Algorithmique TD7

Magistère Informatique 1ère Année

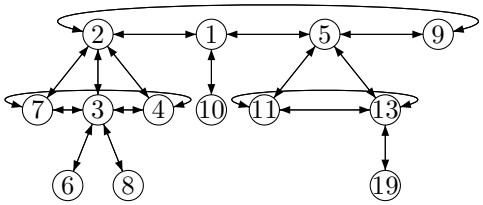
06 Novembre 2008

Exercice 1 Tas de Fibonacci

Un tas de Fibonacci est, comme un tas binomial, un ensemble d'arbres tournois (arbres dont la clé de chaque nœud est inférieure aux clés de ses fils). Un tas de Fibonacci T est implémenté comme suit. Le tas vide est noté NIL. Un nœud x contient :

- Un pointeur $x \cdot p$ vers le père de x (si x est une racine $x \cdot p = NIL$) ;
- Un pointeur $x \cdot f$ vers l'un des fils de x (si x est une feuille $x \cdot f = NIL$) ;
- Un pointeur $x \cdot g$ vers le frère gauche de x (si x n'a pas de frère gauche $x \cdot g$ pointe vers le frère le plus à droite de x) ;
- Un pointeur $x \cdot d$ vers le frère droit de x (si x n'a pas de frère droit $x \cdot d$ pointe vers le frère le plus à gauche de x) ;
- Un entier $x \cdot n$ qui contient le nombre de fils de x ;
- Une clé $x \cdot cle$.

Si x est fils unique, on a en particulier $x \cdot g = x \cdot d = x$. On donne en dessous un exemple de tas de Fibonacci.



L'ordre donné entre les racines et entre les fils d'un nœud par l'implémentation est arbitraire. On accède a un tas de Fibonacci, T grâce au pointeur $min(T)$ qui pointe vers la racine de clé minimale, si T est vide $min(T) = NIL$. On dispose également d'une variable $n(T)$ contenant le nombre de nœuds de T.

Pour $n \in \mathbb{N}$ on note $D(n)$ l'arité maximale dans les tas de Fibonacci de taille n que nos procédures garantissent (l'arité d'un nœud est son nombre de fils). En particulier on aura toujours $D(n) < n - 1$ et on montrera qu'on peut maintenir $D(n) = O(log(n))$.

1) Ecrire des procédures pour effectuer la recherche du nœud de clé minimale, l'insertion d'une clé (qui n'est pas déjà présente dans le tas) et la fusion de deux tas de Fibonacci (aux ensembles de clés disjoints). Les procédures doivent être en $O(1)$.

2) Dans cette question on cherche à écrire une procédure qui va extraire le nœud de clé minimale du tas. Une telle procédure ne peut se faire en temps constant car il faut en particulier mettre à jour le pointeur vers la racine de clé minimale, ce qui prend dans le pire des cas (toutes les racines sont des feuilles) $O(n(T))$ opérations.

Pour palier à ce problème on se sert de l'opération d'extracion pour ***consolider*** le tas. L'opération de consolidation consiste à limiter le nombre de racines du tas en regroupant les arbres du tas de même degré pour que toutes les racines aient un degré distinct.

Ecrire les procédures *consolider* et *extraire*. Calculer le coût amorti de l'opération d'extraction est alors en fonction de $D(n(T))$ (on procédera à une analyse par potentiel). On suppose que tous les arbres du tas de Fibonacci sont des ***arbres binomiaux non ordonnés***. Montrer que le coût amorti de l'opération d'extraction est alors en $O(log(n(T)))$

3) On veut maintenant écrire une procédure qui remplace la clé d'un nœud x par une valeur k plus petite, $diminuer(T, x, k)$. On procède comme suit :

- Soit $y = x \cdot p$, si $y \cdot cle < k$, on se contente de changer la clé de x.
- Sinon on coupe le sous-arbre de racine x pour en faire un nouvel arbre du tas.

Cette procédure ne préserve pas la structure d'arbre binomial non-ordonné. Pour préserver le coût amorti de la procédure d'extraction on demande donc une condition supplémentaire. Dans sa vie de nœud d'un tas de Fibonacci, un nœud x passe par les étapes suivantes :

1. Il est une racine.
2. Il devient le fils d'un autre nœud. (au cours d'une consolidation)
3. On lui coupe deux fils. (au cours d'une diminution)

On veut qu'une fois la troisième étape passée x redevienne une racine. Pour ce faire on ajoute aux nœuds un booléen $x \cdot marque$, qui est vrai si on a coupé un fils à x depuis qu'il a un père. Ecrire la procédure *diminuer* et montrer que son coût amorti est constant (on procédera à une analyse par potentiel).

4) On cherche maintenant à trouver une borne sur le degré des nœuds. Soit x un nœud dans un tas de Fibonacci construit avec les procédures écrites dans les questions précédentes et soit k son degré, soit $y_1, ..., y_k$ ses fils classés dans l'ordre où ils sont devenus fils de x. Montrer que $degre[y_1] \geq 0$ et pour $i \geq 2$, $degre[y_i] \geq i - 2$.

On appelle $taille(x)$ le nombre de nœuds du sous-arbre enraciné en x, montrer que $taille(x) \geq F_n \geq \Phi^k$ où F_n est le $n$ième nombre de Fibonacci (le nom des tas prend enfin tout son sens) et Φ le nombre d'or. Conclure.

Exercice 2 *Codage de Huffman, Algorithmes Gloutons*

Soit Σ un alphabet fini et de cardinal supérieur ou égal à deux. On appelle mot de code toute suite finie de 0 et de 1, soit $\mathcal{C}$ l'ensemble des mots de code. On appelle codage binaire une application injective α de l'alphabet Σ dans $\mathcal{C}$. En utilisant l'opération concaténation, α s'étend de manière naturelle en :

$$\alpha : \Sigma^* \to \mathcal{C}$$

On dit qu'un codage est de longueur fixe quand toutes les lettres sont codées par un mot de code de même longueur. Un codage est dit préfixe si aucune lettre n'est codée par un mot de code qui est préfixe du codage d'une autre lettre.

1) Montrer que pour une codage de longueur fixe et un codage préfixe, α est injective sur Σ^*.

2) Montrer qu'on peut représenter un codage préfixe par un arbre binaire dont les feuilles sont les lettres de l'alphabet.

On considère un mot w. A chaque codage préfixe de w, représenté par un arbre T, est associé un coût défini par :

$$C_w(T) = \sum_{a \in \Sigma} f(a) l_T(a)$$

avec $f(a)$ le nombre d'occurences de a dans w et $l_T(a)$ la taille du mot de code associé à a par T. Un codage préfixe est dit optimal si il minimise la fonction C_w

3) Montrer qu'à un codage préfixe optimal correspond un arbre binaire où tout nœud interne a deux fils.

4) Montrer qu'il existe un codage préfixe optimal pour lequel les deux lettres dont le nombre d'occurences est le plus faible sont sœurs dans l'arbre.

Etant donnés x et y les deux lettres dont le nombre d'occurences est le plus faible dans w, on considère l'alphabet $\Sigma' = \Sigma - x, y + z$ où z est une nouvelle lettre à laquelle on associe $f(z) = f(x) + f(y)$.

5) Soit T' l'arbre d'un codage optimal pour Σ', montrer que l'arbre T obtenu à partir de T' en remplaçant la feuille associée à z par un nœud interne ayant x et y comme feuilles représente un codage optimal pour Σ.

6) En déduire un algorithme recherchant un codage optimal et donner sa complexité.

Exercice 3 *Belle Impression*

Le problème consiste à effectuer une belle impression d'un paragraphe par une imprimante. L'entrée est un texte de n mots de longueur $l_1, l_2, ..., l_n$. Le but est d'imprimer ce paragraphe de manière équilibrée sur un nombre de lignes à calculer qui contiendront au plus m caractères chacune. Le critère d'équilibre est le suivant. Si une ligne donnée contient les mots i à j, le nombre de caractères d'espacement supplémentaires à la fin de la ligne est :

$$m - j + i - \sum_{k=i}^{j} l_j$$

L'objectif est de minimiser la somme, sur toutes les lignes hormis la dernière, des cubes des nombres de caractères d'espacement présents à la fin de chaque ligne.

1) Est-ce que l'algorithme glouton consistant à remplir les lignes une à une en mettant à chaque fois le maximum de mots possibles sur la ligne en cours, fournit l'optimum ?

2) Donner un algorithme de programmation dynamique résolvant le problème. Analyser sa complexité en temps et en espace.

Supposons que pour la fonction de coût à minimiser, on ait simplement choisi la somme des nombres de caractères d'espacement présents à la fin de chaque ligne.

3) Peut-on faire mieux en complexité que pour la question 2 ?

Exercice 4 *Polygones*

Etant donné un polygone convexe, une triangulation de celui-ci est un ensemble de cordes qui ne se coupent pas et le divisent en triangles.

1) Etant donné un polygone à n côtés, combien faut-il de cordes pour une triangulation, et combien de triangles sont produits.

2) On cherche à calculer pour un polygone donné une triangulation optimale. L'optimalité est définie en associant à toute triangulation un poids correspondant à la somme des périmètres

de ses triangles. Etant donné un polygone $p_1, ..., p_n$, on pose $t[i, j]$ le poids d'une triangulation optimale du polygone $p_{i-1}, ...p_j$ pour $1 \leq i \leq j \leq n$. Trouver une relation de récurrence sur t et en déduire un algorithme pour trouver une triangulation optimale.

3) Si on prend comme poids la somme des aires des triangles, que pensez-vous de votre algorithme ?

Exercice 5 *Comment construire une tour ?*

On veut construire une tour la plus haute possible à partir de différentes briques. On dispose de n types de briques et d'un nombre illimité de briques de chaque type. Chaque brique de type i est un parallélépipède de taille (x_i, y_i, z_i) et peut être orientée dans tous les sens, deux dimensions formant la base et la troisième dimension formant la hauteur. Dans la construction de la tour, une brique ne peut être placée au dessus d'une autre que si les deux dimensions de la base de la brique du dessus sont strictement inférieures aux dimensions de la base de la brique du dessous.

Proposer un algorithme efficace pour construire une tour de hauteur maximale.