Université Pierre et Marie Curie
Année 2008-2009

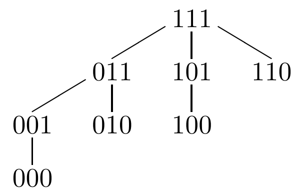
Master Informatique
MI015-ALGAV

Devoir

1 Partiel de novembre 2004

1.1 Arbres binômiaux [5 points]

On étiquette l'arbre binômial B_k (formé de 2^k nœuds) en ordre postfixé, chaque étiquette (de 0 à $2^k - 1$) étant un mot binaire sur k bits. Par exemple, pour B_3, on obtient :



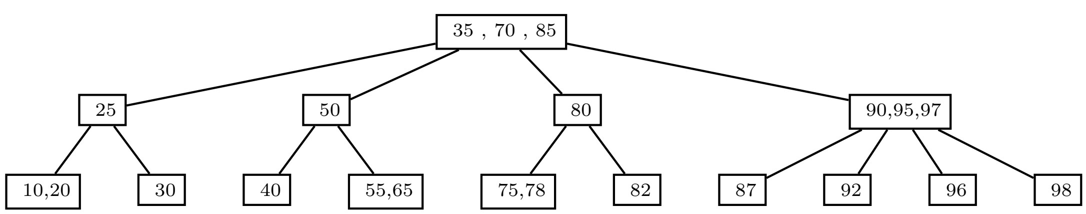
1. Dessiner l'arbre binômial B_4 dont les nœuds sont étiquetés en binaire en ordre postfixé.
2. Dans B_k, soit x un nœud à profondeur i, dont l'étiquette est e. Montrer que le nombre de 1 dans e est égal à $k - i$.
3. Combien d'étiquettes de B_k contiennent exactement $k - i$ bits égaux à 1 ?
4. Montrer que le degré d'un nœud est égal au nombre de 1 à droite du 0 le plus à droite de son étiquette (et si l'étiquette ne contient pas de 0, le degré du nœud est égal au nombre de 1).

1.2 Coût amorti [5 points]

Dans cet exercice, on considère l'insertion aux feuilles dans un arbre 2-3-4, avec éclatement des 4-nœuds à la descente (algorithme vu en cours) et on s'intéresse à sa complexité, comptée en nombre d'éclatements. Calculer le coût amorti d'une insertion dans un arbre 2-3-4, en utilisant la méthode du potentiel. Pour un arbre 2-3-4 A ayant n_4 4-nœuds et n_3 3-nœuds, on prendra comme potentiel $\Phi(A) = 2n_4 + n_3$.

1.3 Concaténation d'arbres 2-3-4 [10 points]

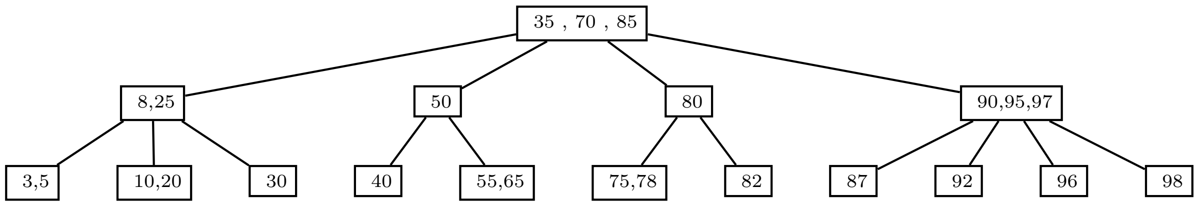
La figure suivante montre un arbre 2-3-4, T_0, de hauteur 2, ayant un nœud à profondeur 0 (la racine), quatre nœuds à profondeur 1 et dix nœuds à profondeur 2 (ces nœuds sont les *feuilles* de T_0).



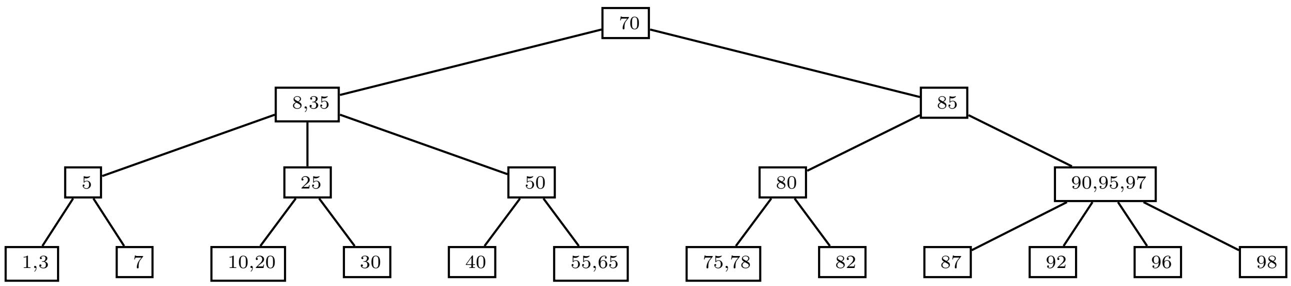
Le but de l'exercice est de définir les fonctions calculant la hauteur et le nombre de nœuds à profondeur i dans un arbre 2-3-4, puis de définir la concaténation de deux arbres 2-3-4. **Toutes ces définitions seront implantées en utilisant les primitives sur les arbres 2-3-4 données en cours.**

1. Écrire une définition de la fonction `hauteur`, qui renvoie la hauteur d'un arbre 2-3-4
2. Écrire une définition de la fonction `nb-noeuds-prof`, qui étant donnés un arbre 2-3-4 T et un entier i renvoie le nombre de nœuds à profondeur i dans T.
3. Calculer l'expression du nombre maximum et du nombre minimum de nœuds à profondeur i dans un arbre 2-3-4. En déduire un majorant de la hauteur d'un arbre 2-3-4 ayant n feuilles.
4. On considère deux arbres 2-3-4, T_1 et T_2, tels que toutes les étiquettes de T_1 sont strictement inférieures à toutes les étiquettes de T_2, et x une étiquette strictement comprise entre les étiquettes de T_1 et celles de T_2. Il s'agit de concaténer T_1, x et T_2 pour en faire un arbre 2-3-4.
– Si les hauteurs de T_1 et T_2 sont égales, il suffit de construire l'arbre ayant pour racine le nœud d'étiquette x, pour sous-arbre gauche T_1 et pour sous-arbre-droit T_2.
– Si h_1, la hauteur de T_1, est strictement inférieure à h_2, hauteur de T_2, soit U le sous-arbre dont la racine est à profondeur $h_2 - h_1$ sur la branche gauche de T_2. On concatène T_1, x et U en incluant x dans le nœud père de U, de telle sorte que toutes les feuilles de l'arbre résultant sont au même niveau.

– Si le nœud père de U n'était pas un 4-nœud, on a terminé, comme le montre l'arbre ci-dessous, qui est la concaténation de l'arbre T_1 réduit à une feuille contenant les valeurs $(3, 5)$, de l'étiquette $x = 8$ et de l'arbre $T_2 = T_0$.



– Si le nœud père de U était un 4-nœud il faut rétablir la situation par des éclatements. Par exemple la concaténation de l'arbre $T_1 =< 5, (1, 3), 7 >$, de l'étiquette $x = 8$ et de l'arbre $T_2 = T_0$ donne l'arbre ci-dessous.



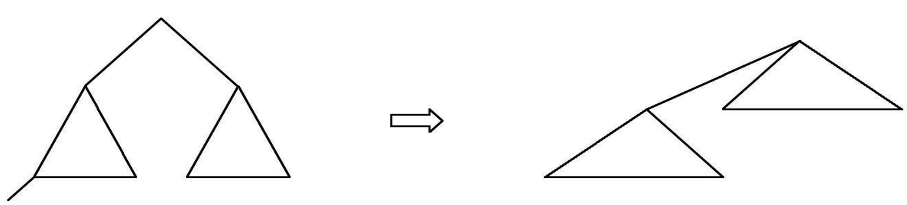
– Si $h_1 > h_2$ on agit de manière analogue sur la branche droite de T_1.

Quel est le résultat de la concaténation de l'arbre $T_1 = T_0$, de l'étiquette $x = 100$ et de l'arbre T_2 réduit à une feuille contenant les valeurs $(105, 110, 120, 125)$.
Décrire l'algorithme de concaténation.
Déterminer le nombre maximum d'éclatements.
5. Il s'agit à présent de concaténer deux arbres 2-3-4, T_1 et T_2, tels que toutes les étiquettes de T_1 sont strictement inférieures à toutes les étiquettes de T_2, *sans l'intermédiaire* d'une étiquette x.
Expliquer quels sont les différents cas de figure possibles et comment étendre l'algorithme précédent.

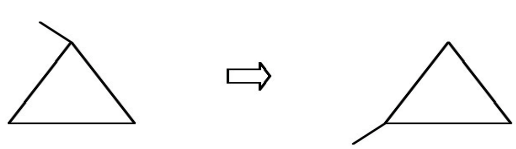
2 Partiel de novembre 2005 (extrait)

2.1 Tas et arbre binômial [5 points]

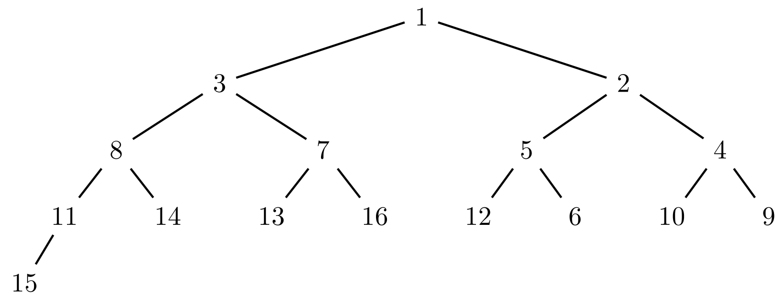
Le but de cet exercice est de convertir un tas binaire (arbre binaire parfait croissant) de 2^k éléments en un tas binômial (arbre binômial croissant), sans faire aucune comparaison.
Un tas binaire avec 1 élément est aussi un tas binômial. Pour $k \geq 1$, un tas binaire avec 2^k éléments est constitué d'un sous-arbre gauche qui est un tas binaire avec 2^{k-1} éléments et d'un sous-arbre droit qui est un tas binaire avec $2^{k-1} - 1$ éléments. On transforme d'abord le sous-arbre droit et la racine en un tas binaire de 2^{k-1} éléments, puis on convertit récursivement ces deux tas en des tas binômiaux de taille 2^{k-1} et on les concatène pour former un tas binômial de taille 2^k, comme le suggère la figure suivante.



1- Décrire un algorithme qui transforme en un tas binaire le sous-arbre droit et la racine d'un tas binaire (en gardant la propriété de tas). Calculer le nombre d'opérations effectuées.



2- Décrire l'algorithme de conversion un tas binaire de taille 2^k en un tas binômial. Et donner le résultat de cette conversion sur l'arbre suivant.



3- Exprimer par une relation de récurrence le nombre de concaténations de tas binômiaux effectués lors de la conversion d'un tas binaire de taille $n = 2^k$. Résoudre cette récurrence.

2.2 Coût amorti [5 points]

Une file FIFO est une structure de donnée linéaire qui supporte deux opérations
- *ajouter(x,F)*, qui ajoute un élément à la file,
- *enlever(F)*, qui enlève de la file l'élément le plus anciennement présent.

On peut implanter une file F à l'aide de deux piles P_1 et P_2 (et les opérations classiques *empiler(x,P)* et *dépiler(P)*), de la façon suivante :
- *ajouter(x,F)* : *empiler(x,P_1)*
- *enlever(F)* : si P_2 est vide alors dépiler tous les éléments de P_1 et les empiler au fur et à mesure dans P_2. Puis (dans tous les cas) dépiler P_2.

1- Montrer que cette implantation est correcte.

2- On va calculer le coût amorti des opérations *ajouter(x,F)* et *enlever(F)* en essayant différentes fonctions de potentiel. Le coût est mesuré en nombre d'opérations faites sur les piles.

a) essai 1 : la fonction de potentiel vaut deux fois le nombre d'éléments de la pile P_1.

b) essai 2 : la fonction de potentiel est égale au nombre d'éléments de P_1 moins le nombre d'éléments de P_2.

Pour chacune des deux fonctions précédentes, donner le coût amorti des opérations *ajouter(x,F)* et *enlever(F)* et dire s'il est possible que le coût réel total soit supérieur au coût amorti total. Peut-on retenir ces deux fonctions ?

2.3 Arbres bicolores [5 points]

La suppression dans un arbre bicolore étant délicate, on envisage de faire de la suppression paresseuse : plutôt que de supprimer "physiquement" chaque sommet au fur et à mesure, on se contente de le *marquer* comme étant supprimé, et lorsque la moitié des sommets d'un arbre sont marqués, on reconstruit totalement l'arbre (avec les "bons" sommets, *i.e.* les sommets non supprimés).

1- Montrer qu'étant donnée une liste triée on peut construire *en temps linéaire* un arbre bicolore contenant les éléments de cette liste.

2- Donner un algorithme qui, étant donné un arbre bicolore de n sommets dont certains sont marqués comme supprimés, reconstruit *en temps* $O(n)$ un arbre bicolore avec les bons sommets.

3- Supposons que l'on part d'un arbre bicolore de n éléments et que l'on effectue $n/2$ opérations de recherche, ajout ou suppression paresseuse. Montrer que le coût total de cette suite d'opérations est en $O(n \log n)$.