Département d'informatique
Examen final
Automne 2015

IFT339

Structures de données

jeudi 17 décembre 2015
9 h à 12 h
Jean Goulet

Cet examen comprend quatre questions. Lisez donc d'abord tout l'examen, puis répondez directement sur le questionnaire. Il y a en général plus d'espace que nécessaire pour vos réponses. L'examen sera corrigé sur 100, puis ramené à 40. Donnez le code demandé en C++. Essayez d'écrire le plus lisiblement possible. Quand on code avec des pointeurs, c'est toujours une bonne idée de commenter le code suffisamment pour que le correcteur comprenne ce que vous vouliez faire... et il faut utiliser avec parcimonie les doubles déréférences de pointeurs!

Ne débrochez pas ce questionnaire. Comme documentation, vous avez droit à la documentation technique, soit le **Résumé C++** jaune, plus une feuille recto-verso de notes personnelles. Il y a du papier supplémentaire pour vos brouillons.

Essayez d'être synthétique en n'utilisant pas plus d'espace que ce qui est prévu pour chaque réponse.

Identifiez-vous lisiblement ci-dessous :

Nom :	
Prénom :	
Matricule :	
Signature :	

1		/18
2		/20
3		/20
4		/42
total		/100

Question 1 – Questions techniques diverses (18 points)

a) Quel lien de complexité y a-t-il entre les fonctions push_back du vector et l'insertion avec indice dans un set de la SL? (EXPLIQUEZ)

b) Pourquoi peut-on implanter les conteneurs associatifs (set et map) de la bibliothèque normalisée (SL) de C++ avec un arbre AVL ou un Rouge-et-noir mais pas avec un arbre équilibré en poids?

c) Voici le parcours en préordre d'un arbre AVL. Dessinez cet arbre en donnant aussi les indices d'équilibre.

3 1 2 7 5 4 6 8

d) Quelle est la complexité en temps de l'algorithme suivant, qui fusionne les ensembles UN et DEUX de la SL pour mettre le résultat dans l'ensemble TROIS. (EXPLIQUEZ)

```
1- Copier DEUX dans TROIS
2- Pour chaque élément de UN
     Insérer une copie de cet élément dans TROIS
```

Question 2 – Manipuler un arbre de recherche (20 points)

Il existe plusieurs fonctions en lien avec la localisation dans les conteneurs de la SL (find, lower_bound, upper_bound, count, etc.). On sait que ces conteneurs sont très souvent implantés par des arbres binaires de recherche. On vous donne ici le code de base d'une classe set implantée par un arbre AVL. Dans un tel arbre, la différence des hauteurs des deux sous-arbres d'un nœud donné n'est jamais supérieure à 1. La différence des deux hauteurs est un indice d'équilibre présent dans chaque nœud. Cet indice nous donne la hauteur du sous-arbre de GAUCHE moins la hauteur du sous-arbre de DROITE. Ces indices sont suffisants pour la gestion de l'équilibre de l'arbre. Ils peuvent aussi être utiles pour déterminer en temps O(log n) la hauteur totale de l'arbre. On sait que la hauteur d'un nœud est 1 de plus que la hauteur du plus haut de ses deux sous-arbres. Normalement, pour connaître la hauteur d'un arbre binaire, il faut un temps O(n), car on doit aller explorer toutes les branches. Mais avec un AVL, comme on connaît les différences de hauteurs, il n'y a toujours qu'une seule branche à explorer.

```
template <typename TYPE>
class set{
   ...
private:
   struct noeud{
      TYPE CONTENU;
      noeud *GAUCHE,*DROITE;
      int INDICE;
   };
size_t SIZE ;
noeud* RACINE ;
...
public:
   ...
   size_t hauteur()const;
   size_t count(const TYPE&)const;
   ...
};
```

Codez la fonction publique hauteur() qui retourne la hauteur de l'arbre en temps O(log n).

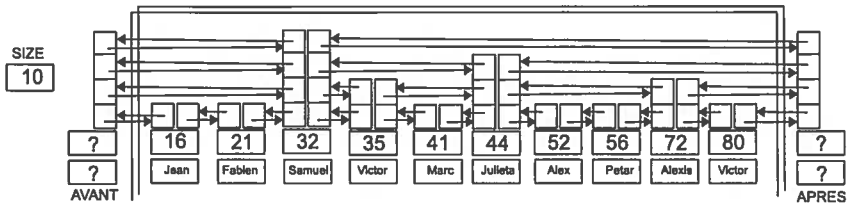
```
template <typename TYPE>
size_t set<TYPE>::hauteur()const{
```

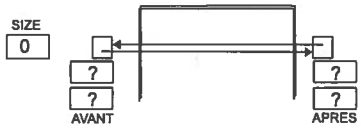
Codez aussi la fonction count, qui retourne 1 si son paramètre est présent dans l'arbre et 0 sinon:

```
template <typename TYPE>
size_t set<TYPE>::count(const TYPE& x)const{
```

Question 3 – Implanter un map à l'aide d'une skip list (20 points)

On a implanté en laboratoire l'insertion et l'élimination dans un ensemble représenté par une skip list, dont vous avez une partie du code ici. On aurait pu aussi faire un map, puisqu'un map est simplement un ensemble de paires d'objets (une clef et une valeur qui lui est associée). On désire implanter la fonction upper_bound. On voit ci-dessous une illustration d'un map<int,string> contenant 10 éléments et une autre d'un map vide. Un itérateur est simplement représenté par un pointeur de cellule.





Voici le code de base de la classe map utilisant une telle structure.

```
template <typename Tclef,typename Tvaleur>
class map{
private:
   struct cellule{
      Tclef CLEF;
      Tvaleur VALEUR ;
      vector<cellule*> PREC,SUIV;
      cellule(T=T(),cellule* =nullptr,cellule* =nullptr);
      };
   size_t SIZE;
   cellule AVANT,APRES;

   cellule* insert(const T&);
   ...
public:
   class iterator;

   iterator upper_bound(const Tclef&)const;
   ...
};

template <typename Tclef,typename Tvaleur>
class map<Tclef,Tvaleur>::iterator{
private:
   cellule* POINTEUR;
public:
   friend class map<Tclef,Tvaleur>;
   iterator(cellule*p=nullptr):POINTEUR(p){}
   const T& operator*()const{return POINTEUR->CONTENU;}
   iterator operator++();    //++i
   iterator operator++(int);  //i++
   iterator operator--();    //--i
   iterator operator--(int);  //i--
   bool operator==(const iterator&i2)const{return POINTEUR==i2.POINTEUR;}
   bool operator!=(const iterator&i2)const{return POINTEUR!=i2.POINTEUR;}
};
```

Codez la fonction upper_bound. Cette fonction est assez semblable à la fonction lower_bound que vous avez codée en laboratoire. Elle retourne un itérateur donnant la position du premier élément strictement supérieur à celui recherché. Elle prend bien sûr un temps O(log n) où n est le nombre d'éléments dans le map.

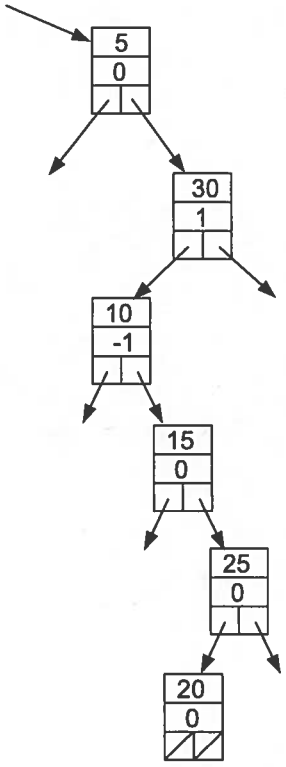
```
template <typename Tclef,typename Tvaleur>
typename map<Tclef,Tvaleur>::iterator
    map<Tclef,Tvaleur>::upper_bound(const Tclef& x)const{
```

Question 4 – L'équilibre des arbres de recherche (40 points)

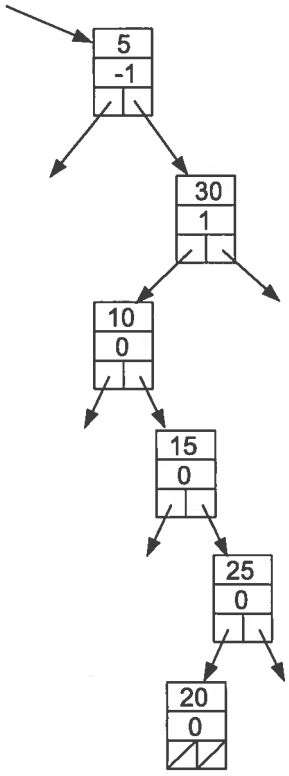
Dans chacun des cas ci-dessous, effectuez l'opération demandée sur la structure qui vous est donnée et dessinez l'arbre résultant. Avant de commencer un problème, essayez d'identifier le plus d'information possible sur les sous-arbres que vous ne voyez pas. **Rappels:**

- 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;
- Dans les arbres rouges et noirs, les arcs rouges sont représentés par des traits doubles et les feuilles auxquelles on accède par un arc noir sont des 2-nœuds (un pointeur nul est un arc noir);
- Dans un B-tree, l'ordre indique le nombre maximal d'enfants que peut avoir un nœud, donc 1 de plus que le nombre d'éléments que peut contenir ce nœud;
- MONTREZ les étapes intermédiaires.

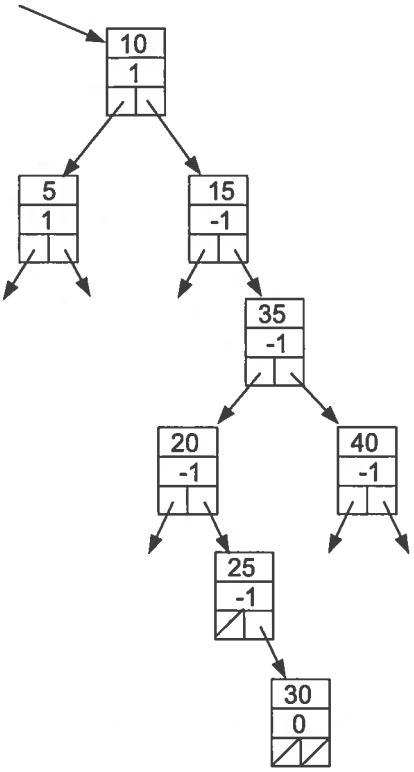
a) Ajoutez 22 dans l'arbre AVL suivant :



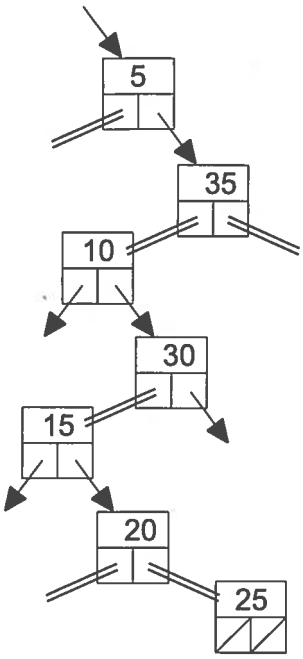
b) Ajoutez 22 dans l'arbre AVL suivant :



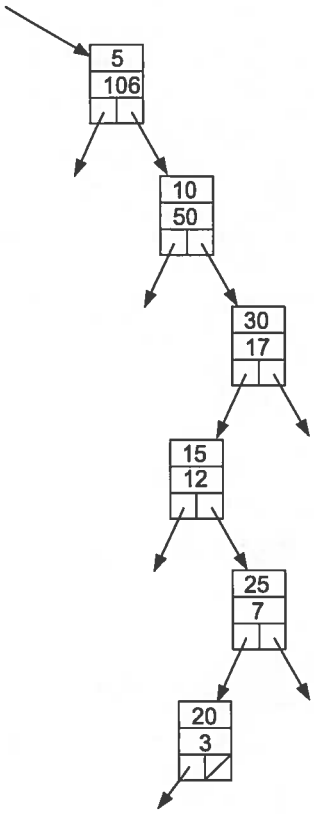
c) Enlevez 35 dans l'arbre AVL suivant :



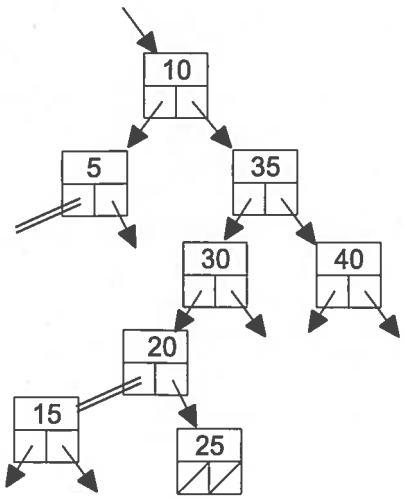
d) Ajoutez 22 dans l'arbre rouge et noir suivant :



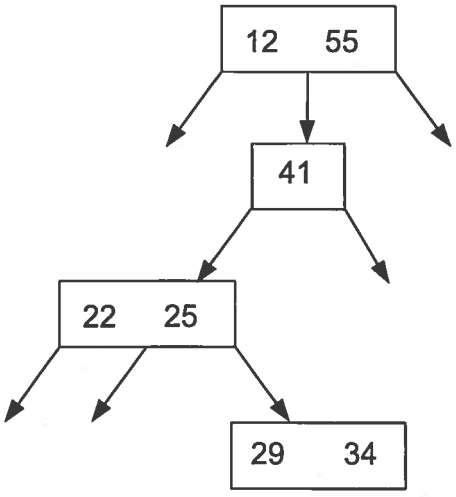
e) Ajoutez 22 dans l'arbre équilibré en poids suivant (chaque noeud contient d'abord une valeur, puis un poids) :



f) Enlevez 30 dans l'arbre rouge et noir suivant :



g) Ajoutez 28 dans le B-tree d'ordre 3 (un arbre-2-3) suivant :



Fin de l'examen (APRÈS la dernière question!)
(Utilisez les feuilles fournies à part pour vos brouillons)