Département d'informatique
Examen final
Hiver 2016

# IFT339

Structures de données

jeudi 28 avril 2016
9 h à 12 h
Jean Goulet

Cet examen comprend quatre questions. Lisez d'abord tout l'examen, puis répondez directement sur le questionnaire. L'examen sera corrigé sur 100, puis ramené à 40. Donnez le code demandé en C++. Essayez d'écrire le plus lisiblement possible. Ne débrochez pas ce questionnaire. Comme documentation, vous avez droit au **Résumé C++** jaune. Pas de calculatrice. Il y a du papier supplémentaire pour vos brouillons.

---

Identifiez-vous lisiblement ci-dessous :

| | |
|---|---|
| Nom : | |
| Prénom : | |
| Matricule : | |
| Signature : | |

| | | |
|---|---|---|
| 1 | | /26 |
| 2 | | /18 |
| 3 | | /36 |
| 4 | | /20 |
| total | | /100 |

## Question 1 – Éléments techniques divers (26 points)

(a) Combien d'éléments contient au minimum
un arbre AVL (équilibré en hauteur) de hauteur 9? _____

Combien en contient-il au maximum? _____

(b) Pourquoi l'insertion avec indice dans un arbre équilibré en poids ne peut-il jamais être de complexité O(1) amortie? (Elle est toujours O(log n))

(c) Quelle est la complexité algorithmique des opérations suivantes? [O(1), O(1)amortie, O(log n), O(n)]?

L'insertion par valeur dans un arbre AVL _____

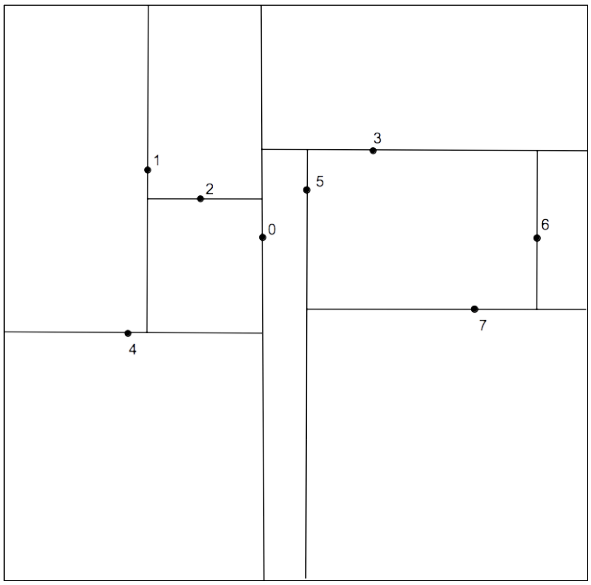
L'insertion avec indice dans une liste à emjambement (skip list)? _____

L'insertion dans un unordered_map (map avec adressage dispersé)? _____

L'insertion à un point quelconque dans un deque? _____

(d) On a construit un arbre KD pour représenter des points dans un plan. Aux niveaux pairs de l'arbre, c'est l'abscisse (x) qui sert à séparer l'ensemble des points en deux. Aux niveaux impairs, c'est l'ordonnée (y). La racine de l'arbre est le nœud numéro 0. Par la suite, une zone vide est identifiée par le nœud 0, comme pour les arbres que vous avez eu à traiter dans le laboratoire 6. Donnez les valeurs KD de tous les nœuds illustrés ici. La racine est au niveau pair.

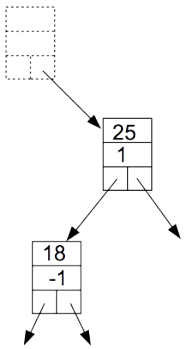
| | <= | > |
|---|---|---|
| 0 | | |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |



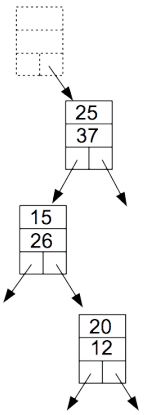

## Question 2 – Les rotations dans les arbres binaires de recherche (18 points)

Une des observations les plus importantes qui ont mené à la construction d'arbres binaires de recherche équilibrés est la capacité d'effectuer des rotations autour de nœuds de l'arbre. Avec une rotation on peut modifier la hauteur totale de l'arbre en temps constant, donc sans avoir à manipuler l'intérieur des sous-arbres. Une telle rotation est possible parce que le sous-arbre au centre de la rotation peut passer de l'autre côté de l'arbre car tous ses éléments se situent entre les deux éléments qui font la rotation. Ces rotations sont utiles dans les arbres AVL (a), les arbres rouges et noirs (b) et les arbres équilibrés en poids (c). Voyez au besoin la question 3 pour des détails sur la représentation graphique de ces arbres. Effectuez une rotation de la gauche vers la droite autour du nœud de valeur 25 dans chacun des trois cas suivants :

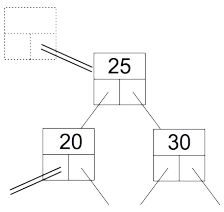
(a)




(c)




(b)



## Question 3 – L'équilibre des arbres de recherche (36 points)

Dans chacun des cas ci-dessous effectuez l'opération demandée sur la structure qui vous est donnée et dessinez l'arbre résultant. S'il y a lieu, montrez les étapes intermédiaires. Avant de commencer un problème, essayez d'identifier le plus d'information possible sur les sous-arbres que vous ne voyez pas. **Rappels:**

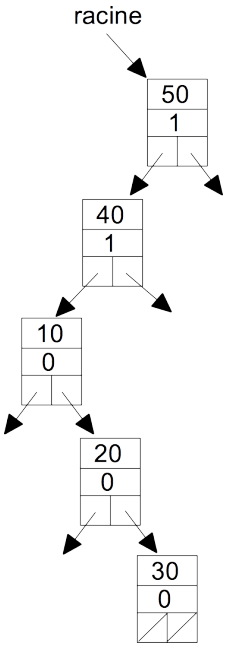
Dans un AVL, l'indice est la hauteur du sous-arbre de gauche moins la hauteur du sous-arbre de droite.

Dans un arbre équilibré en poids, on ne s'occupe pas de l'équilibre d'un nœud dont le poids est inférieur ou égal à 3, et on doit rééquilibrer un nœud dont un des sous-arbres a un poids strictement supérieur à trois fois celui de l'autre. On doit faire une double rotation si une rotation unique cause un déséquilibre dans l'autre sens.

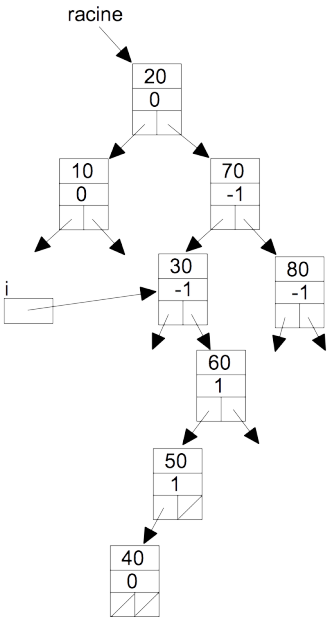
Dans les arbres rouges et noirs, les arcs rouges sont représentés par des traits doubles et les pointeurs nuls sont des arcs noirs.

Dans un monceau, un élément est supérieur ou égal à ses enfants.

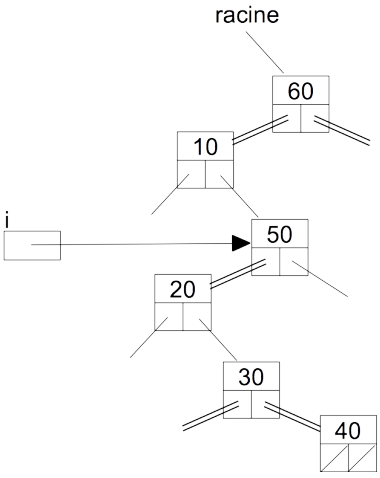
**A-** Ajoutez 25 dans l'arbre AVL suivant:



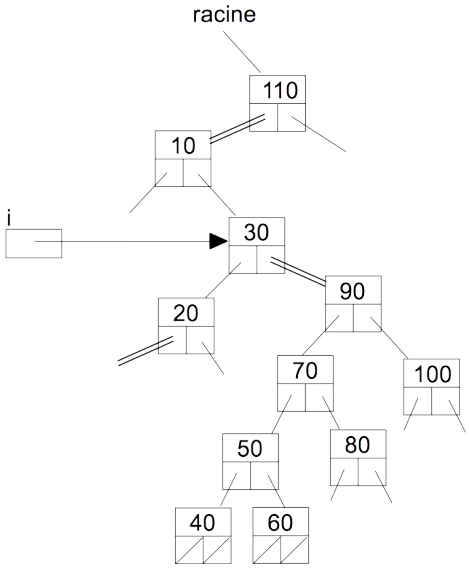
**B-** Enlevez l'élément identifié par l'itérateur i dans l'arbre AVL suivant:



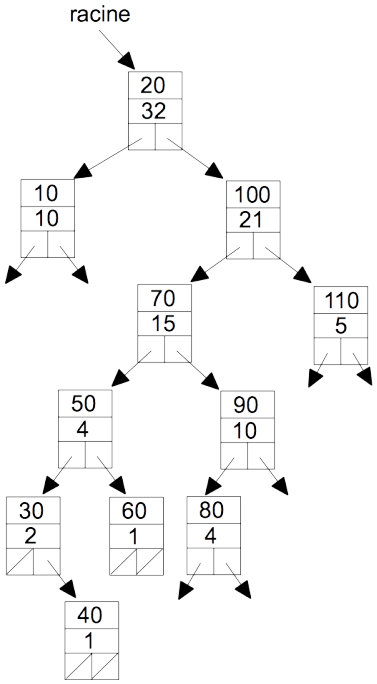
**C-** Ajoutez 45 dans l'arbre rouge et noir suivant en temps O(1) amorti en utilisant comme indice l'itérateur i



**D-** Enlever dans l'arbre rouge et noir suivant l'élément identifié par l'itérateur i:



**F-** Ajouter 65 dans l'arbre équilibré en poids suivant:




**F-** Ajoutez 35 dans le monceau suivant:

40 29 38 21 23 25 35 12 19 17 11 14 21 19 17 7 9 2 18

## Question 4 – L'itération dans les arbres binaires de recherche (20 points)

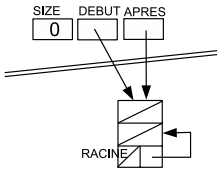
Il est très important que l'on puisse itérer sur un conteneur en temps linéaire (O(n)). C'est toujours possible de le faire, quelle que soit la structure. Il est plus difficile de s'assurer de pouvoir toujours faire l'opération de mouvement en inordre en temps mieux que O(log n). On sait que dans les arbres binaires de recherche, on peut faire cette opération en temps constant amorti si chaque nœud contient un lien vers son parent. Codez la fonction `operator--` pour les maps représentés par des arbres rouges et noirs, avec la représentation illustrée ci-contre (la même que dans le laboratoire 5). Cette fonction amène le pointeur de l'itérateur au précédent en inordre dans l'arbre. Comme cette classe est amie de la classe map, on a le droit d'accéder directement à la représentation des arbres et des nœuds d'arbres. Vous pouvez utiliser toutes les fonctions de base des arbres rouges et noirs, comme estNul(), estRouge(), etc. Un itérateur ne contient qu'un pointeur vers un nœud de l'arbre. La position de la fin est un pointeur sur la cellule supplémentaire (celle identifiée par le pointeur APRES). N'oubliez pas qu'une opération illégale ne doit pas passer inaperçue. Voici les identificateurs dont vous pourriez avoir besoin :

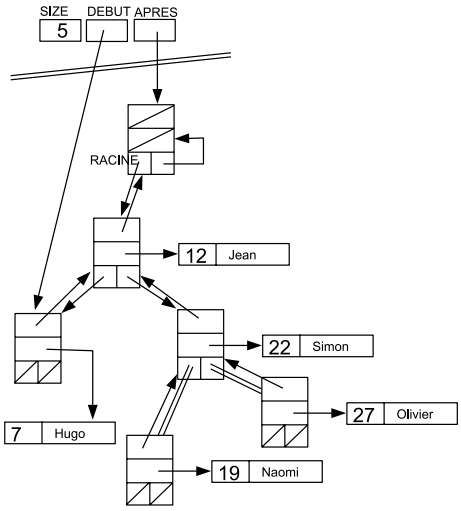
```
template <typename Tclef,typename Tvaleur> class map{
private:
    pointeur APRES,DEBUT;  size_t SIZE;
    ...

struct noeud{
    PAIR* CONTENU;
    pointeur PARENT,GAUCHE,DROITE;
    ...

struct pointeur{
    noeud* REP; couleur COULEUR;
    ...

class iterator{
private:
    pointeur lePointeur;
public:
    iterator& operator--();    //--i
    ...
};
```




```
template <typename Tclef, typename Tvaleur>
typename map<Tclef,Tvaleur>::iterator& map<Tclef,Tvaleur>::iterator::operator--(){
```