ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Sections d'Informatique et de Systèmes de Communication

Série d'exercices 4 12 octobre 2007

1. Exponention rapide

Dans la série précédente, nous avons vu un algorithme pour calculer x^n en $\leq 2\log_2(n)$ multiplications. Dans cet exercice nous verrons que cet algorithme n'est pas optimal, mais aussi qu'il est nécessite au plus deux fois plus de multiplications que le meilleur algorithme d'une grande classe.

a) Une *chaîne d'additions pour* n est une suite finie $a_0, \ldots, a_r$ tel que $a_0 = 1$, $a_r = n$ et qu'on peut écrire $a_j = a_{k_j} + a_{l_j}$ avec $k_j, l_j \leq j-1$ pour tout $1 \leq j \leq r$.
Montrer que s'il existe une chaîne d'additions pour n de taille $r+1$ (i.e., une chaîne d'additions $a_0, \ldots, a_r$ avec $a_r = n$), alors on peut calculer x^n en au plus r multiplications.

b) Soit

$$\ell(n) := (\text{La taille de la plus courte chaîne d'additions pour } n) - 1.$$

Avec cette notation, le résultat de la dernière série implique que $\ell(n) \leq 2\log_2(n)$, puisque l'algorithme de l'exercice en question peut être modélisé avec des chaînes d'additions.
Montrer que si $m, n \in \mathbb{N}$, alors

$$\ell(mn) \leq \ell(m) + \ell(n).$$

Trouver une valeur de n pour laquelle l'algorithme de la série précédente n'est pas optimal.

c) Montrer que si $r = \ell(n)$, il existe une chaîne d'additions strictement croissante de taille $r+1$ pour n, i.e., il existe une chaîne d'additions $a_0, \ldots, a_r$ telle que

$$1 = a_0 < a_1 < \cdots < a_r = n.$$

d) Montrer par induction que $\ell(n) \geq \lceil \log_2(n) \rceil$ pour tout n.

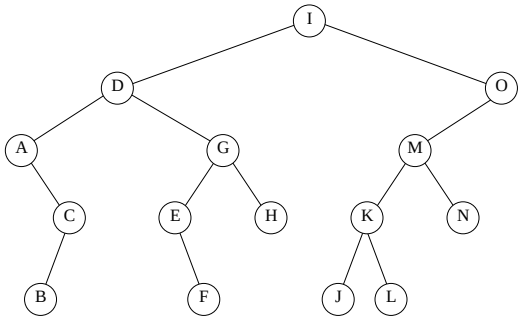
2.Stacks et files d'attente

a) Supposons que nous disposons uniquement de la structure de données stack. Est-il possible de réaliser une file d'attente en utilisant deux stacks ? Si oui, donner les algorithmes correspondants pour les opérations de la file d'attente.

b) Nous voulons de nouveau réaliser une file d'attente en utilisant deux stacks, mais avec la condition de plus que le temps de calcul des opérations est similaire à celui d'une implémentation classique de file d'attente.
Plus précisément, supposons que les opérations sur les stacks se font en temps $O(1)$ (i.e. ne dépendent pas de la taille du stack). Donner alors des algorithmes pour les opérations d'une file d'attente basée sur deux stacks qui ont la propriété que, si l'on commence avec une file vide, pour faire les m premières opérations de la file d'attente, on a besoin de $O(m)$ opérations du stack.

3. Arbres

a) Donner les suites de sommets obtenus en parcourant l'arbre binaire suivant en l'ordre preorder, inorder et postorder.



b) Vérifier qu'il s'agit bien d'un arbre de recherche binaire pour l'ordre alphabétique.

c) Prouver qu'un arbre binaire est un arbre de recherche si et seulement si la suite des valeurs obtenue en prenant les clés dans un parcours inorder est croissante.

d) Effacer le sommet D de manière à préserver la structure d'arbre de recherche.

e) Écrire un algorithme pour effacer dans un arbre de recherche. Donner un argument qui justifie cet algorithme.

f) Soit T soit un arbre binaire de hauteur h. Montrer que T peut avoir au plus $2^{h+1} - 1$ sommets.

4. Jeu d'échecs

On considère un jeu d'échecs de taille $n \times n$. On a n dames, et on aimerait les placer de façon *compatible*, i.e., de façon à ce que aucune ligne ne soit occupée de deux dames, ni aucune colonne, ni aucune diagonale.

Le but de cet exercice est de développer une méthode qui utilise la téchnique du *backtracking* pour trouver les placements compatibles. (Vous avez brièvement vu le backtracking dans le cours dans l'exemple de l'odomètre.)

a) Supposons que vous avez placés m dames ($m < n$) de manière compatible, dans les lignes 1 à m dans l'échiquier. Notons $a[0]$ la position (colonne) de la dame dans la première ligne, $a[1]$ celle de la dame dans la deuxième ligne, etc.

Ecrire un algorithme qui calcule les positions compatibles dans la $m+1$-ème ligne, i.e., écrire un algorithme qui trouve toutes les positions dans la $m+1$-ème ligne dans lesquelles on peut placer une dame de façon à ce que le placement reste compatible.

b) Utiliser la solution du point précédent pour calculer toutes les placements valides de n dames dans l'échiquier de taille $n \times n$. (*Indication :* Vous aurez besoin d'un appel récursif ou au moins d'un stack.)

c) Montrer que le temps de parcours de cet algorithme est $O(\text{poly}(n) \cdot n!)$. Donner la compléxité de l'algorithme basique qui consiste simplement à tester toutes les placements de n dames pour compatibilité. Commenter.