Algorithmique et Programmation

TD n° 2 : Tris

École normale supérieure – Département d'informatique

2016-2017

Exercice 1. *k*-IÈME ÉLÉMENT D'UN TABLEAU

Nous allons utiliser une variante du tri rapide pour déterminer le k-ième élément d'un ensemble de n éléments. Nous supposons que les n clés sont distinctes deux à deux.

1. Décrire une procédure SÉLECTIONNER utilisant une « procédure de pivot » (comme dans l'algorithme de tri rapide) qui détermine le k-ième élément du tableau.
2. Montrer que sans hypothèse particulière sur l'algorithme de pivot, la fonction SÉLECTIONNER peut réaliser $O(n^2)$ comparaisons.
3. Montrer que si l'algorithme de pivot garantit que la taille du sous-tableau de l'appel récursif ne dépasse pas αn où $\alpha < 1$, la complexité en nombre de comparaisons de la fonction SÉLECTIONNER est $O(n)$.

On considère l'algorithme de choix du pivot suivant :

- Découper le tableau en $n/3$ blocs $\{B_1, \ldots, B_{n/3}\}$ de trois éléments ;
- Déterminer les éléments médians m_k des B_k, $k \in \{1, \ldots, n/3\}$;
- Utiliser l'algorithme récursivement pour déterminer l'élément médian p de la liste $m_1, \ldots, m_{n/3}$. Déterminer le rang de p dans le tableau, puis utiliser l'algorithme récursivement sur une partie du tableau.

4. Montrer que le pivot choisi est strictement supérieur à au moins $n/3 - O(1)$ éléments du tableau et est inférieur ou égal à au moins $n/3 + O(1)$ éléments du tableau.
5. Donner la complexité de la fonction SÉLECTIONNER.

Exercice 2. 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 sous forme $[d, f[$ pour utiliser la salle du temps d au temps f (on peut supposer que d et f sont toujours des entiers). L'administration cherche à rejeter le moins de requêtes possibles parmi toutes les requêtes reçues $\mathcal{I} = \{[d_1, f_1[, [d_2, f_2[, [d_3, f_3[, \ldots, [d_n, f_n[\}$.

Nous proposons d'utiliser un algorithme glouton pour résoudre ce problème. L'idée consiste à trier les intervalles de $\mathcal{I}$ selon un ordre bien choisi, à accepter la première requête, à supprimer les requêtes qui entrent en conflit et à répéter ces trois étapes jusqu'à ce qu'il reste aucun intervalle. 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 d_i.
2. Trier les cours par ordre croissant de leur heure de fin f_i.
3. Trier les cours par ordre croissant de leur durée $f_i - d_i$.
4. Trier les cours par ordre croissant du nombre de cours c_i avec lesquels ils entrent en conflit (*i.e.* $c_i = \#\{[d, f[\in \mathcal{I} \mid [d_i, f_i[\cap [d, f[\neq \emptyset\}$).

Exercice 3. 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 une 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 4. MAXIMUM & MINIMUM

1. Écrire un algorithme naïf qui calcule le minimum et le maximum sur un tableau de n éléments et donner un équivalent du nombre de comparaisons au pire des cas.
2. Est-il possible de réduire le nombre de comparaisons à faire ? Décrire un algorithme avec moins de comparaisons au pire des cas.
3. Montrer une borne inférieure de n comparaisons.
4. Montrer une meilleure borne inférieure sur le nombre de comparaisons à effectuer.

   Indication : On pourra utiliser la méthode de l'*adversaire*.

   Soit $\mathcal{A}$ un algorithme qui trouve le maximum et le minimum. Donner une stratégie d'un adversaire qui choisit les réponses aux comparaisons de façon à obliger $\mathcal{A}$ à faire plus de comparaisons.

Exercice 5. CHOIX DE PIVOT

Analyser le temps de l'algorithme de sélection randomisé dans les cas suivant.

1. Le pivot est tiré aléatoirement uniformément dans le tableau.
2. Le pivot est choisi comme la médiane de trois éléments choisis uniformément aléatoirement dans le tableau.