

Complexité et Calculabilité : TD1

SAT et réductions

1.1 Temps de calcul, représentation de l'entrée

Exercice 1.1

1. Un graphe (orienté) est constitué de la donnée d'un ensemble fini de sommets V (« vertices ») et d'un ensemble fini d'arcs E (« edges »). Pour appliquer un algorithme à un graphe, il faut d'abord être capable de stocker ces informations en mémoire. Quelles informations doit-on stocker pour cela ?
2. Un codage classique d'un graphe consiste à le représenter par sa *matrice d'adjacence*. Un graphe à n sommets est alors représenté par une matrice de taille $n \times n$ dont la case (i, j) comporte un 1 si il y a un arc de i à j et un 0 sinon. Quelle est la taille de ce codage ? On l'exprimera en fonction des deux paramètres $n = |V|$ (nombre de sommets) et $m = |E|$ (nombre d'arcs).
3. Un autre codage classique est la *liste d'adjacence*. Dans ce cas, à chaque sommet i (qu'on a au préalable ordonnés de 1 à n), on associe la liste des sommets j tels qu'il existe un arc de i à j . Quelle est la taille de ce codage ? Dans quel cas peut-il être avantageux ?
4. Les algorithmes de *parcours en largeur* (breadth-first-search) et de *parcours en profondeur* (depth-first-search) peuvent être programmés, pour des graphes représentés par liste d'adjacence, de manière à s'exécuter en temps $\Theta(n + m)$, où n désigne le nombre de sommets et m le nombre d'arcs.
Rappel : « temps $\Theta(f)$ » signifie que le temps est linéairement proportionnel à f .
Quel est, au maximum (pour un graphe simple), l'ordre de grandeur de m par rapport à n ? On affirme fréquemment que la complexité des parcours de graphes est linéaire ; au vu de ce qui précède, cela vous semble-t-il correct ?
5. Y a-t-il une différence entre les notions de complexité *polynomiale en n* , *polynomiale en $n + m$* , *polynomiale en la taille de la représentation du graphe* ? (en supposant que le graphe est simple et représenté soit par matrice d'adjacence, soit par liste d'adjacence)

Exercice 1.2

On rappelle que le problème SAT est : étant donné une formule booléenne, est-elle satisfaisable ? Donc est-ce qu'elle possède une valuation qui la rend vraie ?

Entrée : Une formule booléenne φ sur les variables x_1, \dots, x_n .

Sortie : Existe-t-il une valuation des variables x_1, \dots, x_n qui rend φ vraie ? (On dit dans ce cas que φ est satisfaisable.)

On considèrera en particulier les formules en *forme normale conjonctive*. Un *littéral* est une variable x ou la négation $\neg x$ d'une variable. Une *clause* est une *disjonction* (un OU) de littéraux : $c = \ell_1 \vee \dots \vee \ell_k$. Une formule est en forme normale conjonctive si c'est une *conjonction* (un ET) de clauses : $\varphi = c_1 \wedge \dots \wedge c_n$. Ainsi $(a \vee b) \wedge (\neg a \vee c \vee \neg d) \wedge (b \vee \neg c)$ est en forme normale conjonctive, mais pas $\neg(a \vee b)$ ou $(a \wedge b) \vee c$.

Pour résoudre le problème SAT, il faut être capable de stocker les formules booléennes en mémoire.

1. Si on se restreint aux formules en forme normale conjonctive, y a-t-il des informations que l'on peut omettre dans le codage ? Comment peut-on coder une variable ? Sa négation ? Comment coder une clause ? Comment coder une conjonction de clauses ?
2. Quel espace mémoire occupera votre codage ? Il convient de l'exprimer en fonction du nombre de variables, et de la taille des clauses (la taille d'une clause étant son nombre de littéraux).

1.2 SAT

Exercice 1.3

1. Donner un algorithme permettant de déterminer si une formule booléenne en forme normale disjonctive est satisfaisable. Quelle est sa complexité ?
2. Mettre la formule $(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge (x_3 \vee y_3)$ en forme normale disjonctive (*i.e.*, donner une formule en forme normale disjonctive qui est satisfaite par exactement les mêmes valuations). Quelle est la taille de cette formule ? Quelle sera la taille de la forme normale disjonctive si on a n clauses dans la formule précédente ?

Exercice 1.4

Une clause de Horn est une clause (sous forme de disjonction de littéraux) contenant *au plus* un littéral positif. Une formule de Horn-SAT est une conjonction de clauses de Horn.

Un exemple de formule de Horn-SAT est : $(a \vee \neg b \vee \neg c \vee \neg d) \wedge (\neg a \vee \neg b) \wedge b \wedge (\neg b \vee c) \wedge (\neg b \vee d)$.

1. Pouvez-vous trouver, pour une clause de Horn quelconque, une formule contenant uniquement des littéraux positifs, des ET et une implication, qui lui est équivalente ? (distinguer selon que la clause comporte ou non un littéral positif)
Rappel : l'implication $x \longrightarrow y$ est équivalente à $\neg x \vee y$.
2. Soit une formule de Horn-SAT dont les clauses contiennent toutes un littéral négatif. Est-elle satisfaisable ?
3. Qu'est-ce qu'une clause de Horn sans littéraux négatifs ? Que force-t-elle à faire pour satisfaire la formule ?
4. Déduisez-en un algorithme pour déterminer si une formule de Horn-SAT est satisfaisable. Appliquer l'algorithme à la formule donnée en exemple. Quelle est, en général, la complexité de l'algorithme ?

Exercice 1.5

Dans cet exercice, on considère plusieurs type d'objets qu'on identifiera par des variables

particulières. Ainsi, x_{17} est une variable, ℓ_{34} et m_3 sont des littéraux, φ_{42} est une clause, ϕ est une formule et s_i est un sommet.

Le problème 2-SAT s'énonce comme suit :

Entrée : une formule booléenne à n variables x_1, \dots, x_n , ayant la forme

$$\phi = \varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_m$$

où chaque clause φ_k est de la forme $\varphi_k = \ell_k \vee m_k$ où ℓ_k et m_k sont des littéraux (donc variables ou négations de variables).

Sortie : La formule ϕ est-elle satisfaisable? (Version calculatoire : Si oui, trouver une valuation des variables x_1, \dots, x_n qui rend la formule vraie.)

1. Parmi les formules suivantes, lesquelles sont des entrées valables au problème 2-SAT?
 - $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee \neg x_3)$.
 - $(x_1 \vee x_1) \wedge (\neg x_2 \vee x_3)$.
 - $(x_1 \vee x_2) \wedge \neg x_1 \wedge (x_2 \vee \neg x_3)$.
2. On considère une clause $\ell \vee m$ d'une formule 2-SAT ϕ . Si ℓ vaut FAUX, que doit valoir m pour que la formule soit satisfaite? Déduisez-en une conjonction d'implications équivalente à ϕ .
3. On cherche maintenant à réduire le problème 2-SAT à un problème sur un graphe dirigé. Pour cela, à chaque variable booléenne x , on associe deux sommets s_x et $s_{\neg x}$, et pour chaque implication $\ell \Rightarrow m$, on ajoute un arc (s_ℓ, s_m) au graphe. Quelle condition sur le graphe est équivalente à la satisfaisabilité de la formule?
4. En utilisant l'algorithme de Warshall (ou le parcours en profondeur), il est possible de calculer en temps polynomial la clôture transitive d'un graphe. Qu'en déduisez-vous sur la complexité de 2-SAT?
5. *Bonus :* Décrivez un algorithme calculant la clôture transitive d'un graphe.

1.3 Réductions

Exercice 1.6

Le problème de *2-coloriage* d'un graphe est le suivant :

Entrée : Un graphe non-orienté (V, E) .

Sortie : Est-il possible de colorier V avec 2 couleurs tel que toute arête ait des extrémités de couleurs différentes?

1. Montrez que ce problème se réduit à SAT, c'est-à-dire que pour tout graphe (V, E) , on peut construire en temps polynomial une formule de SAT qui est satisfaisable si et seulement si le graphe est 2-coloriable.
2. Combien de variables contient chaque clause de votre formule? Qu'en déduisez-vous sur la complexité du problème de 2-coloriage d'un graphe?

Exercice 1.7

Une instance de 3-SAT est une formule de SAT dont toutes les clauses ont *exactement* trois littéraux. Une instance de ≤ 3 -SAT est une formule de SAT dont toutes les clauses ont *au plus* trois littéraux.

1. Montrez que ≤ 3 -SAT se réduit polynomialement à 3-SAT.
2. Montrez que 3-SAT se réduit polynomialement à ≤ 3 -SAT.

Exercice 1.8

On considère le problème SOMME D'ENTIERS suivant :

Entrée : Des entiers positifs x_1, x_2, \dots, x_n et un entier N .

Sortie : Existe-t-il une sous-suite $1 \leq i_1 < i_2 < \dots < i_p \leq n$ telle que $x_{i_1} + x_{i_2} + \dots + x_{i_p} = N$?

1. On suppose que l'ensemble $X = \{x_1, \dots, x_n\}$ est déjà trié $x_1 \leq x_2 \leq \dots \leq x_n$. Pour le problème SOMME D'ENTIERS proposer un algorithme qui utilise une mémoire de taille $O(N)$ et dont le temps de calcul est $O(nN)$.
2. Il est connu que le problème SOMME D'ENTIERS est NP-complet. Pourquoi l'existence de l'algorithme de la question 1 ne contredit-il pas la NP-complétude du problème ?