

TD n°14 Examen 2008

1 Questions de cours pour la chauffe

Soient P_1 et P_2 deux problèmes de décision, et supposons qu'on connaisse une transformation polynomiale (une réduction) de P_1 en P_2 . Répondre aux sept questions suivantes avec un maximum de deux lignes de justification par question.

1. Si $P_1 \in \mathcal{P}$, a-t-on $P_2 \in \mathcal{P}$?

Solution : *Non.*

□

2. Si $P_2 \in \mathcal{P}$, a-t-on $P_1 \in \mathcal{P}$?

Solution : *Oui.*

□

3. Si P_1 est NP-complet, P_2 est-il NP-complet ?

Solution : *Non, car $P_2 \notin \mathcal{NP}$.*

□

4. Si P_2 est NP-complet, P_1 est-il NP-complet ? **Solution :** *Non.*

□

5. Si on connaît une transformation polynomiale de P_2 en P_1 , P_1 et P_2 sont-ils NP-complets ?

Solution : *Non.*

□

6. Si P_1 et P_2 sont NP-complets, existe-t-il une transformation polynomiale de P_2 en P_1 ?

Solution : *Oui.*

□

7. Si $P_1 \in \mathcal{NP}$, P_2 est-il NP-complet ?

Solution : *Non, cela n'indique rien sur la NP-complétude des problèmes.*

□

2 Quelques réductions

Montrer que les problèmes suivants sont NP-complets :

Question 2.1 Étant donnés $2n$ entiers a_1, a_2, \dots, a_{2n} , trouver un sous-ensemble $I \subset [1..2n]$ tel que $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$.

Solution : En partant d'une instance de 2-partition, si n est pair c'est ok, sinon on ajoute 3 elts $2S, 2S, 4S$ où $S = \sum a_i$. L'équivalence est alors simple, on divise la nouvelle instance en deux sous-ensembles de somme $4S + S/2$. \square

Question 2.2 Étant donnés $2n$ entiers a_1, a_2, \dots, a_{2n} , trouver un sous-ensemble $I \subset [1..2n]$ tel que $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$ et $|I| = n$.

Solution : En partant d'une instance \mathcal{I} de 2-Partition, on crée l'instance \mathcal{I}' avec $2n$ entiers $a_1 + 1, a_2 + 1, \dots, a_n + 1, 1, 1, \dots, 1$. Alors si on a une solution I pour \mathcal{I} , on obtient $S/2 + n$ avec les $(a_i + 1), i \in I$ et $n - |I|$ éléments parmi les 1, ce qui fait un total de n éléments. La somme des n éléments restant est également $S/2 + n$, donc on a une solution à \mathcal{I}' . Réciproquement, toute solution à \mathcal{I}' est de la forme $\sum_{i \in I} (a_i + 1) + n - |I|$ car on doit avoir n éléments dans la solution, et $0 < |I| < n$. On en déduit une solution à 2-Partition. \square

Question 2.3 Étant donnés $2n$ entiers a_1, a_2, \dots, a_{2n} , trouver un sous-ensemble $I \subset [1..2n]$ tel que $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$, et pour $j = 1..n$, soit on a $a_{2j-1} \in I$ et $a_{2j} \notin I$, soit le contraire.

Solution : A partir de 2-Partition-Eq, on crée la suite avec $4n$ entiers $a_1, B, a_2, B, \dots, a_{2n}, B$, et $B > S$. A partir de là, l'équivalence est simple, on a forcément n éléments B dans chaque sous-ensemble pour obtenir une somme $nB + S/2$. \square

Question 2.4 Étant donnés un graphe $G = (V, E)$ et un entier $K \geq 3$, déterminer si G contient une roue de taille K , i.e. un ensemble de $K + 1$ sommets w, v_1, v_2, \dots, v_K tels que $(v_i, v_{i+1}) \in E$ pour $1 \leq i < K$, $(v_K, v_1) \in E$ et $(v_i, w) \in E$ pour $1 \leq i \leq K$ (w est le centre de la roue).

Solution : en partant de Cycle Hamiltonien (instance \mathcal{I} graphe $g = (V, E)$), on construit un nouveau graphe en rajoutant un sommet w relié à tous les autres. On demande si il existe une roue de taille $|V|$ dans le nouveau graphe (instance \mathcal{I}' deroue). si \mathcal{I} possède un cycle hamiltonien, alors on a une roue de centre w . Réciproquement si \mathcal{I}' possède une roue, alors soit w' le centre de la roue. On peut facilement voir qu'il existe également une roue de centre w car w' est relié à tous les sommets par définition du centre de la roue, et donc il peut facilement remplacer w' dans le cycle. On a donc un cycle hamiltonien dans le graphe initial de l'instance \mathcal{I} . \square

Question 2.5 Étant donnés un graphe $G = (V, E)$ et un entier $K \geq 3$, déterminer si G contient un dominateur de cardinal K , i.e. un sous-ensemble $D \subset V$ de cardinal K tel que pour tout sommet $u \in V \setminus D$, il existe $u \in D$ avec $(u, v) \in E$

Solution : Réduction à partir de vertex-cover en rajoutant dans la nouvelle instance in sommet uv pour chaque arête (u, v) que l'on relie à u et à v . \square

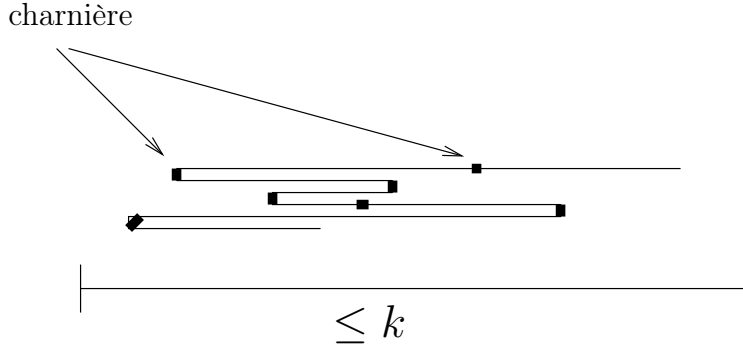


FIG. 1: Pliage des baguettes

Question 2.6 (*difficile*) Etant données n baguettes rigides de longueur entières a_1, a_2, \dots, a_n , pouvant être reliées dans cet ordre bout-à-bout par des charnières, et étant donné un entier k , assembler toutes les baguettes de manière qu'en repliant la chaîne obtenue la longueur totale ne dépasse pas k (voir la Figure 1).

Indication : réduire à partir de 2-Partition.

Solution : [Kozen] A partir de 2-partition, on crée les baguettes $S, S/2, a_1, a_2, \dots, a_n, S/2, S$. On fait un pliage qui ne dépasse pas la longueur S . La première baguette de longueur S part de 0 et va vers la droite, la suivante vers la gauche, on est en $S/2$. Les a_i vont à droite si $a_i \in I$ et à gauche sinon, et on doit avoir une somme égale pour revenir au point $S/2$ et terminer le pliage en allant à droite puis à gauche. Les deux sens de l'équivalence se montrent simplement à partir de cette observation.

□

3 Recherche d'éléments

On veut une structure de données qui permette la recherche (d'un élément), et l'insertion (d'un nouvel élément) de manière efficace.

Question 3.1 Quels sont les coûts de la recherche et de l'insertion si on utilise un tableau trié de taille n ?

Solution : Recherche en $\log(n)$, insertion en $O(n)$. □

On propose la solution suivante : soit $k = \lceil \log(n + 1) \rceil$ et soit $(n_{k-1}, n_{k-2}, \dots, n_0)$ la représentation binaire de n . On a k tableaux triés A_0, A_1, \dots, A_{k-1} , et la taille de A_i est 2^i pour tout i . Le tableau A_i est plein si $n_i = 1$ et vide si $n_i = 0$. Ainsi le nombre total d'éléments stockés dans les k tableaux est $\sum_{i=0}^{k-1} n_i 2^i = n$. Noter que chaque tableau est trié mais qu'il n'y a aucune relation entre les éléments de deux tableaux.

Question 3.2 Expliquer comment faire une recherche dans cette structure, et donner le coût au pire cas.

Solution : On peut effectuer une recherche dans cette structure de données en répétant la recherche dans chacun des sous-tableaux. La recherche dans un sous-tableau de taille m se fera en $\log(m)$ puisque les sous-tableaux sont triés. au pire cas (tout les tableaux sont pleins et la recherche est infructueuse), le temps total sera :

$$\begin{aligned}
 T(n) &= \Theta(\log(2^{k-1} + \log(2^{k-2} + \dots + \log(2^1) + \log(2^0))) \\
 &= \Theta((k-1) + (k-2) + \dots + 1 + 0) \\
 &= \Theta(k(k-1)/2) \\
 &= \Theta(\lceil \log(n+1) \rceil (\lceil \log(n+1) \rceil - 1)/2) \\
 &= \Theta(\log^2(n))
 \end{aligned}$$

□

Question 3.3 Expliquer comment faire une insertion dans cette structure, et donner le coût au pire cas et en analyse amortie.

Solution : On commence par créer un nouveaux tableau trié contenant uniquement l'élément à insérer. Si le tableau A_0 est vide, on remplace A_0 par le nouveau tableau, sinon on crée un nouveau tableau trié de taille deux à partir de A_0 et du nouveau tableau. si A_1 est vide, on remplace A_1 par le nouveau tableau sinon etc. Au final, on obtient toujours un tableau A_i de taille 2^i . Au pire cas (les $k-2$ premiers tableaux sont pleins) Le temps de calcul traitement sera :

$$T(n) = 2(2^0 + 2^1 + \dots + 2^{k-2}) \quad (1)$$

$$= 2(2^{k-1} - 1) \quad (2)$$

$$= 2^k - 2 \quad (3)$$

$$= \Theta(n) \quad (4)$$

En utilisant la méthode de l'agrégat pour calculer le coût total d'une série de n insertion à partir d'une structure de données vide. Soit r la position du 0 le plus à gauche de la représentation binaire $(n_{k-1}, n_{k-2}, \dots, n_0)$ de n . $n_j = 1$ pour $j < r$. Le coût d'une insertion quand n éléments ont déjà été insérés est $\sum_{j=0}^{r-1} 2 \cdot 2^j = \mathcal{O}(2^r)$.

$r = 0$ une fois sur deux, $r = 1$ une fois sur quatre etc. et il y a au plus $\lceil n/2^r \rceil$ insertion pour chaque valeur de r . on peut donc borner le coût de n insertions par :

$$\mathcal{O} \left(\sum_{r=0}^{\lceil \log(n+1) \rceil} \left(\left\lceil \frac{n}{2^r} \right\rceil \right) 2^r \right) = \mathcal{O}(n \log(n))$$

Le coup amortie d'une insertion est donc $\log(n)$.

□

Question 3.4 Expliquer comment supprimer un élément.

Solution : pour supprimer l'élément x :

- Trouver de plus petit j tel que A_j est plein. soit y le plus petit élément de A_j .
- Trouver A_i , le tableau contenant x .
- Enlever x et le remplacer par y . A_j a désormais 2^{j-1} élément, On place alors le premier élément de A_j dans A_0 les deux suivant dans A_1 etc. et on marque A_j comme vide. Il n'y a pas besoin de trier les tableaux nouvellement créés.

□

4 Nuages de points

On considère n points dans un espace métrique (les distances entre les points satisfont l'inégalité triangulaire). On veut partitionner les points en k groupes de manière à minimiser le plus grand diamètre d'un groupe. Le diamètre d'un groupe est la distance maximale entre deux points de ce groupe. Noter que n et k sont fixés dans l'énoncé du problème.

Question 4.1 On suppose que le diamètre optimal d est connu. Trouver une 2-approximation pour le problème.

Solution : On propose l'algorithme suivant :

Entrées : \mathcal{S} //ensemble de points

Résultat : $\mathcal{P} = \{\emptyset\}$ //partition des points

début

pour i allant de 1 à k et $\mathcal{S} \neq \emptyset$ **faire**

 choisir au hasard un point $p \in \mathcal{S}$;

$C = \{p' \mid d(p, p') \leq d\}$;

$\mathcal{P} = \mathcal{P} \cup \{C\}$;

$\mathcal{S} \leftarrow \mathcal{S} \setminus C$;

fin

retourner \mathcal{P} ;

fin

L'espace étant métrique, le diamètre d'un groupe est au plus $2d$. De plus chaque point est exactement dans un groupe. En effet, par construction chaque point est au plus dans un groupe. De plus si il existe point qui n'est dans aucun groupe, il doit être à plus de d des k points à l'origine des groupes. cela implique que toute k -partition aura un diamètre supérieure à d , ce qui contredit l'optimalité de d . □

Question 4.2 On considère l'algorithme qui, k fois, choisit comme "centre" le point à distance maximale de tous les centres déjà choisis, puis alloue chaque point au centre le plus proche. En faisant le lien avec la question précédente, montrer que cet algorithme est une 2-approximation pour le problème.

Solution : Nous appellerons A l'algorithme précédent et B l'algorithme proposé dans cette question. À chaque itération l'algorithme A peut choisir soit le même point que B soit un autre point. Comme

A est une 2-approximation, si A et B choisissent les même ensemble de points, B sera également une 2-approximation.

Lemme 1. *Soit p le centre choisi par l'algorithme B à l'itération i. Soit l'algorithme A peut choisir le point p comme centre, soit l'algorithme B réalise déjà une 2-approximation.*

Preuve : *À une itération donnée, Si A ne peut pas choisir p, cela signifie que p est au plus à une distance d de tous les centre déjà choisi. Mais comme p est choisi par B, cela signifie que tous les points sont au plus à une distance d des centres déjà choisis. Cela implique que d'assigner chaque point au centre le plus proche donne une 2-approximation* □

5 Ordonancement à une machine

On a n tâches indépendantes à exécuter sur une seule machine. La durée de la tâche T_j est p_j , son coût est w_j . Pour tout j , p_j et w_j sont des entiers positifs. Dans un ordonnancement, la première tâche commence son exécution au top 0, et la tâche T_j finit son exécution au top C_j . La fonction objective est la minimisation de la somme pondérée des temps de fin d'exécution, i.e. $S = \sum_{j=1}^n w_j C_j$

Question 5.1 Exemple avec $n = 4$, $(p_1, p_2, p_3, p_4) = (2, 3, 1, 4)$, $(w_1, w_2, w_3, w_4) = (2, 3, 1, 4)$. Que vaut S si on exécute les tâches dans l'ordre (T_1, T_2, T_3, T_4) ? Peut-on faire mieux? Si oui, quel est l'optimal?

Solution : *TODO* □

Question 5.2 Dans le cas général, proposer un algorithme (polynomial) optimal.

Solution :

Règle de smith, ou retard moyen pondéré *On obtient un ordonnancement optimal en séquençant les tâches par ordre croissant des p_i/w_i . Cet ordre est appelé **WSPT** (Weighted Shortest Processing Time).*

Soient deux tâches T_i et T_j telles que $p_i/w_i > p_j/w_j$. Supposons qu'il existe un ordonnancement optimal S ne respectant pas l'ordre WSPT pour ces deux tâches (i.e. T_i est ordonnancée avant T_j). Considérons S' l'ordonnancement obtenu en inversant T_i et T_j . On notera φ la valeur de la fonction objective pour l'ordonnancement S et φ' la valeur de cette même fonction pour S' . On a :

$$\varphi' - \varphi = w_i(C'_i - C_i) + w_j(C'_j - C_j) \tag{5}$$

$$= w_i p_j - w_j p_i \tag{6}$$

Or $p_i/w_i > p_j/w_j$ donc $p_i w_j > p_j w_i$ et $\varphi' - \varphi < 0$. L'ordonnancement S' est ainsi meilleur que S ce qui est contradictoire avec l'optimalité de S .

□

On rajoute des temps de disponibilité : T_j ne peut pas commencer avant le top r_j .

Question 5.3 Traiter l'exemple précédent avec $(r_1, r_2, r_3, r_4) = (6, 2, 1, 0)$. Que vaut S si on exécute les tâches dans l'ordre (T_1, T_2, T_3, T_4) ? Peut-on faire mieux ?

Solution : *TODO* □

Question 5.4 (*difficile*) On suppose $w_j = 1$ pour tout j . Quelle serait la stratégie optimale si on avait le droit de morceler l'exécution des tâches en plusieurs tranches de durée entière ?

Solution : *Si on permet le morcellement des tâches, La stratégie d'ordonnement optimale sera l'ordre **SRPT** Shortest Remaining Processing Time. À chaque top on choisi parmi les tâches ordonnables la tâche ayant le temps de calcul résiduel minimal.* □

Question 5.5 (*très difficile*) Toujours avec $w_j = 1$ pour tout j , en déduire un algorithme d'approximation de la solution optimale (on admettra que le problème est NP-complet).

Solution : *On va montrer que l'ordonnement sans morcellement en fonction de leur temps de completion avec l'ordonnement avec morcellement est une 2-approximation de l'ordonnement non-préemptif avec temps de disponibilité.*

soit S' l'ordonnement optimal des tâches suivant leurs temps de completion dans l'ordonnement avec morcellement S . On notera C'_j et C_j les temps de completion de la tâche j respectivement dans S' et S . j doit être ordonnancé avant C_j dans S' . Les tâches précédent j sont les tâches qui ont un temps de completion dans S inférieur à C_j . La somme des durées des tâches précédant j est inférieur à C_j , on en déduit $C'_j \leq C_j + p_j \leq 2C_j$.

□