ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Sections d'Informatique et de Systèmes de Communication

Série d'exercices 9 21 Nov. 2011

1. *Application de* QuickSort *et* HeapSort

Voici une suite de 30 nombres:

$$23, 29, 84, 15, 58, 19, 81, 17, 48, 15, 36, 49, 91, 26, 89,$$
$$22, 63, 57, 33, 10, 50, 56, 85, 4, 10, 63, 1, 72, 10, 48.$$

a) Trier cette suite avec QuickSort (en prenant toujours comme pivot le dernier élément, comme dans le cours). S'il vous reste des sous-suites de ≤ 5 éléments à trier, vous pouvez le faire à la main.

b) Trier les quinze premiers éléments ci-dessus avec HeapSort.

2. *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.

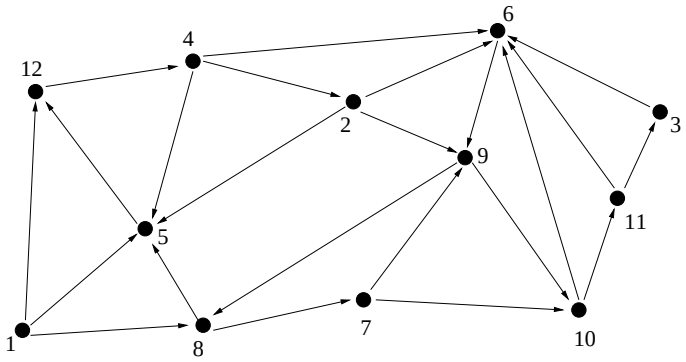
Intuitivement, QuickSort aura une meilleure performance si le pivot choisi est plus proche de la médiane de la suite (pour qu'elle soit coupée en deux sous-suites de même taille lors de l'appel récursif). Par contre, si le pivot se trouve parmi les plus grands (ou plus petits) éléments de la suite, il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

3. *Traverser des graphes*

Voici un graphe orienté:



a) Parcourir ce graphe avec la méthode DepthFirstSearch, en commençant au sommet 1.

b) Parcourir ce graphe avec la méthode BreadthFirstSearch, en commençant au sommet 1.

c) Modifier l'algorithme BFS pour qu'il retourne pour chaque sommet sa distance par rapport à 1.

d) Modifiez l'algorithme AbstractTraversal pour qu'il compte aussi le nombre de sommets qui sont atteignables depuis 1.

4. 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.