Algorithmique et structures de données : examen de première session

ENSIMAG 1A

Année scolaire 2008–2009

Consignes générales :
- Durée : 3 heures. Tous documents et calculatrices autorisés.
- Le barème est donné à titre indicatif. Il est sur 15 points, la note de chacun des 2 TP étant ramenée sur 2.5 points.
- Les exercices sont indépendants et peuvent être traités dans le désordre.
- La syntaxe Ada ne sera pas un critère déterminant de l'évaluation des copies. En d'autres termes, les correcteurs n'enlèveront pas de point pour une syntaxe Ada inexacte, mais compréhensible (pensez à bien commenter votre code !).
- **Merci d'indiquer votre numéro de groupe de TD et de rendre votre copie dans le tas correspondant à votre groupe.**

Exercice 1 : B-arbres *(10 pts)*

Un *B-arbre* est une structure de données permettant d'implémenter un dictionnaire de manière efficace. Il s'agit d'une structure hybride mêlant les structures d'arbres et de listes (ou de tableaux). En effet, chaque nœud de l'arbre contient plusieurs clés du dictionnaire. Un B-arbre peut être vu comme une généralisation d'un arbre binaire de recherche au sens où les valeurs des clés stockées dans les nœuds le sont de manière imbriquée.

Un *B-arbre de degré d*, $d \geq 2$, est un arbre (pas forcément binaire) tel que :

1. chaque nœud x comporte quatre informations :
   - un entier $k \geq 1$: le nombre de clés stockées dans ce nœud ;
   - une liste de k clés, généralement stockées par ordre croissant : $cle_1(x) \leq cle_2(x) \leq \ldots \leq cle_k(x)$;
   - un booléen $EstFeuille$, indiquant si x est une feuille de l'arbre ou non ;
   - un pointeur pour chaque nœud fils de x, dans le cas où x n'est pas une feuille ;
2. le nombre de fils d'un nœud x est exactement $k + 1$ (sauf bien sûr si x est une feuille) ;
3. $d \leq k + 1 \leq 2d$, excepté pour la racine de l'arbre qui peut avoir moins de d fils ;
4. les valeurs des clés d'un nœud x et les valeurs des clés de ses fils sont intercalées, voir figure 1 pour un exemple : les clés du premier fils de x ont des valeurs inférieures à la valeur de la première clé de x, les clés du deuxième fils de x ont des valeurs comprises entre les valeurs de la première et de la deuxième clés de x, etc. ;

5. les feuilles sont toutes situées à la même hauteur dans l'arbre, c'est-à-dire au dernier niveau.

La figure 1 montre un exemple de B-arbre.

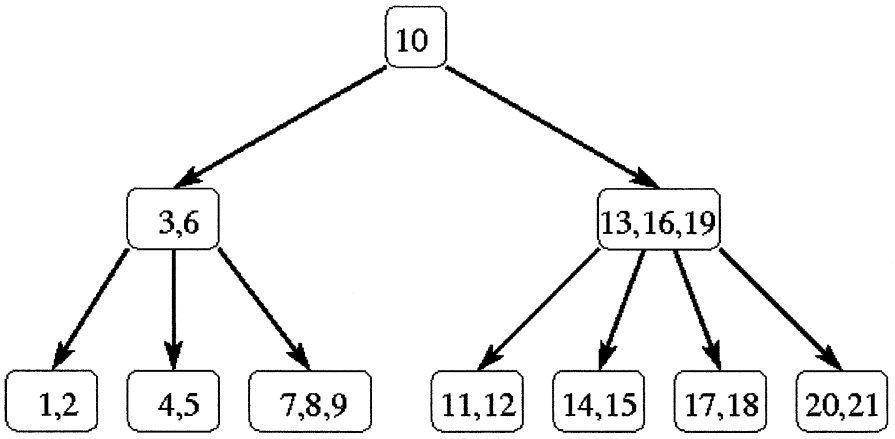

FIG. 1 – Exemple d'un B-arbre.

L'objectif de cet exercice est d'étudier comment implémenter les trois opérations standards que sont la recherche, l'insertion et la suppression d'une clé pour cette structure de données.

1 Questions générales

Question 1 Pour quel(s) degré(s) d l'arbre de la figure 1 est-il un B-arbre valide? Justifier.

Question 2 Dessiner les quatre B-arbres de degré supérieur ou égal à 2 et dont les clés ont les valeurs { 1, 2, 3, 4, 5 }.

Question 3 Définir, en Ada, un type `B-Arbre`. On suppose connus la valeur (fixe) de d et le type `Cle`.

Question 4 Soit n le nombre de clés d'un B-arbre de degré d, et h sa hauteur. Montrer que

$$h \leq log_d \frac{n+1}{2} \tag{1}$$

On rappelle que la hauteur de l'arbre vide est fixée à -1.

2 Recherche d'une clé

On s'intéresse dans cette partie à la recherche d'une clé dans un B-arbre.

Question 5 Décrire en français le principe d'un algorithme **récursif** de recherche d'une clé dans un B-arbre.

Question 6 Ecrire la procédure Ada `Recherche(A : in B-Arbre ; C : in Cle ; A' : out B-Arbre ; I : out Integer)` qui implémente cet algorithme.

Cette fonction renvoie à la fois un pointeur vers le nœud contenant la clé recherchée et un entier indiquant la position de la clé dans la liste des clés stockées en ce nœud (exemple : la recherche de la clé 8 dans l'arbre de la figure 1 renvoie un pointeur vers le nœud contenant les clés 7, 8 et 9, ainsi que l'entier 2). Si la clé recherchée n'est pas présente dans l'arbre, la fonction renvoie le pointeur `NULL` ainsi qu'un entier quelconque.

Question 7 Quel est le coût asymptotique en pire cas de cet algorithme, en fonction de n et de d ?
Indication : utilisez la relation (1) de la question 4.

3 Insertion d'une clé

L'insertion d'une clé dans un B-arbre est assez complexe, car il y a de nombreuses propriétés à conserver. La figure 2 montre un exemple de mise à jour d'un B-arbre de degré 3 lors de l'insertion successive de quatre clés.
Comme pour un arbre binaire de recherche, l'insertion se fait en cherchant à partir de la racine la feuille où ajouter la clé, en fonction des valeurs des clés des nœuds traversés. Dans le cas où la feuille où on souhaite insérer la clé est complète, c'est-à-dire qu'elle contient déjà $2d-1$ clés, il faut "séparer" cette feuille en deux. Cette séparation se fait en remontant la clé médiane vers le nœud père puis en remplaçant la feuille par deux feuilles contenant chacune $d-1$ clés (respectivement les plus petites et les plus grandes), voir la figure 3 pour un exemple. La nouvelle clé est ensuite insérée dans la feuille adéquate. Mais le problème peut alors se propager : le nœud père peut devenir complet à son tour. Afin de remédier à ce problème, les nœuds complets sont systématiquement séparés (de la même manière) lors de la traversée à partir de la racine.

La procédure globale peut s'écrire de la manière suivante :

```
procedure Insertion(A: in out B-Arbre; C : in Cle) is
  A': B-Arbre;
begin
  if A.K = 2*D-1 then
    -- Si la racine est un noeud complet, on doit la séparer

    -- Dans ce cas, allocation d'un nouveau noeud A', --
    -- avec k = 0,  EstFeuille = faux et un unique fils qui est A --

    Separation(A,A',1); -- 1 indique que A est le premier fils de A'
    InsertionNoeudIncomplet(A',C);
  else
```

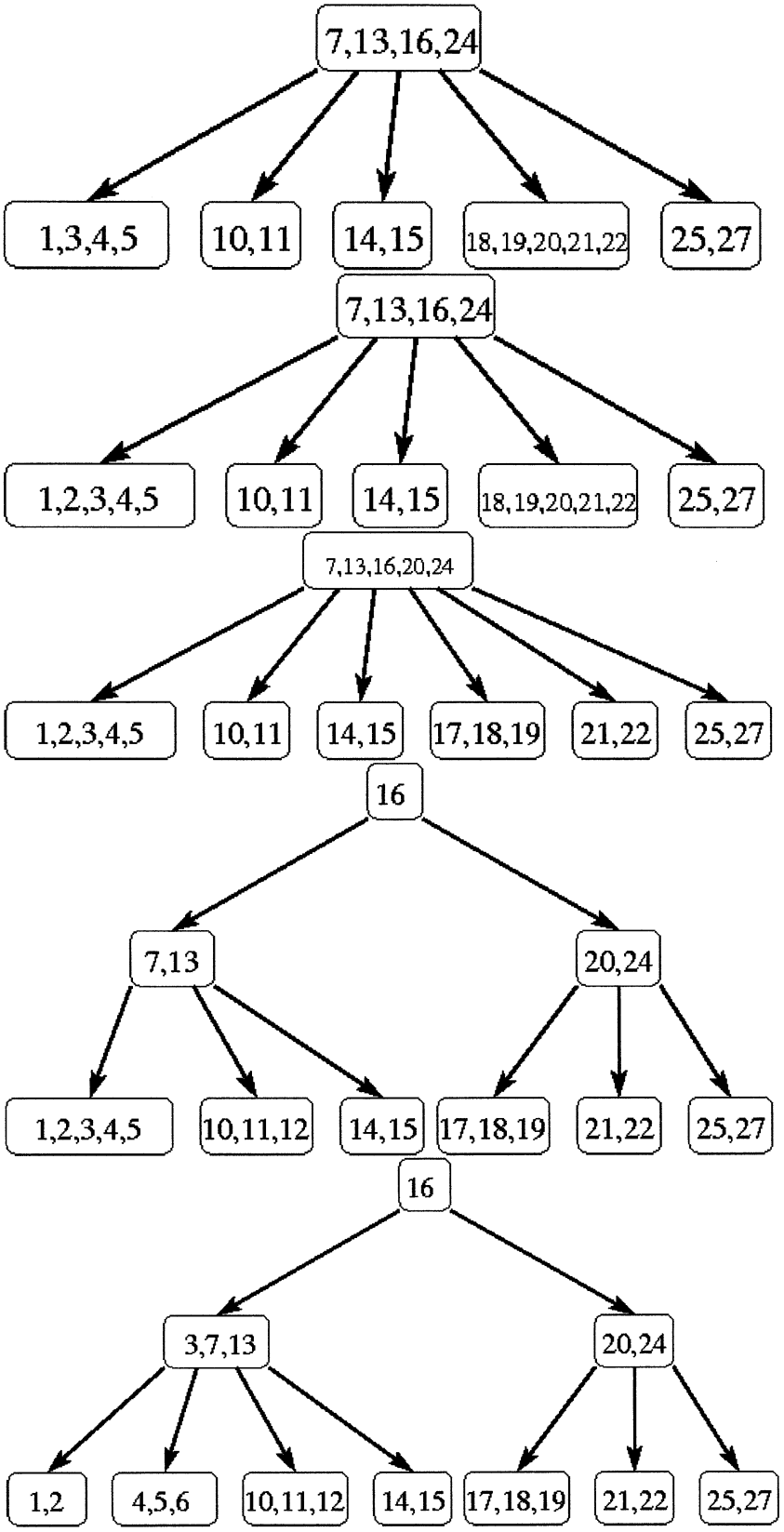

FIG. 2 – Mise à jour d'un B-arbre de degré 3 après insertion successive des clés 2, 17, 12 et 6.

```
    InsertionNoeudIncomplet(A,C);
  end if;
end Insertion;
```

Cette procédure permet de gérer le cas particulier où la racine de l'arbre est complète, et où il faut donc créer une nouvelle racine. La procédure récursive parcourant l'arbre est la procédure `InsertionNoeudIncomplet(A : in out B-Arbre ; C : in Cle)`.

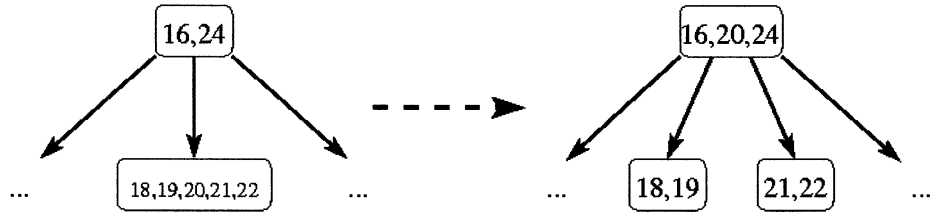

FIG. 3 – Séparation d'un nœud complet.

Question 8 Dessiner les mises à jour d'un B-arbre de degré 2, initialement vide, lors de l'insertion successive des clés $2, 9, 8, 4, 1, 5, 3, 6$ et 7.

Question 9 Ecrire en Ada la procédure `Separation(A,A' : in out B-Arbre ; I : in Integer)` qui sépare un nœud complet selon la technique décrite ci-dessus.

Cette procédure prend en paramètre un pointeur `A` vers le nœud, un pointeur `A'` vers son père et un entier I indiquant la position du nœud parmi les fils de son père (le nœud étudié est le Ième fils de son père). Vous pouvez supposer connue, à condition de bien les définir, toute procédure ou fonction de recopie, d'insertion, de suppression ou de recherche dans une liste ou un tableau.

Question 10 Ecrire en Ada la procédure **récursive** `InsertionNoeudIncomplet(A : in out B-Arbre ; C : in Cle)` qui insère une clé `C` dans un B-arbre `A` de degré `D`. On suppose que le nœud pointé par `A` n'est pas complet. En revanche, les fils de ce nœud peuvent éventuellement être complets.

Vous pouvez supposer connue, à condition de bien les définir, toute procédure ou fonction de recopie, d'insertion, de suppression ou de recherche dans une liste ou un tableau.

Question 11 Quel est le coût asymptotique en pire cas d'une insertion, en fonction de n et de d?
Indication : utilisez la relation (1) de la question 4.

4 Suppression d'une clé

L'algorithme de suppression d'une clé est encore plus complexe et ne sera pas traité dans cet examen.

5 Conclusion

Question 12 A quel type d'arbre binaire de recherche vu en cours les B-arbres vous font-ils penser ? Pourquoi ?

Question 13 Quel(s) avantage(s) voyez-vous à utiliser les B-arbres pour gérer un dictionnaire, plutôt que les différents types d'arbres binaires de recherche vus en cours ?

Exercice 2 : Points les plus proches *(5 pts)*

On se propose de trouver un algorithme rapide pour la recherche des 2 points les plus proches dans un nuage de n points dans le plan. Cet algorithme sera de type "diviser pour régner".

Les points sont triés une fois pour toutes dans deux tableaux, l'un X par abscisse croissante, l'autre Y par ordonnée croissante. A chaque étape sont passés en paramètres de la fonction, les points P, les deux tableaux X et Y de ces points triés respectivement par abscisse et ordonnée croissantes (ceci se fait à partir de la liste préalablement triée et ne nécessite donc pas un nouveau tri).

On partitionne P en deux parties égales P_1 et P_2 par une ligne verticale Λ. On résout sur P_1 et P_2, ce qui donne deux points dans P_1 et une distance δ_1, et deux points dans P_2 et une distance δ_2. Soit $\delta = min\{\delta_1, \delta_2\}$. Soit la distance minimum pour les points de P est δ, soit ce sont deux points à cheval sur la frontière Λ qui donnent la plus courte distance.

On prend une bande de largeur 2δ centrée sur la droite Λ.

Question 1 Montrer que pour chaque point p dans cette bande, le rectangle de hauteur δ et de largeur 2δ centré en p ne peut contenir plus de 6 points de P. Pour chaque point de la bande, combien au maximum de distances faudra-t-il calculer ? Borner en fonction de $|P|$ (nombre de points de P) le temps nécessaire à l'exploration de cette bande.

Question 2 Ecrire en Ada la fonction `ExploreBande(Lambda : in Float ; Delta : in Float) return Float` d'exploration de la bande (`Lambda` représente l'abscisse de Λ). Cette fonction renvoie la distance minimum entre deux points de la bande. `X` et `Y` sont des variables globales.
On pourra utiliser les types suivants :

```
type St_Point ;
type Point is access St_Point ;

type St_Point is record
  Abscisse, Ordonnee: Float ;
  IndX: Natural ; -- indice du point dans X
  IndY: Natural ; -- indice du point dans Y
```

```
end record ;

X,Y: array(1..N) of Point ;
```

Question 3 En déduire la complexité de cet algorithme de calcul des points les plus proches.

Question 4 [**Bonus**] Si on trie dans chaque appel les points, obtient-on la même complexité ? La réponse étant clairement NON, quelle est-elle ? (Pas facile, utiliser le théorème magique de l'appendice du polycopié).