ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
Sections d'Informatique et de Systèmes de Communication

Série d'exercices 8 16 Novembre 2007

1. Sorting

a) Trouver un algorithme pour trier exactement 3 éléments et qui utilise en moyenne $\frac{16}{6}$ comparaisons.

b) Supposons que nous avons un tableau de N éléments avec $a[i].key = N - i$ pour tout $i = 0, \ldots, N-1$ (c'est-à-dire que a est trié dans l'ordre inverse, de façon décroissante). Si nous trions ce tableau avec BubbleSort, quel sera le nombre exact d'échanges nécessaires ?

c) Supposons que nous avons un tableau de N éléments $a[0], \ldots, a[N-1]$. Soit $(p_0, \ldots, p_{N-1})$ la permutation de $\{0, \ldots, N-1\}$ telle que

$$a[p_0].key \leq \ldots \leq a[p_{N-1}].key.$$

Montrer que si on veut trier a avec BubbleSort, il faudra au moins q passages, où

$$q = \max_{0 \leq i \leq N-1} (p_i - i).$$

2. Shell Sort

Dans cet exercice, nous supposons qu'un échange entre deux éléments x et y nécessite 3 mouvements (x vers $temp$, y vers x, $temp$ vers y), où $temp$ est un élément utiliser temporairement pour réaliser le dit échange.

a) Utiliser l'algorithme ShellSort avec les incréments $2, 1$ pour ordonner la suite de clés $(4, 3, 2, 6, 1, 5)$. Combien d'échanges ont été utilisés ?

b) Montrer que si une suite de longueur N est 2-triée et 3-triée alors on peut la trier en effectuant au plus $N - 1$ échanges.

3. Tri par échanges adjacents

Les algorithmes de tri élémentaires se réalisent souvent en échangeant uniquement des éléments adjacents (par exemple Bubble sort). Considérons la version alternative de InsertionSort suivante :

Call : InsertionSort(a)
Input: Suite a d'objets avec des clés étant des entiers.
Output: Transformation de a de sorte à avoir $a[i].key \leq a[i+1].key$ pour $0 \leq i < N-1$

```
1: for i = 1, ..., N − 1 do
2:     j ← i − 1
3:     while a[j].key > a[j + 1].key et j ≥ 0 do
4:         Echanger a[j + 1] et a[j]
5:         j ← j − 1
```

a) Comparer le nombre d'échanges qu'effectue cet algorithme au nombre de mouvements que fait la version de InsertionSort du cours : vérifier que si un échange est réalisé en effectuant trois mouvements, alors l'algorithme ci-dessus effectue au plus trois fois le nombre de mouvements que l'algorithme du cours.

b) Soit σ une permutation aléatoire des éléments $\{1, \ldots, N\}$. Montrer que $\forall i \in \{1, \ldots, N\}$ et $\forall j \in \{0, \ldots, \lfloor N/2 \rfloor\}$ on a :

$$P(|\sigma(i) - i| = j) \geq \frac{1}{N}$$

($|\sigma(i)-i|$ représente la distance entre la position de i dans $(1, \ldots, N)$ et sa nouvelle position dans la permutation $(\sigma(1), \ldots, \sigma(N))$).

c) En déduire que la moyenne sur tous les σ de $|\sigma(i) - i|$ est supérieure ou égale à $\frac{N}{9}$, pour tout $i \in \{1, \ldots, N\}$ fixé, et N assez grand.

d) Conclure du point précédent que tout algorithme de tri qui se limite à échanger des éléments adjacents est $\Omega(N^2)$ en moyenne.

4. Dégénérescence de QuickSort

Rappelons que QuickSort nécessite en moyenne $O(N \log N)$ comparaisons si la permutation des clés est aléatoire et uniformément choisie. Néanmoins, il faut à QuickSort $\Omega(N^2)$ comparaisons dans le pire des cas.

Intuitivement, QuickSort aura une meilleure performance si le pivot choisi est plus proche de la médiane de la suite (pour qu'elle soit coupée en deux sous-suites de même taille lors de l'appel récursif). Par contre, si le pivot se trouve parmi les plus grands (ou plus petits) éléments de la suite, il faudra faire plus d'appels récursifs, et donc plus de comparaisons. Le choix du pivot est donc crucial.

a) Si on choisit toujours comme pivot le dernier élément de la suite, montrer que pour tout N on peut trouver une suite de N éléments pour laquelle QuickSort nécessite $\Omega(N^2)$ comparaisons.

b) Considérons la stratégie suivante : on prend comme pivot toujours la médiane des éléments au début, au milieu et à la fin. Puis on échange cet élément avec le dernier élément de la suite (si nécessaire), et on procède ensuite comme dans le cours. (C'est la *3-median strategy*.) Montrer que pour tout N on peut trouver une suite de N éléments pour laquelle QuickSort nécessite $\Omega(N^2)$ comparaisons.

5. Priority Queues

a) Lesquelles des permutations de $\{1, 2, 3, 4, 5\}$ résulteront, après transformation en heap par BottomUpHeapCreate, en le tableau contenant $5, 3, 4, 1, 2$, dans cet ordre ?

b) Le crible d'Erastosthène est la procédure suivante pour trouver tous les nombres premiers de 1 à N :

   (a) Écrire une liste des nombres $2, 3, 4, \ldots, N$.

   (b) Prendre le plus petit nombre de la liste. L'enlever de la liste : C'est un premier. Biffer tous ses multiples de la liste.

   (c) Répéter l'étape précédente jusqu'à ce qu'il ne reste plus rien à faire.

Donner un algorithme efficace qui utilise un heap pour créer un tableau des N premiers nombres premiers, similaire au crible d' Erastosthène. Essayer d'économiser la mémoire vive afin d'obtenir un algorithme qui a besoin de moins de mémoire vive que celui d'Erastosthène.

c) Donner un algorithme efficace pour insérer un élément dans un heap. (Le heap est représenté par un tableau.) Donner un algorithme efficace pour enlever un élément spécifié par sa position dans le tableau d'un heap.

6.Sac à dos 0/1 avec des poids rationnels

Cet exercice suggère que le problème du sac à dos est peut-être plus difficile qu'on pourrait le croire.

a) Donner le temps de parcours de l'algorithme sac à dos 0/1 du cours en fonction de W et du nombre n d'objets candidats.

b) Pour le problème du sac à dos 0/1 comme vu dans le cours, on avait toujours supposé que les poids ainsi que les valeurs étaient des entiers. L'algorithme, s'applique-t-il encore si les valeurs sont des rationnels ?

c) On aimerait maintenant modifier l'algorithme du sac à dos 0/1 pour qu'il fonctionne aussi avec des poids rationnels. Que faut-il changer pour pouvoir l'appliquer dans ce cas ? (*Indication :* il suffit de multiplier W ainsi que les poids w_i par une valeur appropriée.)

d) Montrer qu'avec le changement de la question c), il est possible de trouver des poids $w_i = p_i/q_i$, $i = 1, \ldots, n$ (où $p_i, q_i \in \mathbb{N}$) tels que le W de l'algorithme modifié sera alors exponentiel en fonction de

$$\text{length}(w_1, \ldots, w_n) := \sum_{i=1}^{n} \log(p_i) + \log(q_i).$$

En déduire que l'algorithme aura alors un temps de parcours exponentiel en fonction de $\text{length}(w_1, \ldots, w_n)$ dans le pire des cas.

e) Expliquer l'intérêt de la fonction $\text{length}(w_1, \ldots, w_n)$. En particulier, pourqoui s'intéresse-t-on au temps de parcours en fonction de $\text{length}(w_1, \ldots, w_n)$.