Chapître VIII.
Tas binomiaux

1. Structures de données avancées

-) tas binomial

-) tas de Fibonacci

-) structure de données d'ensembles disjoints (UNION-FIND structure)

-) B-arbres

2. En voici quelques autres:

-) arbres van Emde Boas

supportent les opérations $MINIMUM$, $MAXIMUM$, $INSERER$, $SUPPRIMER$, $RECHERCHER$, $EXTRAIRE - MIN$, $EXTRAIRE - MAX$, $PREDECESSEUR$ et $SUCCESSEUR$ en $O(\log\log n)$ dans le pire des cas, à la condition que l'univers des clés soit l'ensemble $\{1, 2, \ldots, n\}$.

-) arbres dynamiques

introduits par Sleator et Tarjan, gèrent une fôret d'arbres disjoints. Les arbres dynamiques supportent les requêtes permettant de trouver le père, les racines, les coûts des arêtes et le coût minimum sur un chemin allant d'un nœud à la racine en $O(\log n)$ dans le pire des cas.

-) **arbres déployés**

sont une forme d'arbre binaire de recherche sur laquelle les opérations fondamentales des arbres de recherche s'exécutent avec un **temps amorti** (amortized time) en $O(\log n)$.

-) **structures de données persistantes**

Au cours d'un algorithme, on a parfois besoin de conserver d'ancienes versions d'un ensemble dynamique quand celui-ci est modifié. Un tel ensemble est dit **persistant**. Il s'agit de le conserver sans perdre beaucoup d'espace et d'une façon qui supporte toutes les opérations.

3. 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 fussionnables que nous considérons supportent 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.

Remarque: Un tas ou un tas fusionnable ne supporte pas $RECHERCHER$.
C'est-à-dire que le nœud x doit être donné par un pointeur pour $DIMINUER - CLE(T, x, k)$ et $SUPPRIMER(T, x)$.

4. Tas binomial contre tas de Fibonacci

Comme l'opération $UNION$ n'est pas nécessaire, les tas binaires ordinaires, comme ceux utilisés pour le `tri_tas` sont au moins aussi efficace que les tas binomiaux.

Les tas de Fibonacci bénéficient de bornes encore 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.

5. Arbres binomiaux

Un tas binomial étant une collection d'arbres binomiaux.

Définition. Un **arbre binomial** B_k est un arbre enraciné dans lequel les fils de chaque nœud sont ordonnés. (En d'autres termes, si un nœud a k fils, il existe un premier fils, un deuxième fils,..., et un $k$ième fils.)

Un arbre binomial B_k est défini récursivement.

(i) B_0 est consitué d'un nœud unique.

(ii) B_k est constitué de deux arbres binomiaux B_{k-1} qui sont **reliés** entre eux: La racine de l'un est le fils le plus à gauche de la racine de l'autre.

Exemple:

Lemme (Propriété des arbres binomiaux).

Soit B_k un arbre binomial.

1. Le nombre de nœuds de B_k est 2^k.

2. La hauteur de B_k est k.

3. Il existe exactement C_k^i nœuds à la profondeur i pour $i = \grave{a}, 1, 2 \ldots, k$.
(On appelle **profondeur d'un nœud** *dans un arbre enraciné la distance de ce nœud à la racine.)*

4. La racine a un degré k, qui est supérieur de degré de tout autre nœud; par ailleurs, si le fils de la racine sont numéroté de la gauche vers la droite par $k-1, k-2, \ldots, 1, 0$ le fils i est la racine d'un sous-arbre B_i.

Corollaire. *Le degré maximum d'un nœud quelquonque dans un arbre binomial à n nœuds est* $\log n$.

Démonstration:

La démonstration se fait par récurrence sur k. Pour chaque propriété, la base est l'arbre binomial B_0. Vérifier que chaque propriété est valide pour B_0 c'est trivial.

Pour l'étape d'induction, on suppose que le lemme est valable pour B_{k-1}.

1. L'arbre binomial B_k est constitué de deux copies de B_{k-1}, et donc B_k contient $2^{k-1} + 2^{k-1} = 2^k$ nœuds.

$\square$

6. Tas binomiaux

Un arbre binomial peut gérer un ensemble dynamique avec comme taille une puissance exacte de 2. Un tas binomial peut gérer un ensemble dynamique ordonné quelconque.

Définition. Un **tas binomial (binomial heap)** T est un ensemble d'arbres binomiaux qui satisfait les **propriétés des tas binomiaux**.

1. Chaque arbre binomial dans T est **ordonné en tas (heap-ordered)**: quelque soit nœud x

$$cle[x] \geq cle[p[x]].$$

2. Il existe dans T au plus un arbre binomial dont la racine possède un degré donné.

La racine d'un arbre ordonné d'un tas contient la plus petite clé de l'arbre.

La deuxième propriété implique qu'un tas binomial T à n nœuds est constitué d'au plus $\lfloor \log n \rfloor + 1$ arbres binomiaux. En effet, soit

$$bin(n) = \langle b_{\lfloor \log n \rfloor}, \ldots b_1, b_0 \rangle$$

la représentation binaire de n, c'est-à-dire que $n = \sum_{i=0}^{\lfloor \log n \rfloor} b_i \cdot 2^i$. Puisque le nombre de sommets d'un arbre binomial B_k est 2^k donc **l'arbre binomial**

$$B_i \textbf{ apparaît dans } T \textbf{ ssi } b_i = 1.$$

Exemple: $n = 13$

$\Rightarrow \quad bin(13) = \langle 1, 1, 0, 1 \rangle$

$\Rightarrow$ Si T est un tas binomial à 13 nœuds alors il est constitué des arbres binomiaux B_3, B_2 et B_0 qui comportent respectivement 8, 4, et 1 nœuds, pour un total de $n = 13$ nœuds.

7. Représentation des tas binomiaux

Chaque arbre binomial se trouvant à l'intérieur d'un tas binomial est stocké dans la représentation **"fils gauche, frère droit"**.

Chaque nœud x comporte:

-) un champ $cle[x]$
(et toute autre information satellite utile à l'application)

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

-) un pointeur $p[x]$ sur son père

-) un pointeur $fils[x]$ sur son fils le plus à gauche

-) un pointeur $frere[x]$ sur le frère qui se trouve immédiatement à sa droite

Liste des racines

Les racines des arbres binomiaux contenus dans un tas binomial sont organisées en **liste chaînée**, qu'on appelle **liste des racines**.

Les **degrés des racines augmentent strictement** lorsqu'on parcourt la liste des racines.

Si x est une racine d'un arbre binomial, $frere[x]$ pointe sur la racine suivante dans la liste des racines.

On peut accéder à un tas binomial T donné par le champ $tete[T]$ qui est un pointeur sur le premier élément de la liste des racines de T.

8. Une procédure de base: $LIEN - BINOMIAL$

L'opération la plus intéressante dans un tas binomial est l'opération $UNION$. Nous l'étudions en détail.

La procédure $UNION - TAS - BINOMIAUX$ relie de façon répetée des **arbres binomiaux dont les racines ont le même degré**.

La procédure $LIEN - BINOMIAL$ relie l'arbre B_{k-1} de racine y à l'arbre B_{k-1} de racine z; (autrement dit, il fait de z le père de y). Le nœud z devient donc la nouvelle racine d'un arbre B_k.

$LIEN - BINOMIAL(y, z)$

$p[y] \leftarrow z$
$frere[y] \leftarrow fils[z]$
$fils[z] \leftarrow y$
$degre[z] \leftarrow degre[z] + 1$

La procédure $LIEN - BINOMIAL$ s'exécute en temps $O(1)$. Elle fait du nœud y la tête de la liste chaînée des fils du nœud z.

L'arbre binomial B_k obtenu est ordonné en tas, si les deux arbres B_{k-1} etaient ordonné en tas et en plus $cle[y] \geq cle[z]$.

9. Union de deux tas binomiaux

La procédure $UNION-TAS-BINOMIAUX$ réunit les tas T_1 et T_2, et retourne le tas résultant. Elle détruit au cours du traitement les représentations de T_1 et T_2.

Procédure auxiliaire:

-) $LIEN-BINOMIAL$

-) $FUSIONNER-TAS-BINOMIAUX$ qui fusionne les listes des racines de T_1 et T_2 en une seule liste chaînée, triée par degré croissant.

Théorème. *Soit* $n = |T_1| + |T_2|$. $UNION-TAS\text{-}BINOMIAUX$ *s'exécute en temps* $O(\log n)$.

10. Autres opérations

On peut implémenter quelques opérations en utilisant $UNION-TAS-BINOMIAUX$.

$INSERER(T,x)$

Créer un tas binomial T' qui contient uniquement le nœud x. Après appel $UNION-TAS-BINOMIAUX(T,T')$.

$\Rightarrow$ en temps $O(\log n)$ au pire des cas

$MINIMUM(T)$

Puisqu'un tas respecte la propriété d'ordre de tas, la clé minimum doit se trouver dans un nœud racine. On fait un parcours de liste des racines.

$\Rightarrow$ en temps $O(\log n)$ au pire des cas

$EXTRAIRE - MIN(T)$

Appel $MINIMUM(T)$ et supprime la racine x de clé minimum. Créer un binomial T' où les fils de x se trouve de la liste des racines en ordre reverse. Appel $UNION\!-\!TAS\!-\!BINOMIAUX(T, T')$.

$\Rightarrow$ en temps $O(\log n)$ au pire des cas

$DIMINUER - CLE$

Diminuer la valeur de clé de nœud x à k. Apres (comme dans un tas binaire) le nœud x remonte à la direction de racine jusqu'a sa valeur est inférieure de valeur de son père courant.

$\Rightarrow$ en temps $O(\log n)$ au pire des cas

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

1 $TAS\text{–}BINOMIAL\text{–}DIMINUER\text{–}CLE(T, x, -\infty)$
2 $TAS - BINOMIAL - EXTRAIRE - MIN(T)$

$\Rightarrow$ en temps $O(\log n)$ au pire des cas