Algorithmique TD6

Magistère Informatique 1ère Année

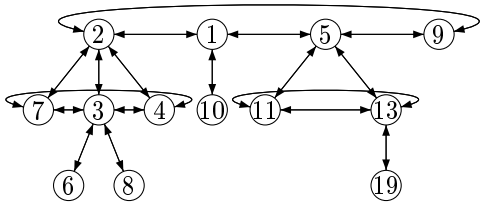
21 Octobre 2008

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 $taille(x) \geq F_{k+2} \geq \Phi^k$ 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.