université de BORDEAUX

textscAnnée universitaire 2016/2017
Session 1 d'automne CPBX AN2

Parcours / Étape : ... Code UE : 4TBX311
Épreuve : 4TBX311EX Informatique
Date : 19/01/2017 Heure : 11h30 Durée : 1h30
Documents non autorisés
Épreuve de M. Dorbec Paul

Collège Sciences et Technologie

1 Chercher le k-ième plus grand

Dans cette partie, on s'intéresse à trouver le k-ième élément le plus grand d'une liste de n éléments, de façon la plus rapide possible. On exprimera les complexités en fonction de n et de k, en ordre de grandeur du nombre de comparaisons.

Une première méthode consiste à trier la liste par ordre croissant et à en piocher le $(n-k)$-ième élément (k-ième en partant de la fin).

1. Quelle est la complexité de cette méthode (en choisissant un bon tri) ?

Avec certains tris, on peut ne faire que le début du tri, pour s'arrêter dès qu'on a l'information voulue. Par exemple avec le tri par tas, on peut s'arrêter avant la fin du tri, en entassant tous les éléments, puis en extrayant la racine seulement k fois.

2. Quelle est la complexité de cette méthode ?
3. Proposez une méthode similaire en adaptant un tri à bulle ? Quelle est sa complexité ?
4. Y a-t-il un autre tri qui donne presque le même résultat que le tri à bulle ?
5. De toutes les méthodes vues, quelle est la meilleure quand $k = 3$? Quand $k = \frac{n}{2}$ (calcul de la médiane) ?

1.1 En s'inspirant du quicksort

En s'inspirant du tri rapide, on peut faire mieux en moyenne que les approches précédentes. On propose la méthode suivante : Dans la liste, on choisit un pivot aléatoirement, et on partitionne les valeurs autour du pivot : les plus grandes et les plus petites. Cela permet de déduire la position du pivot. Si le pivot est placé en position supérieure à k, on recommence sur la partition contenant les valeurs plus petites que le pivot, sinon on recommence sur les valeurs plus grandes, en mettant évidemment à jour la valeur de la cible k.

6. Soit $k = 4$, la liste $L = [12, 19, 7, 14, 33, 24, 8, 42, 4]$. Si le pivot choisi est 12, comment évoluent k et L pour l'appel récursif suivant (sans se préoccuper de l'ordre des éléments dans la liste L) ? Même question si au lieu de 12, le pivot avait été 24 ?
7. Quel argument montre que l'algorithme s'arrête ?
8. On note $C(n)$ le nombre de comparaisons effectuées par cet algorithme pour traiter un tableau de taille n. En supposant que le pivot tombe toujours au milieu du tableau, quelle est la relation de récurrence satisfaite par $C(n)$? (Notez que cette complexité ne dépend pas de k.)
9. En déduire la complexité de l'algorithme dans ce cas favorable.
10. Quelle serait une situation très défavorable au déroulement de l'algorithme ? Quelle est la complexité dans le pire cas de l'algorithme ?

1.2 Pseudo-médiane

Voici un algorithme de calcul de la pseudo médiane d'une liste L : dans la liste L, on regarde des groupes de cinq éléments consécutifs, et pour chacun, on calcule sa médiane (avec une méthode quelconque), que l'on place par exemple dans une nouvelle liste. Ensuite, on calcule la médiane de la liste des médianes, qui est notre pseudo-médiane.

11. Quel est le nombre d'éléments garantis plus petits que la pseudo-médiane ? Quel est le nombre maximum d'éléments plus petits que la pseudo-médiane ?

Notre intention maintenant est d'utiliser la pseudo-médiane comme pivot de l'algorithme précédent, ce qui nous donne un algorithme hybride. On note P le temps de calcul de la pseudo médiane sur une liste de taille n et $C(n)$ le temps de calcul, à l'aide de cet algorithme hybride, de l'élément de position k (donc en particulier de la médiane).

12. Exprimez la complexité $P(n)$ du calcul de la pseudo médiane en fonction de la complexité de l'algorithme hybride et de n.
13. Exprimez maintenant la complexité dans le pire cas de l'algorithme hybride en fonction de celle du calcul de la pseudo médiane.
14. Déduire la complexité de l'algorithme hybride.

2 Preuve d'algorithme

Voici un algorithme de lièvre et la tortue, qui appelle une fonction f.

```
def algo(x):
  lievre=f(f(x))
  tortue=f(x)
  while lievre != tortue:
    lievre = f(f(lievre))
    tortue = f(tortue)
```

1. Montrer (avec un invariant de boucle) que si la fonction f est périodique sur l'entrée x, c'est à dire qu'il existe un entier k tel que pour tout $i > 0$, $f^{i+k}(x) = f^i(x)$, alors le programme s'arête.

3 Programmation dynamique

Cette année, la mère Noël s'est exceptionnellement proposée pour aider le père Noël. Ils se sont donc réparti l'ensemble des cheminées à visiter pour déposer les cadeaux. Ils ont trouvé un algorithme malin pour définir leur trajet.

Le problème est le suivant : étant données n cheminées $x_1, \dots x_n$ et le pôle nord x_0, trouver deux chemins $p_0, p_1, \dots, p_{n_p}$ et $m_0, m_1, \dots, m_{n_m}$ avec les contraintes suivantes :

— ils partent tous les deux du pôle nord donc $p_0 = m_0 = x_0$.

— toutes les cheminées sont visitées : $\{x_1, \dots, x_n\} = \{p_1, \dots, p_{n_p}\} \cup \{m_1, \dots, m_{n_m}\}$

— les chemins minimisent la distance totale parcourue

$$\sum_{i=1}^{n_p} dist(p_{i-1}, p_i) + \sum_{j=1}^{n_m} dist(m_{j-1}, m_j)$$

où $dist(x_i, x_j)$ désigne la distance entre x_i et x_j.

Le problème est en fait légèrement simplifié par le fait que le père Noël a l'habitude de voyager d'est en ouest pour profiter du décalage horaire [1]. Ceci implique que les cheminées sont visitées précisément dans l'ordre de leurs indices, c'est à dire que si la même personne visite les cheminées x_i et x_j et que $i < j$, alors la cheminée x_i est visitée avant x_j dans le parcours. En particulier, en supposant que le père Noël a visité en dernier la cheminée x_i et la mère Noël la cheminée x_j, la prochaine cheminée à visiter est la cheminée x_k avec $k = \max(i, j) + 1$.

1. et être ainsi à minuit dans toutes les maisons !

Nous considérons une famille de sous problèmes du problème principal. On désigne par $\mathcal{L}(i, j, [k..n])$ la longueur de la solution optimale du sous problème consistant à visiter les cheminées de x_k à x_n et où le père Noël et la mère Noël partent de la cheminée x_i et x_j respectivement. En particulier, $\mathcal{L}(0, 0, [1..n])$ est la longueur de la solution optimale du problème initial.

1. Exprimez $\mathcal{L}(i, j, [k..n])$ en fonction des $\mathcal{L}(i', j', [k+1..n])$ (choisissez pour i' et j' des valeurs pertinentes) et des distances utiles entre les paires de sommets (on note $d(x_a, x_b)$ la distance entre x_a et x_b).
2. À l'aide de cette formule de récurrence, proposez un algorithme de programmation dynamique qui permet de trouver la longueur optimale des chemins pour le problème posé.
3. Quelle est la complexité de cet algorithme (en nombre d'utilisations de distance $d(x_a, x_b)$) ?
4. Comment utiliser les résultats de l'algorithme précédent pour trouver les deux chemins optimaux $p_0, p_1, \ldots, p_{n_p}$ et $m_0, m_1, \ldots, m_{n_m}$? Quelle est la complexité de cette étape ?