
Documents (cours et notes personnelles) autorisés.

Solveur surpuissant

On dispose d'un solveur surpuissant capable de résoudre n'importe quel problème NP de décision sur les graphes en 1h de temps, à condition que le graphe soit simple et non orienté, et surtout que l'instance se code sur au plus 1 000 bits.

Question 1. Donner un exemple de problème de décision qui soit NP-complet pour les graphes simples et non-orientés.

SOLUTION. 3-coloration ou Hamiltonisme, mais pas Vertex Cover de taille k (nécessite de coder k).

Question 2. Quel est le nombre maximum de sommets n du graphe pour lequel on est sûr de pouvoir résoudre un tel problème en 1h de temps avec un tel solveur ?

SOLUTION. On code le graphe G par une demi-matrice supérieure d'adjacence, sans sa diagonale. Donc on cherche le n maximum tel que $(n-1) + (n-2) + \dots + 2 + 1 = n(n-1)/2 \leq 1\,000$, ce qui implique que $n = 45$, puisque $44 \times 45 = 1\,980$ et $45 \times 46 = 2\,070$.

Question 3. Quelle serait la meilleure approche entre un algorithme naïf hypothétique de complexité 2^n exécuté sur un ordinateur à 1 GHz ou le solveur sur 1 000 bits ?

SOLUTION. Le solveur ! qui va prendre 1h au lieu de presque 10h sinon. En effet, l'approche naïve, pour effectuer ces 2^{45} instructions, va prendre sur un ordinateur 1 GHz un temps de $(2^{45}/10^9)/3\,600$ heures $\approx 9\text{h}46$.

Vertex Cover

Question 4. Donner un algorithme efficace qui résout Vertex Cover de taille k pour les graphes de degré maximum au plus 1. Donner sa complexité en fonction du nombre de sommets n et d'arêtes m du graphe.

SOLUTION. Lorsque le degré maximum au plus 1, le graphe est composé d'arêtes isolées et de sommets isolés. Pour couvrir les arêtes de ce graphe, il faut donc choisir exactement un sommet par arête. Le problème revient donc à savoir si $m \leq k$ ou pas. En supposant une représentation du graphe par liste d'adjacence, on peut le faire en temps $O(n)$ en parcourant les n listes tout en vérifiant si leur longueur est ≤ 1 (arête) ou pas (sommet isolé).

Bien sûr, si m était connu avec une entrée composée du graphe G et de son nombre d'arêtes m (ce qui n'était pas le cas ici), alors on pouvait répondre à la question en temps constant avec un simple `Return (m<=k)` ;

Question 5. Proposer un algorithme pour Vertex Cover dans le cas général qui soit récursif et basé sur le principe que soit le sommet u est dans la solution, soit tous ses voisins le sont, avec comme cas terminal le cas où le degré maximum est au plus 1.

SOLUTION. C'est une simple adaptation de l'algorithme vu en cours, en passant d'un degré maximum 2 par un degré maximum 1. En utilisant les mêmes notations que le cours pour $E(G)$, $\Delta(G)$, $N(u)$, cela donne :

Algorithme VC(G, k)

1. Si $E(G) = \emptyset$ et $k \geq 0$, renvoyer VRAI.

2. Si $k \leq 0$, renvoyer FAUX.
 3. Si $\Delta(G) \leq 1$, renvoyer $\text{VCdeg1}(G, k)$.
 4. Choisir un sommet u de G de degré ≥ 2 .
 5. Construire $G_1 := G \setminus \{u\}$ et $G_2 := G \setminus N(u)$.
 6. Renvoyer $\text{VC}(G_1, k - 1) \vee \text{VC}(G_2, k - \deg_G(u))$.
-

Question 6. Donner l'équation de récurrence que doit vérifier la complexité de votre algorithme. Justifiez.

SOLUTION. Si on note $T(n, m, k)$ la complexité de l'algorithme $\text{VC}(G, k)$ pour un graphe à n sommets et m arêtes, alors il vient l'équation : $T(n, m, k) = O(n + m) + T(n, m, k - 1) + T(n, m, k - 2)$, alors $T(n, m, 0) = T(n, m, 1) = O(n + m)$.

Il est cependant plus simple de lier cette complexité au nombre maximum d'appels récursifs pour $\text{VC}(G, k)$. Notons a_k ce nombre. À cause de la dernière ligne, a_k doit vérifier l'équation $a_k = a_{k-1} + a_{k-2}$. Clairement la 2e ligne implique que $a_0 = 0$, et il est facile de voir que $a_1 = 2$.

Question 7. En déduire sa complexité, éventuellement un majorant. Justifiez.

SOLUTION. Puisque $\text{VC}(G, k)$ possède un nombre constant d'étape, il est plus simple de majorer la complexité de $\text{VC}(G, k)$ par la complexité de l'étape la plus coûteuse, soit la 5e avec $O(n + m)$, fois a_k , le nombre maximum d'appels récursifs.

Pour résoudre la récurrence sur $a_k = a_k = a_{k-1} + a_{k-2}$ (c'est la même que pour les nombres de Fibonacci), il suffit de résoudre l'équation associée $x^k = x^{k-1} + x^{k-2} \Leftrightarrow x^0 = x^{-1} + x^{-2} \Leftrightarrow x^2 = x + 1 \Rightarrow x = (1 + \sqrt{5})/2 = \varphi \approx 1.61$, le nombre d'or. On a alors $a_k = O(\varphi^k)$, la constante multiplicative exacte c dans le « O » ne dépendant que des valeurs initiales a_0 et a_1 . En fait, on a $c = \max_{k \in \{0,1\}} \{a_k / \varphi^k\} = 2/\varphi \approx 1.23$.

La complexité de $\text{VC}(G, k)$ est donc majorée par $O(a^k \cdot (n + m)) = O(\varphi^k \cdot (n + m))$.

Max Fun

On dira qu'un problème est un problème de « type sommet » s'il s'agit de trouver dans un graphe un sous-ensemble de sommets vérifiant une certaine propriété, l'objectif étant d'en trouver un de taille minimum, maximum, ou de taille k donné suivant la variante. La variante « valué » d'un problème de type sommet est le même problème mais dans un graphe où les sommets possèdent des poids (non négatifs). Il s'agit alors de trouver un ensemble de sommets solution de poids minimum, maximum, ou de poids k donné.

Par exemple, ENSEMBLE DOMINANT VALUÉ MINIMUM est le problème consistant à trouver un ensemble dominant dont la somme des poids de l'ensemble est le plus petit possible.

On considère le problème décrit informellement comme suit. Vous voulez organiser une fête avec potentiellement tous vos collègues d'une grande entreprise. Chaque collègue x a un « taux de fun » individuel bien déterminé, noté $\text{fun}(x)$, mais aussi a un chef de service, un chef direct noté $\text{boss}(x)$, sauf peut-être le PDG de l'entreprise. Inviter un collègue x et son chef $\text{boss}(x)$ n'est pas fun du tout. Vous voulez absolument l'éviter. Votre objectif est donc d'inviter certains de vos collègues, sans leurs chefs, de façon à maximiser le fun total. On supposera valide une solution où vous n'êtes pas invité.

Vous noterez que la relation $x \mapsto \text{boss}(x)$ définit un arbre T dont la racine est le PDG. Pour tout sommet x , on notera $T[x]$ le sous-arbre composé de x et de tous ses descendants.

Question 8. Donnez un problème de type sommets bien connu permettant de résoudre MAX FUN.

SOLUTION. Ensemble indépendant de poids maximum dans un arbre, car chaque arête est du type $x \mapsto \text{boss}(x)$ ne peut avoir que x ou $\text{boss}(x)$, mais pas les deux. Il faut maximiser le taux de fun total de sommets sélectionnés.

On se propose de résoudre le problème MAX FUN par programmation dynamique, en partant des feuilles. On introduit les deux variables suivantes, définies pour chaque sommet x de T .

- $A[x]$ est le fun total maximum du sous-arbre $T[x]$; et
- $B[x]$ est le fun total maximum de $T[x]$ sans que x soit invité.

Question 9. Soit x un sommet de T ayant pour fils $x_1, \dots, x_t$. Donnez des formules de récurrence entre $A[x]$, $B[x]$ et les fils de x .

SOLUTION.

- $B[x] = \sum_{i=1}^t A[x_i]$, car si x n'est pas invité, alors on peut calculer A librement sur chacun des fils.
- $A[x] = \max \{ B[x], \text{fun}(x) + \sum_{i=1}^t B[x_i] \}$, car ou bien x n'est pas invité, et c'est $B[x]$ par définition de B , ou bien x est invité, et donc aucun de ses fils ne doit l'être. Bien entendu, on cherche le maximum de ces deux valeurs.

Question 10. Détailler le code permettant de résoudre MAX FUN et donnez sa complexité en fonction du nombre de sommets de T . Justifiez.

SOLUTION. À l'aide des formules précédentes, on peut calculer $A[x]$ et $B[x]$ à partir des feuilles et remonter jusqu'à la racine.

Avec la technique de mémorisation (programmation dynamique), on peut même l'écrire simplement avec une fonction récursive en charge de remplir deux variables globales, les tableaux contenant les valeurs $A[u]$ et $B[u]$ pour chaque sommet u de T .

La fonction suivante $\text{MaxFun}(T, x)$ calcule ainsi $A[x]$, le max de fun du sous-arbre $T[x]$ en utilisant des tableaux A et B précédemment alloués qu'il n'est pas nécessaire d'initialiser. Ce qui nous intéresse c'est donc $\text{MaxFun}(T, \text{PDG})$.

Algorithme $\text{MaxFun}(T, x)$

1. Si x est une feuille, alors poser $A[x] := \text{fun}(x)$ et $B[x] := 0$.
 2. Sinon, soit $x_1, \dots, x_t$ les fils de x :
 - (a) calculer $B[x] := \sum_{i=1}^t \text{MaxFun}(T, x_i)$
 - (b) calculer $A[x] := \max \{ B[x], \text{fun}(x) + \sum_{i=1}^t B[x_i] \}$
 3. Renvoyer $A[x]$
-

Bien sûr, les $\sum_{i=1}^t$ s'implémenter par de simple boucles **for**.

Avec la mémorisation, l'étude de la complexité est moins facile, mais c'est $O(n)$, où n est le nombre de sommets de $T[x]$. En effet, les appels $\text{MaxFun}(T, x)$ sont empilés uniquement lorsque les valeurs $A[x]$ ne sont pas connues. Sinon, c'est une simple lecture en mémoire, en temps constant. Il n'y a donc que $O(n)$ appels récursifs en tout et pour tout lors du calcul de $\text{MaxFun}(T, x)$.

Puis le calcul de $\text{MaxFun}(T, x)$, hors les appels récursifs, est proportionnel aux nombres de fils de x : $O(\deg_T(x))$. La complexité est donc $O(\sum_{y \in V(T[x])} \deg_T(y)) = O(n)$.

Question 11. Comment définiriez-vous les variables A et B si, à juste titre, vous ne voulez retenir que la solution où vous faites partie des invités ?

SOLUTION. Si on veut garder l'individu y dans la solution, alors on peut redéfinir $A[x]$ ainsi, $B[x]$ restant identique :

$$A[x] = \begin{cases} \text{fun}(x) + \sum_{i=1}^t B[x_i] & \text{si } x = y \\ \max \{ B[x], \text{fun}(x) + \sum_{i=1}^t B[x_i] \} & \text{sinon} \end{cases}$$

FIN.