TD1

13 septembre 2013

`Exercice 1` — `$k$-ème élément d'un tableau`

1. Adapter l'algorithme de tri rapide pour trouver le k-ème élément d'un tableau.

2. Évaluer sa complexité dans le pire des cas.

3. Montrer que si le pivotage assure que l'appel récursif se fait toujours sur un tableau de taille inférieure à cn, avec $c < 1$, alors l'algorithme est en $O(n)$.

Pour sélectionner le pivot, on utilise l'algorithme suivant.
- On découpe le tableau en $\lfloor \frac{n}{5} \rfloor$ blocs de 5 éléments (on laisse de côté les éléments restants) ;
- on détermine les éléments médians m_k des blocs ci-dessus ;
- on détermine le $\lfloor \frac{n+5}{10} \rfloor$-ème éléments de la liste $m_1, \ldots, m_{\lfloor \frac{n}{5} \rfloor}$.

4. Montrer que le pivot est alors strictement supérieur à au moins $3\lfloor \frac{n+5}{10} \rfloor$ éléments de T et est inférieur ou égal à au moins $3\lfloor \frac{n+5}{10} \rfloor$ éléments.

En déduire un algorithme en $O(n)$.

Exercice 2 — Diviser pour régner - majorité

Un tableau $T[1, \ldots, n]$ a un élément majoritaire si au moins la moitié des éléments sont les mêmes. Le but est d'écrire un algorithme efficace pour dire si un tableau T a un élément majoritaire et si tel est le cas, de le donner.

Les éléments du tableau ne sont pas comparables : impossible de demander $T[i] > T[j]$ (pensez que les éléments du tableau sont des images). Mais on peut tester si $T[i]T[j]$ en temps constant.

1. Montrer comment résoudre ce problème en $O(n \log n)$. (si on divise T en deux sous-tableaux T_1 et T_2, et que vous connaissez les éléments majoritaires respectifs de T_1 et T_2, pouvez-vous retrouver l'élément majoritaire de T ?)

2. Pouvez-vous écrire un algorithme en temps linéaire ? (apparier les éléments du tableau, et supprimer les paires composés d'éléments différents. Que peut-on dire sur les éléments majoritaires ?)

Exercice 3 — Deux piles dans un tableau

1. Comment peut-on représenter deux piles par un seul tableau en assurant que, tant que le tableau n'est pas plein, il est possible d'ajouter un élément dans n'importe quelle pile ?

2. Proposer une implémentation des opérations de base des piles dans ce cas.

Exercice 4 — Parenthèses

On considère ici des séquence de parenthèses avec plusieurs symboles de parenthèses : des crochets, des accolades, des parenthèses, etc. Une séquence de parenthèses est correcte si et seulement si 1) chaque parenthèse ouverte est fermée ensuite par une parenthèse du même type et 2) une parenthèse n'est fermée que lorsque toutes les parenthèses ouvertes après elle ont été fermées. Par exemple la séquence $([]\{\big(()\big)\})$ est correcte mais les séquences $\{()[()]$ et $(\{)\}$ ne le sont pas. On souhaite vérifier si une séquence de parenthèses est correcte.

1. Proposer un type abstrait adapté à la résolution de ce problème.

2. Écrire un algorithme vérifiant qu'une séquence de parenthèses est correcte.

Exercice 5 — Quelques propriétés des arbres binaires

1. Montrer que le nombre maximal de nœuds à la profondeur k dans un arbre binaire est 2^k.

2. Soit un arbre binaire à n nœuds, montrer que le nombre de feuilles f de cet arbre est tel que $2f \leq n + 1$.

Soit la fonction suivante, qui parcourt un arbre binaire :

```
1: parcours A =
2:       si A est vide alors terminer
3:       sinon
                (...)
4:            parcours G(A)
                (...)
5:            parcours D(A)
                (...)
```

Si on ajoute un affichage de la racine de A à différentes positions dans cette fonction, on obtient la liste des sommets de l'arbre sur lequel on appelle la fonction :

- en ordre prefixe si l'affichage est fait entre la ligne 3 et la ligne 4,
- en ordre infixe s'il est fait entre les lignes 4 et 5,
- ou en ordre suffixe s'il est fait après la ligne 5.

3. Montrer que si on connaît la liste des nœuds d'un arbre binaire en ordre préfixe et en ordre infixe on peut reconstruire l'arbre. Est-ce toujours vrai si on connaît la liste des nœuds en ordre préfixe et en ordre suffixe ? En ordre infixe et en ordre suffixe ?

`Exercice 6` `Arbres PATRICIA (ou arbre radix)`

On rappelle le type abstrait 'Ensemble' dont le but est de représenter un ensemble dynamique avec les opérations : ajouter un élément, supprimer un élément et tester l'appartenance. Vous avez vu les tableaux, listes, tables de hachage et arbres binaires de recherche pour implémenter un ensemble. On se propose ici d'étudier une autre structure de données : les arbres PATRICIA (pour "Practical Algorithm To Retrieve Information Coded In Alphanumeric") adaptés pour les ensembles de mots.

Un arbre PATRICIA est l'arbre vide ou un arbre binaire dont les arcs a sont étiquetés par des mots m_a. Étant donné un nœud x et deux étiquettes d'arcs distincts partant x n'ont pas de préfixes communs non triviaux. Seuls les arcs qui arrivent sur des feuilles peuvent être étiquetées par le mot vide. Un mot m qui appartient à l'ensemble est représenté par une feuille f et on lit le mot m en concaténant les étiquettes des arcs qui relie la racine à f.

1. Ecrire les algorithmes correspondants aux opérations de test d'appartenance, ajout et suppression.

2. Discuter des avantages / inconvénients des arbres PATRICIA.

`Exercice 7` `Une file avec deux piles`

Il est possible d'implémenter une file à l'aide de deux piles (A et B) de la manière suivante : l'ajout des éléments se fait dans la pile A et leur retrait depuis la pile B. Lorsque l'on souhaite retirer un élément alors que la pile B est vide on déplace d'abord tous les éléments de la pile A vers la pile B.

1. Proposer une implémentation des opérations de manipulation d'une file selon ce principe.

2. Donner la complexité temporelle des différentes opérations.

3. On considère l'ajout et le retrait de n éléments de la file, dans n'importe quel ordre. Quelle est alors la complexité temporelle de toutes ces opérations ?

Exercice 8 — Diviser pour régner - choix

Supposons que l'on vous donne le choix entre :

- L'algorithme A qui résout les problèmes en les divisant en 5 sous-problèmes dont la taille est la moitié, qui récursivement résout chaque sous-problème, et qui combine les solutions en temps linéaire ;
- L'algorithme B qui résout les problèmes de taille n en résolvant récursivement deux sous-problèmes de taille $n-1$ et combinent les solutions en temps constant ;
- L'algorithme C qui résout les problèmes de taille n en les divisant en neuf sous-problèmes de taille $\frac{n}{3}$, qui résout récursivement chaque sous-problème, et combine les solutions en $O(n^2)$.

1. Quel algorithme choisissez-vous ?

Exercice 9 — Diviser pour régner - carré d'une matrice

1. Montrer qu'il n'y a besoin que de 5 multiplications pour calculer le carré d'une matrice de taille 2×2.

2. Pourquoi ne peut-on pas généraliser l'algorithme de Strassen avec ces 5 multiplications (au lieu de 7) à des matrices de taille $n \times n$.

3. En fait, calculer le carré d'une matrice n'est plus simple que calculer le produit de deux matrices. Montrer que si on peut calculer le carré d'une matrice $n \times n$ en $O(n^c)$, alors on peut multiplier deux matrices $n \times n$ en $O(n^c)$.

Exercice 10 — diviser pour régner - k élément de l'union de deux listes triées

1. On vous donne deux listes triées de taille m et n. Donne un algorithme qui calcule le k plus petit élément de l'union des deux listes en temps $O(\log m + \log n)$.

Exercice 11 diviser pour régner - 3SAT en temps linéaire

SAT est défini de la manière suivante : l'entrée est une formule de la logique propositionnelle en forme normale conjonctive et le but est de déterminer s'il existe une valuation (des valeurs vraies/fausses pour les variables) qui rend la formule vraie.

Ici, on considère des instances avec une propriété de localité (*) : s'il y a n variables propositionnelles dans la formule d'entrée, on les suppose numéroté de 1 à n de telle sorte que les numéros des variables qui apparaissent dans chaque clause ne différent que de 10 au plus.

1. Donner un algorithme linéaire pour résoudre le problème SAT restreint aux entrées ayant la propriété de localité (*).