; après avoir défini ce qu'est un problème qui admet une solution, on va se poser la question de la possibilité d'obtenir des résultats dans un temps raisonnable.

Ce qu'on va étudier n'est pas la complexité temporelle fine mais on va considérer 2 types de problèmes :

- ceux dont on sait qu'il existe une solution qu'on estime qu'elle est raisonnable
- ceux pour lesquels on ne connaît pas de solution estimée raisonnable mais qui forment un ensemble homogène en ce sens que toute bonne solution de l'un d'entre eux donnera une bonne solution pour tous les autres.

I Taille des données

Jusqu'à présent on a calculé la complexité comme fonction des données d'entrée sans aller trop loin dans la mesure de la taille de l'entrée.
Par exemple le problème de décision PREMIER

Instance : un entier n.

Question : n est-il premier ?

admet la solution suivante

```
let est_premier n
   let rec test p =
   if p*p > n
   then true
   else if n mod p = 0
      then false
      else test (p+1)
   in test 2
```

dont la complexité est un $\mathcal{O}(\sqrt{n})$.
Cependant, si on veut évaluer la complexité en fonction de la taille de l'entrée, on doit se poser la question du codage de l'entrée. Or un entier est mémorisé sous la forme de son codage en binaire. Un entier n est codé avec p_n bits si on a $2^{p_n-1} \leqslant n < 2^p_n$. Ainsi la taille de l'entrée et p_n avec $n \equiv 2^{p_n}$ donc la complexité est un $O\left(2^{p_n/2}\right)$: la complexité est devenue exponentielle en la taille de l'entrée.

Définition

Pour un problème, la *taille* de l'instance est l'espace mémoire nécessaire pour représenter l'instance.

Comme la taille intervient dans les calculs de complexité, on ne s'intéresse généralement qu'à un ordre de grandeur, ce qui signifie que l'unité n'a pas d'importance. Dans notre modèle, on peut compter les bits, les octets, ...
Un entier n pourra être considéré :

- de taille $\mathcal{O}(1)$ si on considère des entiers sur une représentation 64 bits (ou 128 bits),
- de taille $\mathcal{O}(\log n)$ si on considère des entiers sur une représentation de taille non bornée,
- même de taille $\mathcal{O}(n)$ si on considère des entiers sur une représentation unaire, un entier n est représenté par n bits 1. On n'utilise pratiquement jamais cette représentation.

Exemples

- Un entier n a une taille de $\log_2(n)$
- Une liste de n entier a une taille majorée $n \cdot \log_n(M)$ où M est un majorant des valeurs absolues des entiers dans la liste ; le plus souvent, on exprimera les entiers sur 64 bits (ou 128 bits), ce qui induit une taille en $\mathcal{O}(n)$.
- Un graphe $G = (S, A)$ a une taille en $\mathcal{O}\big(|S| + |A| \log_2(|S|)\big)$; le $\log_2(|S|)$ provient du fait que les éléments des listes dont les éléments appartiennent à un ensemble de $|S|$ éléments.

II Classe P

Définition

La complexité d'une solution d'un problème est une fonction $t(n)$ telle que la complexité de la solution pour une instance de taille n est majorée par $t(n)$.

On voudrait donner une notion de complexité raisonnable ; pour cela on va se poser la question de ce qui arrive lorsque la rapidité d'exécution évolue.
Si la vitesse d'exécution est multipliée par 10, pour le même temps de calcul la taille d'entrée

- peut passer de n à $10 \cdot n$ si $t(n) = \mathcal{O}(n)$,
- peut passer de n à $3 \cdot n$ si $t(n) = \mathcal{O}(n^2)$,
- peut passer de n à $1,1 \cdot n$ si $t(n) = \mathcal{O}(n^{10})$ (ce qui est peu courant),
- peut passer de n à $n + 3$ si $t(n) = \mathcal{O}(2^n)$,
- peut passer de n à $n + 6 \cdot \sqrt{n}$ si $t(n) = \mathcal{O}(2^{\sqrt{n}})$,

On choisit de considérer comme raisonnable une complexité logarithmique. Lorsque l'exposant de n est grand le gain peut sembler faible mais, il est toujours un multiplicateur et l'expérience montre qu'on peut faire des gains de complexité de $\mathcal{O}(n^p)$ à $\mathcal{O}(n^q)$ avec $q < p$.

Définition

Un problème de décision appartient à la classe P s'il admet une solution de complexité en $\mathcal{O}(n^p)$ pour un entier p.

Exemples

- Les opérations arithmétiques, addition, multiplication, division (entière), calcul du reste se font directement bit-à-bit : elles sont donc linéaires en la tailles des entiers. On en déduit que le problème PGCD, par exemple, appartient à P.

  Bien entendu, on peut trouver un algorithme de complexité exponentielle même pour l'addition de deux entiers, par exemple en effectuant n additions de 1 pour ajouter n à un entier.
- Les algorithmes simples de résolution de PREMIER sont exponentiels en la taille de l'entrée. Cependant, en 2002, Agrawal,Kayal et Saxena ont publié un algorithme linéaire qui résout le problème (algorithme AKS). Ainsi PREMIER $\in$ P.
- Les algorithmes vus en cours sur les graphe sont de complexité polynomiale et $n = |S| + |A|$ pour un graphe $G = (S, A)$ ils appartiennent à P.

III Classe NP

NP ne signifie pas non polynomial.
L'origine est Non-deterministic polynomial qui fait référence à une machine de Turing non-déterministe (comme un automate peut être non-déterministe).
On considère un problème de décision A et on note I_A l'ensemble des ses instances.

Définition

Un *vérificateur* pour A est un problème de décision B tel que $I_B = I_A \times C$ qui vérifie que, pour chaque instance positive x de A il existe un élément $c \in C$, le certificat de x tel que (x, c) est une instance positive de B.

Telle que définie ci-dessus, la notion est trop vague : on peut prendre $B = A$, considérer qu'on $I_A = I_A \times \{\emptyset\}$ et choisir $c = \emptyset$ pour tout x. Comme notre objectif est la complexité polynomiale on va rajouter cette condition aux deux endroits :

- dans la taille de c,
- dans la complexité de B.

Définition

Un problème de décision est dans la classe NP s'il admet un vérificateur appartenant à P et une fonction polynomiale f tels que, pour toute instance positive x de A, il existe d'un certificat c dont la taille est majorée par $f(n)$ où n est la taille de x.

On a une sorte de réduction à un problème dans P mais on ajoute l'apparition d'une donnée qui va miraculeusement permettre de résoudre le problème. La seule contrainte est cette donnée ne soit pas trop grosse.
Quand le problème de décision demande s'il existe un objet vérifiant une propriété $\mathcal{P}$, la recherche de certificat peut se simplifier. Le plus souvent, le certificat sera une solution possible. On doit donc, pour une instance x,

1. *recevoir* une solution c,
2. vérifier sa taille polynomiale,
3. vérifier en temps polynomial que c est une solution.

Le vérificateur, est ici

Verificateur

Instance : une instance x de A et un certificat c

Question : c est-il une solution de A pour l'instance x ?

On notera une dissymétrie : l'existence d'un certificat pour les instances positives ne dit rien au sujet des instances négatives.

Théorème 1

$\mathsf{P} \subset \mathsf{NP}$

Démonstration Si A est un problème dans P, il est son propre vérificateur : pour toute instance positive x, A appliqué à x permet de valider son acceptation en temps polynomial.
On peut choisir un certificat vide. . . . . ∎

On notera que la négation de A, notée coA admet aussi A comme vérificateur.

IV Exemples de problèmes NP

IV.1 SAT

Instance : une fonction booléenne φ.

Question : φ est-satisfiable ?

1. Une fonction booléenne peut être codée par un arbre dont les nœuds représentent les connecteurs logiques et les feuilles, les variables utilisées dans la fonction. Le nombre de variables utiles n est donc majoré par la taille de l'instance.
2. Un certificat peut être une valuation des variables utiles dont la taille est un $\mathcal{O}(n)$, polynomiale en la taille de l'instance.
3. La vérification se fait en calculant la valuation appliquée à φ : cela se fait en temps linéaire sur la taille de l'arbre donc de l'instance.

SAT $\in$ NP.

IV.2 2-PARTITION

Instance : une suite finie d'entiers, a_1, a_2, …, a_n.

Question : existe-t-i $I \subset \{1, 2, \dots, n\}$ tel que $\displaystyle\sum_{i\in I} a_i = \frac{1}{2}\sum_{i=1}^{n} a_i$?

1. Une instance est codée par un tableau de taille n d'entiers.
   Sa taille est un $\mathcal{O}(n)$ pour des entiers sur 64 bits, ou un $\mathcal{O}\big(n \cdot M\big)$ si les entiers sont de taille non fixée mais majorée par M pour l'instance.
2. Un certificat peut être un tableau de booléens de taille n majorée par la taille de l'instance.
3. La vérification se fait en calculant deux sommes : la complexité est un $\mathcal{O}(n^2)$ ou un $\mathcal{O}\big(n^2 \cdot M\big)$ selon le cas, c'est toujours polynomial en la taille de l'instance.

2-PARTITION $\in$ NP.

IV.3 HAMILTONIEN

Instance :un graphe non orienté $G = (S, A)$.

Question : G contient-il un cycle hamiltonien, c'est-à-dire un cycle passant une et une seule fois par chaque sommet ?

1. On considère que les sommets sont codés sur 64 bits.
   Une instance est codée par une matrice d'adjacence de taille $|S|^2$ ou par un tableau de liste d'adjacences de taille en $\mathcal{O}\big(|S| + |A|\big)$.
2. Un certificat peut être un tableau de $|S|$ sommets.
3. La vérification se fait en prouvant que les sommets sont 2-à-deux distincts, calcul en $\mathcal{O}\big(|S|^2\big)$ puis à vérifier que deux sommets consécutifs forment bien une arête calcul en $\mathcal{O}\big(|S| \cdot |A|\big)$. La complexité est bien polynomiale.

HAMILTONIEN $\in$ NP.

V NP-complétude

L'ensemble des instances d'un problème A est noté I_A

Définition

Un problème de décision A se *réduit en temps polynomial* au problème de décision B, et on note $A \leqslant_{\mathsf{P}} B$, s'il existe une fonction $f : I_A \mapsto I_B$, de complexité polynomiale, telle que x est une instance positive de A si et seulement si $f(x)$ est une instance positive de B.

C'est une réduction de A à B, $A \leqslant_{\mathsf{P}} B$ implique $A \leqslant B$, par transformation des instances (many-to-one) mais avec une contrainte de temps de calcul.
On notera que si $f(x)$ est calculé en temps T alors la taille de $f(x)$ ne peut dépasser T car une instrucction élémentaire ne peut créer/modifier qu'une unité d'espace. La taille de $f(x)$ est donc polynomiale en la taille de x pour une réduction polynomiale.

Théorème 2

Si A et B sont deux problèmes de décision tels que $A \leqslant_{\mathsf{P}} B$ alors

- si $B \in \mathsf{P}$, alors $A \in \mathsf{P}$,
- si $B \in \mathsf{NP}$, alors $A \in \mathsf{NP}$.

Démonstration Soit $f : I_A \mapsto I_B$, de complexité polynomiale, telle que x est une instance positive de A si et seulement si $f(x)$ est une instance positive de B.
On note T un polynôme tel que $\big|f(x)\big| \leqslant T\big(|x|\big)$.

- si $B \in \mathsf{P}$, on considère un algorithme polynomial Φ_B résolvant B.
  L'algorithme $\Phi_B \circ f$ résout A en temps polynomial en la taille de $f(x)$: $t \leqslant P\big(\big|f(x)\big|\big)$.
  On a donc $t \leqslant P \circ T\big(|x|\big)$: la complexité est encore polynomiale car la composée de deux polynômes est un polynôme.
- si $B \in \mathsf{NP}$ alors il existe, pour tout instance positive y de B, un certificat c_y de taille polynomiale en la taille de y.
  Pour tout instance positive de A, $c_{f(x)}$ est un certificat dont la taille comme ci-dessus, est polynomiale en la taille de x.

Définition

Un problème de décision A est *NP-complet* s'il appartient à NP et si tout problème de décision appartenant à NP se reduit polynomialment à A.

Leur intérêt provient du théorème ci-dessus, si un seul problème NP-complet appartient à P, alors **tous** les problèmes NP appartiennent à P, c'est-à-dire $\mathsf{P} = \mathsf{NP}$.
Le plus étonnant est qu'il puisse exister des problèmes NP-complet.

Théorème 3 de Cook

SAT est NP-complet.

VI Autres problèmes NP-complet

Pour l'instant nous ne connaissons qu'un problème NPC et sa démonstration est très difficile. Le théorème suivant va permettre d'en déterminer d'autres avec une démonstration moins difficile.

Théorème 4

Si A et B sont deux problèmes de décision tels que A est un problème NP-complet, B est un problème NP et $A \leqslant_{\mathsf{P}} B$ alors B est un problème NP-complet.

Démonstration On commence par prouver la transitivité de la réduction :

$$\text{Si } P_1 \leqslant_{\mathsf{P}} P_2 \text{ et } P_2 \leqslant_{\mathsf{P}} P_3 \text{ alors } P_1 \leqslant_{\mathsf{P}} P_3$$

En effet soient $f : I_{P_1} \mapsto I_{P_2}$ et $g : I_{P_2} \mapsto I_{P_3}$ de complexité polynomiale qui conservent la vérité des instances. Alors $g \circ f$ est définie de I_{P_1} vers I_{P_3} et conserve la vérité des instances.
Si on note $|x|$ la taille d'une instance, la complexité du calcul de $f(x)$ est un $\mathcal{O}\big(|x|^p\big)$ et la complexité du calcul de $g(y)$ est un $\mathcal{O}\big(|y|^q\big)$. De plus $\big|f(x)\big|$ est aussi un $\mathcal{O}\big(|x|^p\big)$ donc la complexité du calcul de $g\big(f(x)\big)$ est en $\mathcal{O}\big(|x|^p\big) + \mathcal{O}\big(\big(|x|^p\big)^q\big) = \mathcal{O}\big(|x|^{pq}\big)$; la complexité reste polynomiale.
$g \circ f$ définit bien une réduction polynomiale.

Comme on a $C \leqslant_{\mathsf{P}} A$ pour tout problème appartenant à NP et $B \leqslant_{\mathsf{P}} B$ on a bien $C \leqslant_{\mathsf{P}} B$ pour tout problème appartenant à NP.
De plus B est lui-même dans NP donc B est bien NP-complet. . . . . ∎

Pour prouver qu'un problème de décision A est NP-complet, il faut donc

1. prouver que A est un problème NP,
2. déterminer un problème B qui soit NP-complet et une réduction polynomiale de B vers A.

VI.1 3-SAT

On commence par une forme de SAT, plus contrainte, ce qui permettra des réductions plus faciles.
Une formule booléenne est sous forme 3-FNC si

- elle sous forme normale conjonctive (conjonction de disjonctions),
- chaque clause est une disjonction de 3 littéraux,
- les variables intervenant dans une clause sont distinctes.

Si on a une formule sous forme normale conjonctive donc les clauses avec moins de 3 littéraux, on peut la transformer en forme normale conjonctive donc les clauses ont 3 littéraux.
On ajoute 2 variables z_1 et z_2.

- Une clause à un seul littéral ℓ est équivalente à la formule sous forme 3-FNC $(\ell \vee z_1 \vee z_2) \wedge (\ell \vee \neg z_1 \vee z_2) \wedge (\ell \vee z_1 \vee \neg z_2) \wedge (\ell \vee \neg z_1 \vee \neg z_2)$.
- Une clause à deux littéraux $\ell_1 \vee \ell_2$ est équivalente à la formule sous forme 3-FNC $(\ell_1 \vee \ell_2 \vee z_1) \wedge (\ell_1 \vee \ell_2 \vee \neg z_1)$.

Cette transformation est linéaire en la taille de la formule et n'ajoute pas de redondance de variable.

La troisième condition n'est pas très contraignante : si une clause contient x et $\neg x$ alors elle est satisfaite par toute valuation, on peut la retirer et si elle contient plusieurs fois un même littéral, on peut enlever les littéraux redondants et reconstituer des 3-clauses comme ci-dessus.

SAT-3

Instance : une formule booléenne φ sous forme 3-FNC.

Question : φ est-elle satisfiable ?

3-SAT est NP

Les instances de SAT-3 sont des instances de SAT donc l'identité permet une réduction polynomiale. Ainsi SAT-3 $\leqslant_P$ SAT donc, d'après le théorème 2, SAT-3 est NP.

Réduction d'un problème NP-complet à 3-SAT

Le seul problème NP-complet connu ici est SAT. On va construire la transformation en plusieurs étapes.

Formule associée à un arbre

On implémente une formule booléenne avec le type OCaml suivant qui fournit un arbre représentant une formule booléenne.
Les noms des variable seront de la forme `"x1"`, `"x2"`, ..., `"xn"` qu'on notera $x_1, x_2, \dots, x_n$.
On note $X = \{x_1, x_2, \dots, x_n\}$ l'ensemble des variables de φ.

```
type formule =   Var of string
               | Non of formule
               | Ou of formule * formule
               | Et of formule * formule
               | Imp of formule * formule
               | Equi of formule * formule
```

On considère une formule booléenne φ définie par son arbre.
On numérote les nœuds de l'arbre de 1 à N, par exemple avec un parcours en largeur, et on définit des nouvelles variables $y_1, y_2, \dots, y_N$, une pour chaque nœud de l'arbre.
En particulier y_1 est la variable associée à la racine.
Pour chaque nœud d'indice i, on définit une nouvelle formule φ_i sur les variables $x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_N$ dont on note Y l'ensemble.

- Si i est l'indice d'un nœud d'étiquette $\neg$ dont le fils est d'indice j, on pose $\varphi_i = y_i \leftrightarrow \neg y_j$.
- Si i est l'indice d'un nœud d'étiquette $*$ ($* \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$) dont les fils sont d'indices j et k, on pose $\varphi_i = y_i \leftrightarrow (y_j * y_k)$.

On pose alors $\Phi = \bigwedge_{1 \leqslant i \leqslant N} \varphi_i$. La construction de Φ est linéaire en fonction de la taille de l'arbre donc en fonction de la taille de φ.

Si V est une valuation sur Y, on note ν sa restriction sur X : $\nu(x_i) = V(x_i)$.
On prouve alors que V est un modèle de Φ si et seulement si $V(y_i)$ vaut l'évaluation de la sous-formule de φ construite à partir du nœud d'indice i. Le résultat est immédiat pour les feuilles et la définition des φ permet de faire une preuve par induction structurelle.
En particulier, dans ce cas, $V(y_1) = \nu(\varphi)$.

Construction de la transformation

On commence par transformer les formules φ_i en formules de forme 3-SAT équivalentes.

- Si le nœud i admet $\neg$ pour étiquette alors

$$\varphi_i = y_i \leftrightarrow \neg y_j \equiv (y_i \rightarrow \neg y_j) \wedge (\neg y_j \rightarrow y_i) \equiv (\neg y_i \vee \neg y_j) \wedge (y_j \vee y_i) = \psi_i$$

- Si le nœud i admet $\vee$ pour étiquette alors

$$\begin{aligned}\varphi_i = y_i \leftrightarrow (y_j \vee y_k) &\equiv \big(y_i \rightarrow (y_j \vee y_k)\big) \wedge \big((y_j \vee y_k) \rightarrow y_i\big) \\ &\equiv \big(\neg y_i \vee (y_j \vee y_k)\big) \wedge \big(\neg(y_j \vee y_k) \vee y_i\big) \equiv (\neg y_i \vee y_j \vee y_k) \wedge (\neg y_j \wedge \neg y_k \vee y_i) \\ &\equiv (\neg y_i \vee y_j \vee y_k) \wedge (\neg y_j \vee y_i) \wedge (\neg y_k \vee y_i) = \psi_i\end{aligned}$$

- Si le nœud i admet $\wedge$ pour étiquette alors

$$\begin{aligned}\varphi_i = y_i \leftrightarrow (y_j \wedge y_k) &\equiv \big(y_i \rightarrow (y_j \wedge y_k)\big) \wedge \big((y_j \wedge y_k) \rightarrow y_i\big)\\ &\equiv \big(\neg y_i \vee (y_j \wedge y_k)\big) \wedge \big(\neg(y_j \wedge y_k) \vee y_i\big)\\ &\equiv \big(\neg y_i \vee y_j\big) \wedge (\neg y_i \vee y_k) \wedge (\neg y_j \vee \neg y_k \vee y_i) = \psi_i\end{aligned}$$

- Si le nœud i admet $\rightarrow$ pour étiquette alors $\varphi_i = y_i \leftrightarrow (y_j \rightarrow y_k)$ et on laisse en exercice que φ_i est équivalent à $\psi_i = (\neg y_i \vee \neg y_j \vee y_k) \wedge (y_j \vee y_i) \wedge (\neg y_k \vee y_i)$.
- Si le nœud i admet $\leftrightarrow$ pour étiquette alors $\varphi_i = y_i \leftrightarrow (y_j \leftrightarrow y_k)$ et on laisse en exercice que φ_i est équivalent à $\psi_i = (y_i \vee y_j \vee y_k) \wedge (\neg y_i \vee \neg y_j \vee y_k) \wedge (\neg y_i \vee y_j \vee \neg y_k) \wedge (y_i \vee \neg y_j \vee \neg y_k)$.

La formule $\Psi' = \bigwedge_{1 \leqslant i \leqslant N} \psi_i$ est alors une formule sous forme FNC équivalente à Φ. et $\Psi = y_1 \wedge \Psi'$ est aussi une formule sous forme FNC.
La construction de Ψ est linéaire en fonction de la taille de Φ donc en fonction de la taille de φ.

Pour l'instant les clauses ont 1, 2 ou 3 littéraux ; on a vu qu'on peut construire une formule sous forme 3-FNC Ψ_3 équivalente à Ψ en temps proportionnel à la taille de Ψ. La complexité a toujours une complexité linéaire.

Conservation de la vérité

Si φ est satisfiable il existe une valuation ν sur X qui soit un modèle de φ. On a vu ci-dessus qu'il existait un prolongement V de ν sur Y qui est un modèle de Φ avec $V(y_1) = \nu(\varphi)$; c'est donc un modèle de $y_1 \wedge \Phi$ d'où un modèle de Ψ_3 : Ψ_3 est une instance positive de 3-SAT.

Il reste à prouver qu'une instance négative, c'est-à-dire une fonction φ non satisfiable est transformée en une fonction Ψ_3 non satisfiable. On va prouver la contraposée.
Si Ψ_3 est satisfiable alors il existe un modèle V pour Ψ_3 donc pour $y_1 \wedge \Phi$.
On doit donc avoir $V(y_1)$ évalué à vrai et V est un modèle de Φ. On a vu que, dans ce cas, $V(y_1) = \nu(\varphi)$ où ν est la restriction de V à X. Ainsi φ est satisfiable, c'est une instance positive de SAT. La réduction est polynomiale et conserve la vérité des instances : SAT $\leqslant_{\mathsf{P}}$ 3-SAT.
On a donc bien prouvé que 3-SAT est NP-complet.

VI.2 Clique

Un premier exemple de problème sur les graphes.
Une clique d'un graphe est un graphe induit complet, c'est -à-dire un sous ensemble de sommets tels toute paire de sommets dans cet ensemble est relié par une arête.

CLIQUE

Instance : un graphe non orienté $G = (S, A)$ et un entier k.

Question : G admet-il une clique à k éléments ?

Clique est NP

Un certificat est la définition d'une partie S' de S.
La vérification consiste à vérifier que toute paire de sommet est bien une arête de G.
Le certificat et la vérification sont polynomiaux en la taille du graphe.

Réduction d'un problème NP-complet à CLIQUE

On choisit de réduire 3-SAT à CLIQUE.
Soit $\varphi = \bigwedge_{i=1}^{m} C_i$ une instance de 3-SAT, avec $C_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$ dont les littéraux sont définis sur un ensemble de variables $X = \{x_1, \ldots, x_n\}$.
On définit $k = n$ et $G_\varphi = (S, A)$ le graphe défini par

- Chaque clause C_i définit 3 sommets, un par littéral, $s_{i,1}$, $s_{i,2}$, $s_{i,3}$.
- Pour $(s,t) \in S^2$, $\{s,t\} \in A$ si et seulement si s et t ne représentent pas deux littéraux d'une même clause et ne représentent pas deux littéraux qui sont la négation l'un de l'autre.

Le graphe à $3n$ sommets et au plus $\frac{9}{2}n^2$ arêtes donc il est de taille polynomiale en la taille de φ.

Si φ est une instance positive de 3-SAT, il admet un modèle, on choisit un littéral évalué à vrai dans chaque clause ; les sommets associées ne sont pas négation l'un de l'autre et appartiennent à des clauses distinctes. Il définissent donc une clique de n éléments de G_φ.
Ainsi (G_φ, n) est une instance positive de CLIQUE.

Si (G_φ, n) est une instance positive de CLIQUE, il admet une clique de n sommets. Les sommets de la clique appartiennent à des clauses distinctes sinon ils ne pourraient pas être connectés. On définit ainsi un littéral pour chaque clause.
Les littéraux associés à une même variable sont positifs ou négatifs en même temps sinon ils ne pourraient pas être connectés. On peut donc donc définir une valeur vraie à chaque littéral et définir de cette manière une valuation sur X qui satisfait φ.
Ainsi φ est une instance positive de 3-SAT.

VI.3 SOMME PARTIELLE

On passe maintenant à un problème plus mathématique

SOMME PARTIELLE

Instance : une suite finie d'entiers positifs $(a_1, a_2, \ldots, a_n)$ et un entier s.

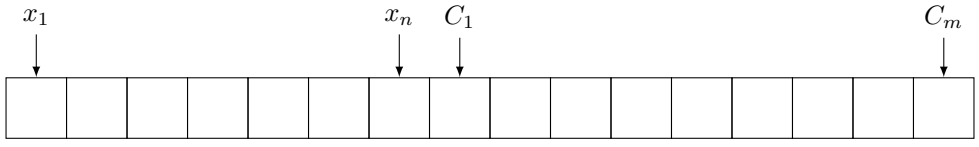
Question : existe-t-il une partie $I \subset \{1, 2, \ldots, n\}$ telle que $\sum_{i \in I} a_i = s$?

SOMME PARTIELLE est NP

Un certificat est la définition d'une partie I de $\{1, 2, \ldots, n\}$ par un tableau de taille n de booléens (par exemple).
La vérification consiste à calculer la somme et la comparer à s
Le certificat et la vérification sont polynomiaux en la taille de l'instance.

Réduction d'un problème NP-complet à SOMME PARTIELLE

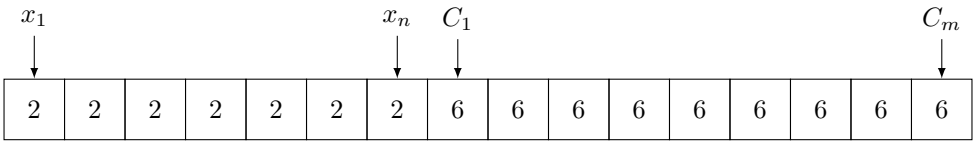
On choisit de réduire 3-SAT à SOMME PARTIELLE.
Soit $\varphi = \bigwedge_{i=1}^{m} C_i$ une instance de 3-SAT, avec $C_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$ dont les littéraux sont définis sur un ensemble de variables $X = \{x_1, \ldots, x_n\}$.
Chacune des m clauses est définie par 3 éléments à choisir parmi $2n$: la taille d'une instance est de l'ordre de $m \cdot \log_2(n)$. On ne va considérer que des variables utilisées dans la formule : il y en a au plus $3m$ donc $n = \mathcal{O}(m)$.
On va construire $2n + 2m$ entiers par leur écriture en base 10 avec $n + m$ chiffres : la taille totale est quadratique donc polynomiale en la taille de φ.



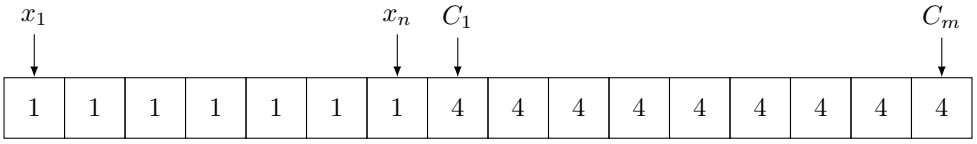
- Pour chaque variable x_i on définit l'entier v_i pour lequel le chiffre correspondant à x_i est 1, le chiffre correspondant à C_j vaut 1 si x_i est une des littéraux de C_j, les autres chiffres sont nuls
- Pour chaque variable x_i on définit v'_i pour lequel le chiffre correspondant à x_i est 1, le chiffre correspondant à C_j vaut 1 si $\neg x_i$ est une des littéraux de C_j, les autres chiffres sont nuls

- Pour chaque clause C_i on définit c_i pour lequel le chiffre correspondant à C_i est 1, les autres chiffres sont nuls et c'_i pour lequel le chiffre correspondant à C_i est 2, les autres chiffres sont nuls

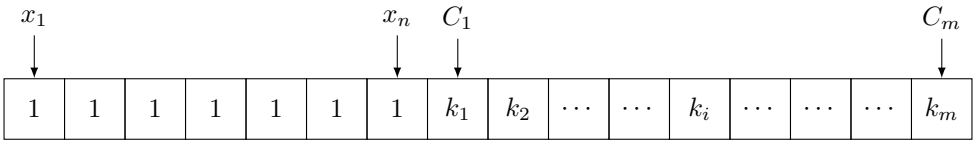
Les caractéristiques d'une formule sous forme 3-CNF impliquent que la somme des entiers est



On définit K comme l'entier



Si ν est un modèle pour φ, on $a_i = v_i$, si $\nu(x_i)$ vaut vrai et $a_i = v'_i$, si $\nu(x_i)$ vaut faux ; la somme $\sum_{i=1}^{n} a_i$ donne une valeur 1 pour chaque colonne x_i et une valeur de 1 à 3 pour chaque colonne C_i



Si $k_i = 1$ on ajoute c_i et c'_i, pour $k_i = 2$, on ajoute c'_i et pour $k_i = 3$, on ajoute c_i. La somme de ces nombres, ajoutée à $\sum_{i=1}^{n} a_i$, donne bien K. Ainsi une instance positive de 3-SAT est transformée en une instance positive de SOMME PARTIELLE.

Inversement si l'instance $\big((v_1, v_2, \ldots, v_n, v'_1, v'_2, \ldots, v_n, c_1, c_2, \ldots, c_m, c'_1, c'_2, \ldots, c'_m), K\big)$ est positive on considère une somme égale à K.
Le chiffre dans la colonne x_i est 1 donc, dans la somme, il y a un seul des deux entiers v_i ou v'_i. On définit donc $\nu(x_i)$ comme valant vrai si v_i est dans la somme et valant faux si v'_i est dans la somme : on définit ainsi une valuation sur l'ensemble des variables.
Le chiffre dans la colonne C_i est 4 et les chiffres c_i et c'_i ne peuvent avoir apporté qu'une somme entre 0 et 3. Cela signifie que les entiers v_j ou v'_j ont apporté une valeur au moins 1. IL existe donc au moins un entier j tel que x_j est un littéral de la clause C_i si c'est v_j qui est présent dans la somme, c'est-à-dire si $\nu(x_i)$ vaut vrai ou $\neg x_j$ est un littéral de la clause C_i si c'est v'_j qui est présent dans la somme, c'est-à-dire si $\nu(x_i)$ vaut faux. Dans les deux cas ν satisfait C_i.
Ainsi ν est un modèle de φ : φ est positive.
On a bien 3-SAT $\leqslant_{\mathsf{P}}$ SOMME PARTIELLE et SOMME PARTIELLE est NP-complet.