Université de Sherbrooke
Département d'informatique

IFT339
Structures de données

Hiver 2026

Examen Final Type

Professeure: Date
Aïda Ouangraoua Durée : 3 h 00

Documentation permise : une feuille de notes recto-verso

Question 1 : 20 points
Question 2 : 20 points
Question 3 : 40 points
Question 4 : 20 points

Total: 100 points

NOM : _____.

PRÉNOM : _____.

CIP : _____.

SIGNATURE : _____.

Question 1 Questions théoriques diverses (20 points)

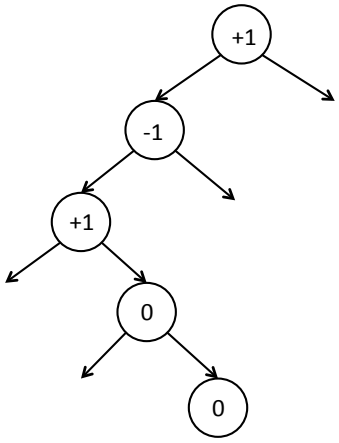
a) Complexité des opérations de différents conteneurs

Quelle est la complexité en temps dans le pire des cas de la fonction insert avec un iterator en entrée pour :
- le vecteur (vector),
- la liste (List),
- la liste à emjambement (Skip_list)
- l'arbre AVL
- l'arbre AA

b) Nombre d'éléments dans un arbre AVL

Combien d'éléments minimum et maximum peut contenir l'arbre AVL dessiné partiellement ci-dessous ? Dans chaque nœud, l'indice d'équilibre est indiqué.

Min :_____ Max : _____ Configuration avec un minimum d'éléments :

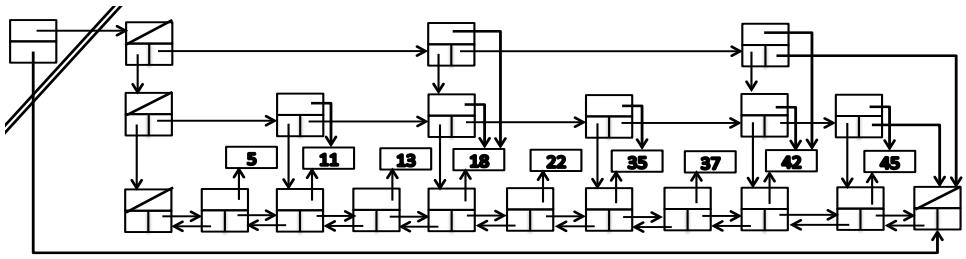


c) Voici le parcours en ordre postfixe d'un arbre AVL. Dessinez cet arbre en donnant les indices d'équilibre de ses nœuds.

0 2 3 1 5 7 6 9 8 4

Question 2 – Implantation d'un conteneur (20 points)

On a implanté en laboratoire une skip list (class set) qui utilise des **vector** de pointeurs de cellules pour stocker les précédents et suivants d'un élément. On vous propose ici une autre représentation telle que chaque élément de la skip list est représenté par une chaîne de cellules de longueur égale au nombre de niveaux de l'élément. On utilise un élément fictif de début ayant un nombre de niveaux maximum et un élément fictif de fin à un seul niveau pour unifier toutes les positions dans la skip list.



Une skip list contient deux attributs : **DEBUT** qui est un pointeur (**cellule***) vers la cellule de niveau supérieur de l'élément fictif de début et **FIN** qui est un pointeur (**cellule***) vers la cellule de l'élément fictif de fin. Une cellule contient trois attributs : **CONTENU** qui est un pointeur (**TYPE***) vers un élément du type TYPE, **PRECINF** qui est un pointeur (**cellule***) vers une cellule de l'élément précédent ou une cellule de niveau inférieur du même élément, et **SUIV** qui est un pointeur (**cellule***) vers une cellule de l'élément suivant. Notez qu'il n'y a pas de pointeur vers les cellules précédentes sauf au niveau 0.

```
template<typename TYPE>
class skip_list{
 private:
    struct cellule{
        TYPE* CONTENU;
        cellule *PREC, *SUIV;
        cellule(TYPE*x=nullptr):CONTENU(x){PRECINF = nullptr ; SUIV=nullptr ;}
        ~cellule();
    };
    cellule* DEBUT, * FIN;
 public:
    bool erase(const TYPE& x);
   …
};
```

Codez la fonction **erase** à partir d'une valeur donnée **x**. La fonction **erase** supprime l'élément de valeur **x** de la skip list si **x** est présent et retourne **True**. Si **x** est absent, elle retourne **False**. Elle prend un temps O(log(n)).

```
template<typename TYPE>
bool skip_list<TYPE>::erase(const TYPE& x) {



}
```

Question 3 – Insertion et supression dans les arbres (40 points)

a)

On vous propose la représentation illustrée ci-dessous pour un arbre binaire de recherche (ABR). Un ABR a un attribut : **RACINE** qui est un pointeur (**nœud***) vers la racine de l'arbre. Un nœud contient trois attributs : **CONTENU** qui est un élément du type TYPE, **GAUCHE** et **DROITE** qui sont des pointeurs (**nœud***) vers les enfants du nœud. Notez qu'il n'y a pas de pointeur vers le parent d'un nœud.

```
template<typename TYPE>
class nœud{
    TYPE CONTENU;
    nœud *GAUCHE, *DROITE;
    nœud(TYPE x):CONTENU(x){ GAUCHE = nullptr ; DROITE = nullptr ;}
};

template<typename TYPE>
class ABR{
 private:
   nœud<TYPE>* RACINE;

   nœud<TYPE>* lower_bound( const TYPE& x );
   …
};
```

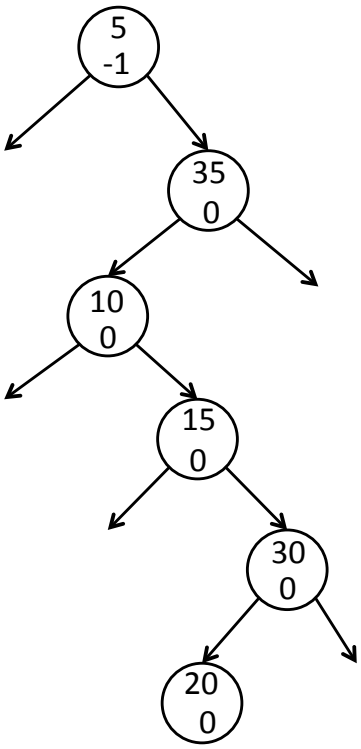
Codez la fonction **lower_bound** qui prend en paramètre une valeur du type TYPE et retourne un pointeur vers le premier nœud de valeur supérieure ou égale à x. Si aucun nœud n'a une valeur supérieure ou égale à x, la fonction retourne **nullptr**.

```
template<typename TYPE>
nœud<TYPE>* ABR<TYPE>:: lower_bound( const TYPE& x ) {




}
```

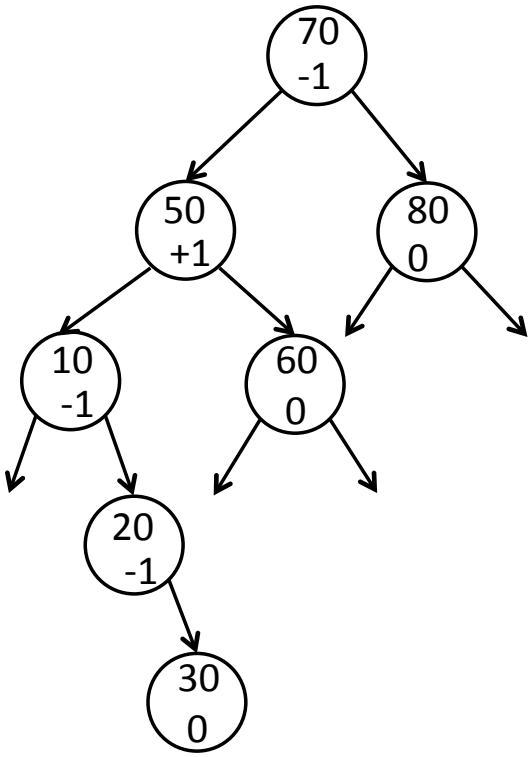
b) Équilibre

Dans chacun des cas ci-après, dessinez le résultat de l'opération demandée sur la structure qui vous est donnée. S'il y a lieu, montrez les étapes intermédiaires.

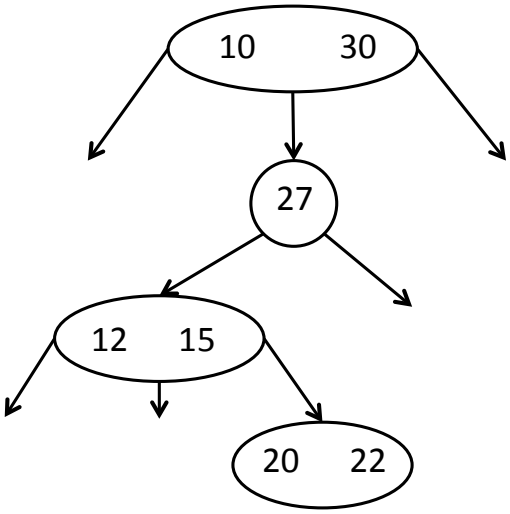
A- Ajoutez l'élément 18 dans l'arbre AVL suivant :



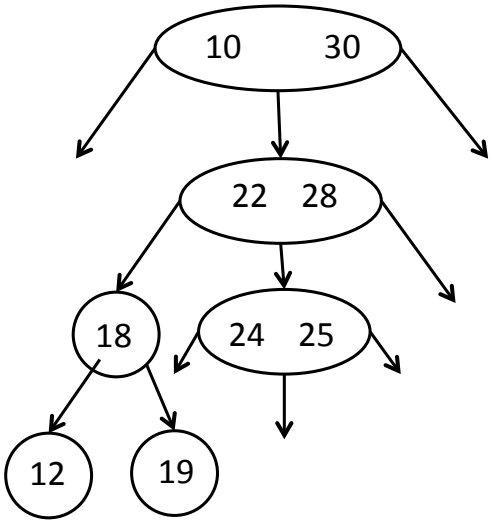
B- Enlevez l'élément 50 dans l'arbre AVL suivant :



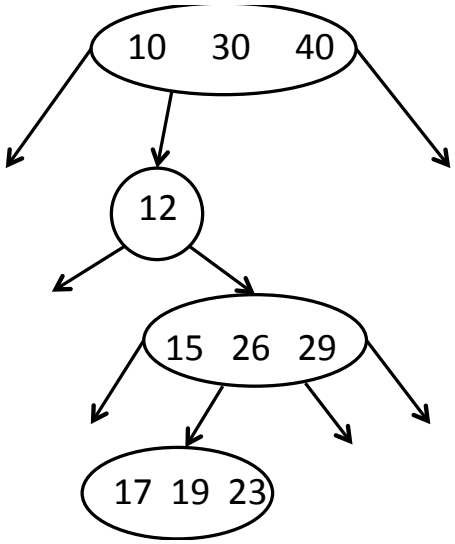
C- Ajoutez l'élément 18 dans le B-arbre d'ordre 3 suivant :



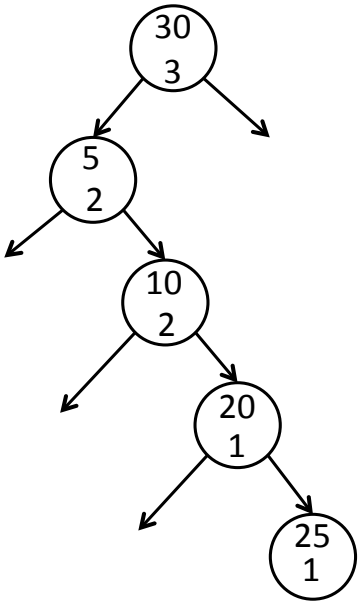
D- Enlevez l'élément 18 dans le B-arbre d'ordre 3 suivant :



E- Ajoutez l'élément 20 dans le B-arbre d'ordre 4 suivant, en utilisant la technique qui consiste à scinder tous les nœuds pleins rencontrés pendant la descente dans l'arbre :



F- Ajoutez l'élément 15 dans le AA-arbre suivant :



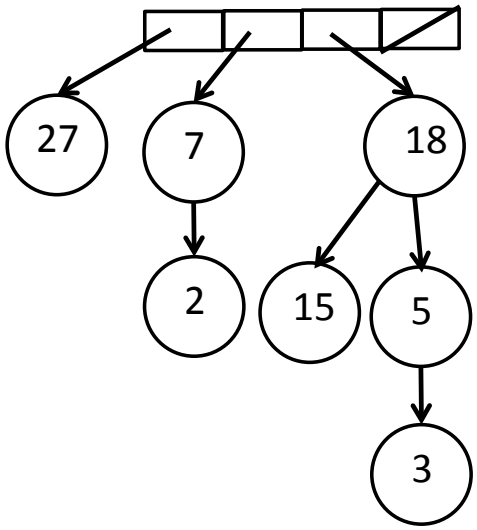
G- Ajoutez l'élément 35 dans le monceau suivant :

44 33 39 17 26 12 28 12 8 4 2 4 2 5 19 5

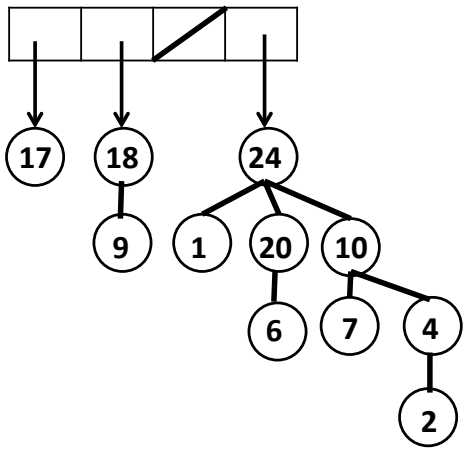
H- Enlevez l'élément 33 dans le monceau représenté dans le tableau suivant :

44 33 39 17 26 12 28 12 8 4 2 4 2 5 19 3

I- Ajoutez l'élément 10 dans le monceau binomial suivant :

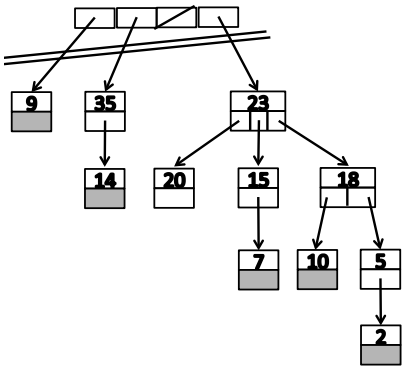


J- Enlevez l'élément maximum dans le monceau binomial suivant :



Question 4 – Implantation d'un monceau binomial (20 points)

On vous propose la représentation illustrée ci-dessous pour un monceau binomial. Un monceau binomial contient un attribut : **ARBRES** qui est un **vector** de pointeurs (**cellule***) vers les racines des arbres binomiaux composant le monceau binomial. Une cellule contient deux attributs : **CONTENU,** qui est un élément du type TYPE et **ENFANTS,** qui est un **vector** de pointeurs (**cellule***) vers les enfants de la cellule.




```
template<typename TYPE>
class monceau_bin{
 private:
    struct cellule{
        TYPE CONTENU;
        vector<cellule*> ENFANTS; //Dimension égale au nombre d'enfants
        cellule(const TYPE& x):CONTENU(x){}
        ~cellule();
    };
    vector<cellule*> ARBRES;
    void fusionner_arbres(cellule*& c1, cellule*& c2);
 public:
   TYPE enlever_max();
…
};
```

Codez la fonction **enlever_max**, en supposant que la fonction **fusionner_arbres** est déjà codée. Cette dernière prend en paramètres passés par référence, deux pointeurs **c**1 et **c**2 vers les cellules racines de deux arbres binomiaux de même ordre et réalise la fusion des deux arbres. Au retour de la fonction, **c**1 contient un pointeur vers la fusion des deux arbres et **c**2 est égal à **nullptr.** La fonction **enlever_max** doit supprimer l'élément maximum du monceau binomial et retourner la valeur de cet élément. Cet élément est forcément la racine d'un des arbres binomiaux composant le monceau binomial. La fonction prend un temps O(log(n)).

```
template<typename TYPE>
TYPE monceau_bin<TYPE>::enlever_max();
```

```
}
```

FIN DE L'EXAMEN