ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
Section d'Informatique et de Systèmes de Communication

Corrigé de la série 13 19 Dec. 2011

1. La taille de l'input

a) $\log_2(n)$.

b) n^2.

c) $\lceil \log_2(1000) \rceil n = 10n$. (On peut aussi faire légèrment mieux, en représentant les entiers $a_1, \ldots, a_n$ comme un grand entier $\sum_{i=1}^{n} 1000^{i-1} a_i$, ce qui donne $\lceil \log_2(1000) n \rceil$.)

d) $n^2 \cdot \log_2(201+1) = O(n^2)$.
Explication : On a n^2 valeurs dans la matrice d'adjacence, pour chaque valeur on a besoin de $\log_2(201+1)$, bits : 201 pour les valeurs possibles de -100 à 100 et une valeur de plus pour codifier le fait qu'il n'y a pas d'arête entre les deux sommets.

2. Réduction polynomiale

a) Supposons que $G = (V, E)$, et soit u le sommet ajouté pour obtenir $f_1(G)$. Ainsi $f_1(G) = (V_1, E_1)$ où $V_1 = V \cup \{u\}$, et $\forall v \in V_1 : \ (u, v_i), (v_i, u) \in E_1$.

$\Rightarrow$. Supposons que G contient une clique $S \subseteq V$ de taille $\geq k$. Puisque u est connecté à tous les autres sommet,

$$S \cup \{u\} \subseteq V_1$$

forme une clique de taille $|S| + 1 \geq k + 1$.

$\Leftarrow$. Supposons que $f_1(G)$ contient une clique $S \subseteq V_1$ de taille $\geq k+1$. Nous distinguons 2 cas :

• **Cas 1 :** $u \notin S$. Dans ce cas nous avons $S \subseteq V$, et donc cette clique de taille $\geq k+1$ est aussi dans G. G contient donc bien une clique de taille $\geq k$.

• **Cas 2 :** $u \in S$. Si nous retirons u de la clique nous obtenons une clique de taille $|S| - 1$, donc de taille $\geq k$. puisque tous les sommets de V_1 sauf u sont aussi dans V, $S - \{u\} \subseteq V$ forme une clique de taille $\geq k$ dans le graphe G.

b) Nous ajoutons a sommets $u_1, \ldots u_a$, qui sont connectés à tous les autres sommets (et aussi connecté entre eux). Soit $U = \{u_1, \ldots, u_a\}$. Formellement si $G = (V, E)$ alors $f_a(G) = (V_a, E_a)$ où

$$\begin{aligned} V_a &= V \cup U \\ E_a &= E \cup (V_a \times U) \cup (U \times V_a) \cup (V_a \times V_a) \end{aligned}$$

Nous pouvons montrer de manière identique au point a) que G contient une clique de taille $\geq k$ si et seulement si $f_a(G)$ contient une clique de taille $\geq k + a$.

c) Nous voyons d'abord que

$$n_a = |V_a| = |V| + a = m + a.$$

Nous savons que G contient une clique de taille $\geq k$ si et seulement si $f_a(G)$ contient une clique de taille $\geq k+a$. Nous voulons donc choisir a tel que $n_a/2 = k+a$.

$$\begin{array}{rcl} \frac{n_a}{2} = k+a & \iff & \frac{m+a}{2} = k+a \\ & \iff & m+a = 2k+2a \\ & \iff & a = m-2k \end{array}$$

Puisque $k \leq m/2$, nous voyons que cette valeur est ≥ 0.

d) Il nous reste à couvrir le cas où $k > m/2$. Nous construisons le graphe $h(G)$ en ajoutant à V $2k-m$ sommets sans aucune arête. Formellement, $h(G) = (V', E')$ avec

$$\begin{array}{rcl} V' & = & V \cup \{w_1, \ldots, w_{2k-m}\} \\ E' & = & E. \end{array}$$

Puisque les nouveaux sommets n'ont pas d'arêtes, il est clair que G contient une clique de taille $\geq k$ si et seulement si $h(G)$ contient une clique de taille $\geq k$.
Ainsi la réduction est la suivante :

$$f(G,k) = \begin{cases} f_{m-2k}(G) & \text{si } k \leq \frac{m}{2} \\ h(G) & \text{sinon} \end{cases}$$

3. SAT

a) Non.

b) $x_1 = \text{FALSE}$, $x_2 = \text{FALSE}$, $x_3 = \text{TRUE}$.

c) Non.

d) Un input consiste en m clauses. La $j^{\text{éme}}$ clause contient k_j littéraux, elle aura donc comme taille $O(k_j)$: pour représenter $\lambda_1 \vee \ldots \vee \lambda_{k_j}$ il faut juste indiquer ce que valent chacun des k_j littéraux (les valeurs possibles sont $x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}$).
Donc la taille totale de l'input est

$$O\big(\sum_{j=1}^{m} k_j \log n\big) = O\big(m \cdot k \log n\big), \quad \text{où } k = \max_j k_j.$$

e) Il faut trouver une preuve vérifiable en temps polynomial. La preuve est une attribution satisfaisante, il est clair qu'on peut vérifier qu'une attribution est satisfaisante en temps polynomial : il faut faire $\leq m \cdot (k-1)$ opérations OR ($\vee$), et $m-1$ opérations AND ($\wedge$).

f) Puisque SAT $\in$ NP (question précédente), si P=NP on aura SAT $\in$ P. Donc il existe un algorithme qui résoud SAT en temps polynomial. Appelons cet algorithme $\mathcal{A}$. Donc on donne à $\mathcal{A}$ comme input une formule CNF F, et il retournera TRUE si F est satisfaisable et FALSE sinon (et son temps est polynomial en la taille de F). Nous allons utiliser $\mathcal{A}$ pour construire l'algorithme dont nous avons besoin.
On commence par donner comme input à $\mathcal{A}$ la formule F originale. Si $\mathcal{A}$ retourne FALSE nous savons que F n'est pas satisfaisable. Sinon nous savons qu'il existe une attribution satisfaisante, il nous faut donc la trouver. L'idée est d'utilise $\mathcal{A}$ pour verifier (en temps polynomial) s'il existe une attribution satisfaisante avec $x_1 = \text{TRUE}$ ou avec $x_1 = \text{FALSE}$.

$\mathcal{A}$ retournera forcément TRUE dans au moins un des 2 cas (puisque nous savons que F est satisfaisable). Si par exemple $\mathcal{A}$ retourne TRUE quand $x_1 =$ TRUE, nous posons $x_1 =$ TRUE et faisons de même avec $x_2, x_3, \ldots, x_n$.
Quel est l'input exact qu'il faut donner à $\mathcal{A}$ pour savoir s'il existe une attribution satisfaisante avec $x_1 =$ TRUE ? Il faut donner une version modifiée de F : Toutes les clauses qui contiennent x_1 peuvent être enlevées (puisqu'elles sont toutes vraies quelles que soitent les valeurs de $x_2, \ldots, x_n$), et nous pouvons enlever $\overline{x_1}$ de toutes les clauses qui le contiennent, puisque

$$\lambda_1 \vee \ldots \vee \lambda_\ell \vee \overline{x_1} \quad = \quad \lambda_1 \vee \ldots \vee \lambda_\ell \quad \text{si } \overline{x_1} = \text{FALSE}.$$

Ainsi pour obtenir cette version modifiée F' de F on enlève certaines clauses, et on enlève certains litteraux dans d'autres clauses, la taille de F' sera donc $\leq$ à la taille de F.

$$\begin{array}{lcl} \#\text{clauses dans } F' & \leq & \#\text{clauses dans } F \\ \text{taille des clauses dans } F' & \leq & \text{taille des clauses dans } F \end{array}$$