Algorithmique TD2

Magistère Informatique 1ère Année

23 Septembre 2010

Exercice 1 On dispose d'un tableau d'entiers $T[1...n]$. On cherche à calculer en simultané le *min* et le *max* de ce tableau.

1. Proposer un algorithme naïf. Donner le nombre exact de comparaisons effectuées lors du calcul.
2. Une idée est de regrouper les éléments par paires afin de diminuer le nombre de comparaisons à effectuer. Proposer un algorithme suivant ce principe et donner le nombre exact de comparaisons effectuées par celui-ci.

Exercice 2 Prouver le théorème de la récurrence de partitions.

Soit $t : \mathbb{N} \to \mathbb{R}_+$ croissante à partir d'un certain rang et telle que:
$\exists n_0 > 0$, $\exists b > 1$, $\exists k \geq 0$, $\exists a, c, d > 0$ vérifiant

$$\begin{array}{rcl} t(n_0) & = & d \\ t(n) & = & at(n/b) + cn^k \quad \text{si } n > n_0 \text{ est une puissance de } b \end{array}$$

alors

$$t(n) = \begin{cases} \Theta(n^k) & \text{si } a < b^k \\ \Theta(n^k \log_b(n)) & \text{si } a = b^k \\ \Theta(n^{\log_b a}) & \text{si } a > b^k \end{cases}$$

Exercice 3 Soit $t : \mathbb{N} \to \mathbb{R}_+$ croissante à partir d'un certain rang et telle que:

$$\begin{array}{rcl} t(1) & = & d \\ t(n) & = & 2t(n/2) + log_2(n) \quad \text{si } n > 1 \text{ est une puissance de } 2 \end{array}$$

Montrer que $t(n) = \Theta(n)$.

Exercice 4 On représente un polynôme $p(x) = \sum_{i=0}^{n-1} a_i x^i$ par la liste de ses coefficients. Nous souhaitons écrire un algorithme effectuant la multiplication de deux polynômes $p(x)$ et $q(x)$ de degrés au plus $n-1$.

1. Proposer un algorithme naïf et donner sa complexité.
2. Observer qu'il est possible de séparer un polynôme en deux parties de tailles égales:

$$p(x) = p_1(x) + x^{\frac{n}{2}} p_2(x)$$
$$q(x) = q_1(x) + x^{\frac{n}{2}} q_2(x)$$

Utiliser cette observation pour écrire un algorithme de type "diviser pour régner" et donner sa complexité.

3. Améliorer l'algorithme de la question précédente en utilisant la relation:

$$ab + cd = (a + c)(b + d) - ad - cb$$

Donner la complexité du nouvel algorithme.

Exercice 5 On dispose d'un tableau d'entiers T de taille n, on cherche à déterminer le $k^{\text{ième}}$ élément de T c'est à dire l'élément qui se trouverait en position $k-1$ si on triait T. Un algorithme naïf est de commencer par trier le tableau puis de sélectionner l'élément voulu. La complexité d'un tel algorithme est en $O(nlog(n))$, le but de l'exercice est de montrer qu'il existe un algorithme en temps linéaire s'inspirant du tri rapide. On dispose d'une fonction $PIVOT(T, i, j)$ qui étant donné une table T et deux positions i et j, réorganise les éléments entre i et j autour d'un pivot k qu'elle retourne. On ne donne pour l'instant aucune précision sur la façon dont le pivot est choisi.

1. Utiliser cette fonction $PIVOT$ pour écrire un algorithme $SELECT$ traitant le problème.
2. Montrer que sans hypothèses sur la façon dont la fonction $PIVOT$ choisit l'élément, l'algorithme $SELECT$ peut effectuer jusqu'à $O(n^2)$ comparaisons.
3. Montrer que si le choix du pivot garantit que la taille des deux sous-tableaux construits autour du pivot ne dépasse pas αn où $\alpha < 1$, le nombre de comparaisons effectuées par l'algorithme $SELECT$ est $O(n)$.
4. On se concentre maintenant sur le choix du pivot. Considérer l'algorithme de choix du pivot suivant:
   (a) Découper le tableau en $\lfloor \frac{n}{5} \rfloor$ blocs $\{B_1, ..., B_{\lfloor \frac{n}{5} \rfloor}\}$, de cinq éléments. Les éléments restants (au plus 4) sont ignorés par la suite.
   (b) Déterminer les éléments médians des B_k, $\{m_1, ..., m_{\lfloor \frac{n}{5} \rfloor}\}$.
   (c) Utiliser la fonction $SELECT$ pour déterminer l'élément d'ordre $\lfloor \frac{n+5}{10} \rfloor$ de la liste $\{m_1, ..., m_{\lfloor \frac{n}{5} \rfloor}\}$ des médians.

1. Montrer que le pivot choisi est strictement supérieur à au moins $3 \lfloor \frac{n-5}{10} \rfloor$ éléments de T et inférieur ou égal à au moins $3 \lfloor \frac{n-5}{10} \rfloor$. En déduire que pour $n \geq 75$ la taille des sous-tableaux construits autour du pivot est au plus égale à $\frac{3n}{4}$.
2. Ecrire un algorithme $SELECT2$ qui résout le problème en $O(n)$ comparaisons.

Exercice 6 Soit $T[1...n]$ un tableau contenant n éléments. Un élément e dans T est dit majoritaire si T contient strictement plus de $\frac{n}{2}$ occurences de e. Le but de cet exercice est de proposer un algorithme qui recherche un élément majoritaire dans un tableau.

1. Proposer un algorithme naïf et donner sa complexité.
2. Proposer un algorithme de type "diviser pour régner" et donner sa complexité.
3. On présente (de façon ludique) un nouvel algorithme:

   On dispose de n balles de couleurs et on veut savoir si il y a une couleur majoritaire. On a à notre disposition une étagère, une boîte et une poubelle. L'algorithme procède comme suit:

   Partie 1 On examine les balles une par une afin de les ranger de façon ordonnée sur l'étagère.

- Si la balle courante est de couleur différente de celle qu'on vient de poser, on la pose sur l'étagère, on prend ensuite une balle dans la boîte (si elle n'est pas vide) et on la place à la suite. On passe ensuite à la balle suivante.
- Si la balle courante est de la même couleur que celle qu'on vient de poser, on la met dans la boîte. On passe ensuite à la balle suivante.

Partie 2 Soit $\mathcal{C}$ la couleur de la dernière balle posée sur l'étagère. On va maintenant considérer les balles une à une en commençant par les dernières posées sur l'étagère.

- Si la balle a pour couleur $\mathcal{C}$, on la jette à la poubelle ainsi que celle qui la précède.
- Si la balle n'est pas de couleur $\mathcal{C}$, on la jette à la poubelle ainsi qu'une balle prise dans la boîte. Si la boîte est vide on s'arrête et on conclut qu'il n'y a pas de couleur majoritaire.
- Si il ne reste plus qu'une balle on la met dans la boîte.

Une fois l'algorithme terminé, si la boîte n'est pas vide, on conclut que $\mathcal{C}$ est majoritaire. Dans le cas contraire on conclut qu'il n'y a pas de couleur majoritaire.

(a) Prouver que l'agorithme est correct.

(b) Donner sa complexité exacte.

Exercice 7 On souhaite implémenter une pile à l'aide d'un tableau. Si on fixe la taille du tableau a priori, on perd en efficacité pour deux raisons : lorsqu'il y a trop peu de données, beaucoup de place est perdue ; lorsqu'il y a trop de données, on ne peut pas tout rentrer dans le tableau. Pour résoudre ce problème, on utilise des piles dynamiques dont la taille peut changer de telle sorte qu'au moins la moitié du tableau est toujours rempli. L'opération qu'on veut implémenter est l'empilement : elle consiste à insérer l'élément à la première place libre lorsque le tableau n'est pas plein ; sinon, on crée un nouveau tableau de taille double du précédent, on copie dedans tous les éléments du premier tableau puis on empile finalement l'élément dedans.

1. Implémenter la fonction `empiler`(x).
2. Quelle est la complexité dans le pire cas de l'appel `empiler`(x)?
3. Trouver la complexité amortie de la fonction `empiler`(x) lorsqu'on commence par une pile vide.
4. On considère une procédure différente `empiler2`(x) qui consiste à augmenter la taille du tableau d'une constante k plutôt que de la doubler, lorsque le tableau est plein. Quelle est la complexité amortie de cette procdure ?
5. Même question dans le cas où on élève au carré la taille du tableau lorsqu'il est plein.
6. Supposons qu'on veuille également une opération de dépilement `depiler`(), qui consiste à effacer le dernier élément du tableau. Si la suppression du dernier élément résulte en un tableau à moitié plein, on crée un tableau de taille divisée par deux et on copie les éléments du tableau dedans. Quel est le coût amorti des opérations `empiler`(x) et `depiler`().