Algorithmique et Programmation

TD n° 1 : Introduction

École normale supérieure – Département d'informatique
algoL3@di.ens.fr

2013-2014

Exercice 1. La salle `INFO 4` est très demandée ... Tous les départements de l'École veulent y faire cours et tous envoient à l'administration des requêtes de la forme $[s_i, t_i[$ (avec $i \in \{1, \ldots, n\}$), pour utiliser la salle entre l'heure s_i et l'heure t_i (on peut supposer que les heures sont exprimées en secondes depuis le 1er janvier 1970). L'administration cherche à rejeter le moins de requêtes possibles.

Nous proposons d'utiliser un algorithme glouton pour résoudre ce problème. L'idée consiste à trier les intervalles $[s_i, t_i[$ selon un ordre bien choisi, à accepter la première requête, à supprimer les requêtes qui entrent en conflit et à répéter récursivement. Pour chacun des ordres suivants, déterminer si l'algorithme glouton associé est optimal (par une preuve ou un contre-exemple).

1. Trier les cours par ordre croissant de leur heure de commencement s_i.
2. Trier les cours par ordre croissant de leur heure de fin t_i.
3. Trier les cours par ordre croissant de leur durée $t_i - s_i$.
4. Trier les cours par ordre croissant du nombre de cours c_i avec lesquels ils entrent en conflit ($i.e.$ $c_i = \#\{j \in \{1, \ldots, n\} [s_i, t_i[\cap [s_j, t_j[\neq \emptyset\}$).

Exercice 2. On dispose d'un stock de boules en verre identiques. Le problème est de déterminer à partir de quel étage d'un immeuble les boules en verre se cassent si on les jette par la fenêtre. Vous êtes dans un immeuble à n étages (numérotés de 1 à n) et vous disposez de k boules.

Il n'y a qu'une seule opération possible pour tester si la hauteur d'un étage est fatale : jeter une boule par la fenêtre. Si elle ne se casse pas, vous pouvez la réutiliser ensuite pour d'autres tests, sinon vous ne pouvez plus.

Vous devez proposer un algorithme pour trouver la hauteur à partir de laquelle un lancer est fatal (renvoyer `erreur` si la boule résiste encore au n-ième étage). La complexité de l'algorithme est le nombre de lancers nécessaires pour déterminer cette hauteur.

1. Proposer un algorithme pour $k = 1$ et donner sa complexité.
2. Si $k \geq \lceil \log_2(n) \rceil$, proposer un algorithme de complexité $O(\log(n))$.
3. Si $k < \lceil \log_2(n) \rceil$, proposer un algorithme de complexité $O\left(k + \frac{n}{2^{k-1}}\right)$. Donner la complexité de cet algorithme lorsque $k = 2$.
4. Proposer un algorithme de complexité $o(n)$ lorsque $k = 2$.
5. Proposer un algorithme qui demande $\sqrt{2n}$ lancers lorsque $k = 2$.

Exercice 3. La distance de Levenshtein entre deux mots est le nombre minimum d'opérations nécessaires pour transformer un mot en l'autre, où une opération est une insertion, une suppression ou une substitution d'une lettre. Proposer un algorithme qui calcule la distance de Levenshtein entre deux mots.

Exercice 4. Nous considérons le problème d'enpaquetage (*bin-packing*) étudié en cours. Étant donnés un entier n et un ensemble $U = (u_1, \cdots, u_n)$ de n objets, auxquels est associée une taille $s(u_i) \in [0, 1]$, trouver une partition de U en k sous-ensemble X de U, k étant choisi minimum et chaque ensemble X de la partition tel que

$$\sum_{i \in X} s(u_i) \leq 1.$$

1. L'algorithme $NextFit$ est le suivant :

Algorithme Next Fit
Pour chaque objet u_ℓ
    S'il l'objet tient dans la dernière boîte utilisée
    Alors mettre u_ℓ dans cette boîte
    Sinon fermer la boîte en cours et mettre u_ℓ dans une nouvelle boîte

Pour une instance I du problème *bin-packing*, notons $OPT(I)$ l'optimum correspondant et $NF(I)$ le nombre de boîtes ouvertes par l'algorithme *Next Fit*.

(a) Montrer que
$$NF(I) \leq 2 \cdot OPT(I) - 1,$$
et que cette inégalité est optimale.

(b) Supposons que pour tout $i \in \{1, \ldots, n\}$, nous avons $s(u_i) < \gamma$. Montrer que
$$NF(I) \leq \left\lceil \frac{OPT(I)}{1-\gamma} \right\rceil,$$

2. L'algorithme *First Fit Decreasing* est le suivant :

Algorithme First Fit Decreasing
Trier les u_ℓ par ordre décroissant de taille $s(u_\ell)$
Pour chaque objet u_ℓ
    Considérer les boîtes par ordre d'ouverture
    S'il existe une boîte dans laquelle u_ℓ tiendrait
    Alors mettre u_ℓ dans la première telle boîte
    Sinon mettre u_ℓ dans une nouvelle boîte

Pour une instance I du problème *bin-packing*, notons $FFD(I)$ le nombre de boîtes ouvertes par l'algorithme *First Fit Decreasing*

(a) Supposons que pour tout $i \in \{1, \ldots, n\}$, nous avons $s(u_i) \in]1/3, 2/3[$. Montrer que
$$FFD(I) = OPT(I).$$

(b) En déduire que si pour tout $i \in \{1, \ldots, n\}$, nous avons $s(u_i) \in]1/3, 1]$, alors
$$FFD(I) = OPT(I).$$

(c) En déduire que pour une instance I du problème *bin-packing*, nous avons
$$FFD(I) \leq \frac{3}{2} \cdot OPT(I) + 1.$$