Chapître X.
Tas de Fibonacci

1. Structures de données avancées

-) tas binomial

-) <u>tas de Fibonacci</u>

-) structure de données d'ensembles disjoints

-) B-arbres

2. Tas fusionnables

On appelle **tas fusionnable** une structure de données qui supporte les cinq opérations suivantes:

$CREER\text{–}TAS$ crée et retourne un nouveau tas sans élément.

$INSERER(T, x)$ insére dans le tas T un nœud x, dont le champ cle a déjà été rempli.

$MINIMUM(T)$ retourne un pointeur sur le nœud dont la clé est minimum dans le tas T.

$EXTRAIRE - MIN(T)$ supprime du tas T le nœud dont la clé est minimum, est retourne un pointeur sur ce nœud.

$UNION(T_1, T_2)$ crée et retourne un nouveau tas qui contient tous les nœuds des tas T_1 et T_2. Les tas T_1 et T_2 sont “detruits” par cette opérations.

Les tas binomiaux et les tas de Fibonacci supportent aussi les deux opérations suivantes.

$DIMINUER-CLE(T, x, k)$ affecte au nœud x du tas T la nouvelle valeur de clé k, dont on suppose qu'elle est inférieure à sa valeur de clé courante.

$SUPPRIMER(T, x)$ supprime le nœud x du tas T.

3. Tas binomial contre tas de Fibonacci

Les tas de Fibonacci bénéficient de bornes meilleures pour certaines opérations. Cependant il faut considérer le **temps d'exécution amorti**.

Remarque:

Les tas binaires, les tas binomiaux, et les tas de Fibonacci sont tous inefficaces pour l'opération $RECHERCHER$; trouver un nœud contenant une clé peut perdre beaucoup de temps.

Attention!

Heureusement cette contrainte ne pose aucun problème pour la plupart des applications.

4. Points importants

Les tas de Fibonacci sont une implémentation de **tas fusionnable** qui supportent toutes les opérations qui nécessitent pas la suppression d'un élément en **temps amorti** $O(1)$.

L'opération $EXTRAIRE{-}MIN$ est la plus compliquée et elle appelle à la procedure auxiliaire $CONSOLIDER$ qui réduit le nombre d'arbres du tas de Fibonacci.

D'un point de vue théorique, **les tas de Fibonacci se révèlent particulièrement adaptés quand le nombre d'opérations** $EXTRAIRE{-}MIN$ **et** $SUPPRIMER$ **(ou** $DIMINUER-CLE$**) est petit par rapport aux autres.**

Par exemple, une exécution de l'algorithme de DIJKSTRA ou de l'algorithme de PRIM appelle à $DIMINUER-CLE$ pour chaque arête. Alors le temps amorti en $O(1)$ de tas de Fibonacci améliore grandement le temps en $\Theta(\log n)$ du tas binomial dans le pire des cas.

Les algorithmes les plus rapides asymptotiquement pour des problèmes comme la recherche plus courts chemins à partir d'une source unique et le calcul des arbres couvrants de coût minimal ont essentiellement recours aux tas de Fibonacci.

Cormen, Leiserson, Rivest (p. 412):

> D'un point de vue pratique, toutefois, les facteurs constants et la programmation complexe des tas de Fibonacci les rendent moins intéressants que les tas binaires ordinaires, pour la plupart des applications.

5. Structure des tas de Fibonacci

Un tas de Fibonacci étant une collection d'arbres enraciné ordonnés en tas.

-) Les arbres d'un tas de Fibonacci ne sont pas obligatoirement des arbres binomiaux.

-) Ils sont arbres enraciné mais il n'y pas d'ordre de fils d'un nœud.

Si ces opérations - $CREER{-}TAS$, $INSERER$, $MINIMUM$, $EXTRAIRE - MIN$ et $UNION$ - sont les seules requises, chaque tas de Fibonacci est simplement une collection d'**arbres binomiaux non triés**.

Définition. Un **arbre binomial non trié** U_k est un arbre enraciné. défini recursivement.

(i) U_0 est consitué d'un nœud unique.

(ii) U_k est constitué de deux arbres binomiaux non triés U_{k-1} qui sont **reliés** entre eux: La racine de l'un est un fils de l'autre.

Exemple:

Lemme (Propriété des arbres binomiaux non triés).

Soit U_k un arbre binomial non triés.

1. Le nombre de nœuds de U_k est 2^k.

2. La hauteur de U_k est k.

3. Il existe exactement C_k^i nœuds à la profondeur i pour $i = \grave{a}, 1, 2 \ldots, k$.

4. La racine a un degré k, qui est plus grand que que celui de n'importe quel autre nœud. Les fils de la racine sont les racines des sous-arbres $U_0, U_1, \ldots, U_{k-1}$ dans un certain ordre.

6. Degré maximum

Corollaire. *Si un tas de Fibonacci à n nœuds est une collection d'arbres binomiaux non triés, alors degré maximum d'un nœud quelconque est* $\log n$.

Les analyses amorties supposent qu'il existe une borne supérieure connue $D(n)$ pour le degré maximum d'un nœud quelconque d'un tas de Fibonacci à n nœuds.

Si les opérations sur les tas fusionnables sont les seules supportées, $D(n) = \log n$.

Quand $DIMINUER-CLE$ et $SUPPRIMER$ sont également supportées on peut montrer que $D(n) = O(\log n)$.

7. Représentation des tas de Fibonacci

Chaque nœud x comporte:

-) un champ $cle[x]$

-) un champ $degre[x]$ qui représente le nombre de fils de x

-) un pointeur $p[x]$ sur son père

-) un pointeur $fils[x]$ sur l'un **quelconque** de ses fils

Les fils de chaque nœud x sont reliés dans une liste circulaire doublement chaînée que on appelle la **liste des fils** de x. L'ordre dans lequel apparaissent les frères dans une liste des fils est arbitraire.

Chaque fils y d'une liste des fils possède:

-) un pointeur $gauche[y]$ sur le frère gauche

-) un pointeur $droit[y]$ sur le frère droit

Les listes circulaires doublement chaînées ont deux avantage:

-) Un nœud peut être supprimer en $O(1)$.

-) Deux listes de ce type, on peut les concaténer en $O(1)$.

On peut accéder un tas de Fibonacci par un pointeur $min[T]$ sur la racine de l'arbre contenant une clé minimale.

Le nombre total de nœuds présents dans T est conservé dans $n[T]$.

Finalement chaque nœud x comporte **un champ** $marque[x]$ **de valeur booléenne qui indique si le nœud** x **a perdu un fils depuis la dérniere fois où** x **est devenu fils d'un autre nœud.**

8. Fonction potentiel

On utilise **la méthode du potentiel** pour analyser la performance amortie des opérations sur les tas de Fibonacci.

Pour un certain tas de Fibonacci, on répresente par

-) $a(T)$ le nombre d'arbres contenus dans T (c'est-à-dire le nombre de nœuds dans la liste des racines de T) et par

-) $m(T)$ le nombre de nœuds marqués dans T.

Fonction potentiel:

$$\Phi(T) = a(T) + 2m(T)$$

Maintenant on peut analyser le temps amorti d'une opérations Op par analyser la valeur $\Phi(T') - \Phi(T)$, si T est un tas de Fibonacci quelconque et T' peut être obtenu de T utilisant l'opération Op.

9. Opérations sur les tas fusionnables excepté $EXTRAIRE - MIN$

$CREER - TAS$ et $MINIMUM$ s'exécutent en temps $O(1)$ dans le pire des cas.

Stratégie paresseuse (lazy strategy):

Pour d'autres opérations des tas fusionnables sur les tas de Fibonacci, la stratégie est de reporter le plus longtemps possible le travail à effectuer.

Exemple:

Contrairement à la procédure $TAS-BINOMIAL-INSERER$ la procédure $TAS-FIB-INSERER$ n'essaye pas de consolider les arbres à l'intérieur du tas de Fibonacci. Après k opérations $TAS - FIB - INSERER$ consécutives, les k arbres à un nœud sont ajoutés à la liste des racines.

$TAS - FIB - INSERER(T, x)$

Ajouter le nœud x à la liste des racines de T tel que le nœud x devient un arbre ordonné un tas à un seul nœud.

$\Rightarrow$ en temps $O(1)$ dans le pire des cas

coût amorti $O(1)$

Remarque. On ne s'inquiete pas même si la liste des racines devient très longue. On fait traîner le travail en longueur. Dans cette façon $INSERER$ est trés vite et on gagne du temps (amorti) pour des opérations coûteuses.

$TAS - FIB - UNION(T_1, T_2)$

Concatener les listes des racines de T_1 et T_2 en une nouvelle liste de T.

$\Rightarrow$ en temps $O(1)$ dans le pire des cas

coût amorti $O(1)$

Remarque. Comme pour la procédure $TAS - FIB - INSERER$ aucune consolidation des arbres n'intervient (stratégie paresseuse).

10. Extraction du nœud minimum

On utilise une implémentation puissante pour les opérations $CREER-TAS$, $MINIMUM$, $INSERER$ et $UNION$. La listes des racines de tas de Fibonacci est peut-être trés longue.

Pour l'opération $EXTRAIRE-MIN$ il faut absolument trouver le nouveau minimum. C'est-à-dire qu'une consolidation est necessaire et ce fait par appel à la procedure $CONSOLIDER$.

Même si cette opération est peut-être coûteuse, l'analyse amorti montre que on a économisé avant l'appel à la procedure $CONSOLIDER$ et le coût amorti est $O(D(n))$.

Le processus d'extraction du nœud minimum est l'opération la plus compliquée.

C'est également dans l'opération $EXTRAIRE-MIN$ que survient le travail de consolidation des arbres de la liste des racines.

$TAS - FIB - EXTRAIRE - MIN(T)$

1. Supprimer le nœud sur lequel $min[T]$ pointe.

2. Transformer en racine chaque fils du nœud minimum.

3. Consolider la liste des racines en reliant entre elles les racines de degré égal, jusqu'à ce qu'il n'existe plus qu'une racine au maximum de chaque degré.

Consolidation

La procédure $CONSOLIDER$ utilise une approche qui est similaire d'approche dans la procédure $TAS - BINOMIAL - UNION$.

En effet, les racines dans la liste des racines d'un tas de Fibonacci ne sont pas triées par degré croissant comme dans la listes des racines d'un tas binomial.

C'est pourquoi, la consolidation d'un tas de Fibonacci est plus compliquée que la consolidation d'un tas binomial (fait dans la procédure $TAS - BINOMIAL - UNION$).

La phase de **consolidation** de la liste des racines de T **réduit le nombre d'arbres du tas de Fibonacci**.

Consolider la liste des racines revient à exécuter de manière répétée les étapes suivantes, jusqu'à ce que toutes les racines de la liste aient une valeur $degre$ distincte.

1. Trouver dans la liste des racines deux racines x et y ayant le même degré, et pour lesquelles $cle[x] \leq cle[y]$.

2. **Relier** y à x: supprimer y de la liste des racines, et faire de y un fils de x.
$(TAS-FIB-RELIER)$

La procédure $CONSOLIDER$ utilise un tableau auxiliaire $A[0..D(n[T])]$. Si $A[i]=y$, alors y est une racine pour laquelle $degre[y]=i$.

$EXTRAIRE - MIN$ et $CONSOLIDER$ s'exécutent en temps $O(n)$ au pire des cas.

Théorème. *Le coût amorti de l'opération* $EXTRAIRE - MIN$ *d'un tas de Fibonacci à* n *nœuds est* $O(D(n))$.

11. Diminution d'une clé et suppression d'un nœud

Ces opérations ne préservent pas la propriété selon laquelle tous les arbres d'un tas de Fibonacci sont des arbres binomiaux non triés.

Neanmois, ils en sont assez proches pour qu'on puisse borner le degré maximum $D(n)$ par $O(\log n)$.

A cause de ces opérations il est difficile de dire quels arbres peut être contenu dans un tas de Fibonacci est quels arbres ne sont pas contenu dans un tas de Fibonacci.

Cependant on peut montrer que **le nombre de nœuds dans un sous-arbres enraciné à un nœud de k fils est au moins $(\frac{1+\sqrt{5}}{2})^k$.**

$TAS - FIB - SUPPRIMER(T, x)$

1 $\quad TAS{-}FIB{-}DIMINUER{-}CLE(T, x, -\infty)$
2 $\quad TAS - FIB - EXTRAIRE - MIN$

$TAS - FIB - SUPPRIMER$ est analogue à $TAS - BINOMIAL - SUPPRIMER$.

12. Diminution d'une clé
$TAS\!-\!FIB\!-\!DIMINUER\!-\!CLE(T, x, k)$

Si k, c'est-àdire la nouvelle clé de x, est inferieure à clé de $y = p[x]$, alors l'ordre de tas n'a pas été respecté.

De nombreux changement peuvent survenir:

1. **Détacher** x de son père y par $COUPER$ qui coupe le lien entre x et son père. x devient une racine et $marque[x] = FAUX$.

2. La procédure $COUPER\!-\!EN\!-\!CASCADE(T, y)$ est appelé. Si $marque[y] = FAUX$ on marque y est s'arête. Si $marque[y] = VRAI$ on coupe y de son pére z etc.

La procédure $COUPE - EN - CASCADE$ remonte dans l'arbre, jusqu'à trouver soit une racine, soit un nœud non marqué.

$DIMINUER - CLE$ s'exécute en temps $O(n)$ au pire des cas.

Théorème. *Le coût amorti de l'opération* $DIMINUER - CLE$ *d'un tas de Fibonacci à* n *nœuds est* $O(1)$*.*

13. Borne pour le degré maximum

Nous avons démontrer que le temps amorti de chaque opération d'un tas de Fibonacci est 0(1) excepté $TAS-FIB-EXTRAIRE-MIN$ et $TAS-FIB-SUPPRIMER$ pour qui le temps amorti est $O(D(n))$.

Pour obtenir le temps amorti $O(\log n)$ de $TAS-FIB-EXTRAIRE-MIN$ et $TAS-FIB-SUPPRIMER$ il reste de montrer que $D(n) = O(\log n)$.

Nous démontrerons que $D(n) \leq log_\phi n$, où $\phi = (1+\sqrt{5})/2 \approx 1.61803$.

Le principe de l'analyse est le suivant. Pour chaque nœud x du tas de Fibonacci, on definit $taille(x)$ comme étant le nombre de nœuds, y compris x lui-même, dans le sous-arbre enraciné en x.

Nous allons montrer que $taille(x)$ est une fonction exponentielle de $degre[x]$.

Remarque. $taille(x) = 2^k$ n'est pas juste à cause des opérations $DIMINUER-CLE$ et $SUPPRIMER$. Cependant quelque soit x de degré k, $taille(x) \geq \phi^k$.

Lemme 1: Soit x un nœud quelconque d'un tas de Fibonacci, avec $degre[x] = k$. Soient $y_1, y_2, \ldots, y_k$ les fils de x dans l'ordre dans lequel ils sont reliés à x.

Dans ce cas, $degre[y_1] \geq 0$ et $degre[y_i] \geq i - 2$ pour $i = 2, 3, \ldots, k$.

Démonstration:

Définition: Le $k$ème nombre de Fibonacci F_k est défini par la récurrence:
$F_{k+2} = F_{k+1} + F_k$, si $k \geq 2$
$F_0 = 0, \quad {}_1 = 1$

Théorème (Moivre): Pour tous les entiers $k \geq 0$,

$$F_k = \frac{(\frac{1+\sqrt{5}}{2})^k - (\frac{1-\sqrt{5}}{2})^k}{\sqrt{5}}.$$

Corollaire: Pour tous les entiers $k \geq 0$,

$$F_{k+2} \geq (\frac{1+\sqrt{5}}{2})^k,$$

où $\frac{1+\sqrt{5}}{2} \approx 1.61803$.

Lemme 2: Pour tous les entiers $k \geq 0$,

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_i$$

.

Démonstration: Par récurrence sur k.

Quand $k = 0$, $F_2 = 1 + \sum_{i=0}^{0} F_i$.

L'hypothèse d'induction: $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$, et on a

$F_{k+2} = F_k + F_{k+1}$

$= F_k + 1 + \sum_{i=0}^{k-1} F_i$

$= 1 + \sum_{i=0}^{k} F_i$. $\square$

Lemme 3: Soit x un nœud quelconque d'un tas de Fibonacci, avec $degre[x] = k$.
Alors $taille(x) \geq F_{k+2} \geq \phi^k$, où $\phi = \frac{1+\sqrt{5}}{2}$.