Université Claude Bernard Lyon 1

Année universitaire : 2024 / 2025

LIFAPSD : Algorithmique, Programmation et Structures de Données

Epreuve Anonyme Commune – Session 1
14 janvier 2025
Durée : 1h30

Note : / 20

Nom :
Prénom :
N° d'étudiant :
Signature :

coller ici

Documents et téléphones portables interdits. Répondez dans les emplacements prévus à cet effet. Travaillez au brouillon d'abord de sorte à rendre une copie propre – nous ne pouvons pas vous garantir une copie supplémentaire. **Il se tenu compte de la présentation et de la clarté de vos réponses**.

Exercice 1 : Tri par arbre binaire de recherche (ABR)

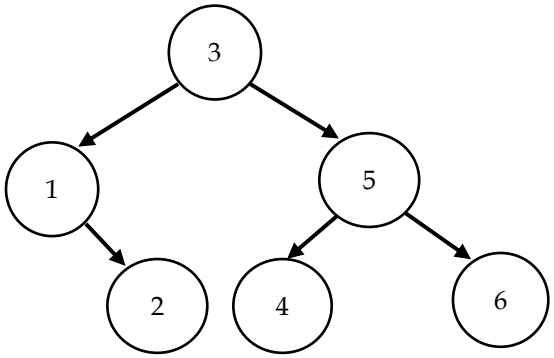
Dans cet exercice nous allons nous intéresser à un algorithme de tri en utilisant un arbre binaire de recherche (ABR). En effet, pour rappel, dans un ABR tous les éléments sont ordonnés : les éléments dans le sous-arbre gauche d'un nœud sont strictement plus petits que ce nœud et les éléments dans le sous-arbre droit sont strictement plus grands.

Pour rappel encore, le **`Noeud`** d'un arbre binaire est défini par une structure contenant trois champs : **`info`** de type **`ElementA`**, **`fg`** et **`fd`** tous les deux de type pointeur sur **`Noeud`**. Et la seule donnée membre de la classe **`Arbre`** est **`adRacine`**, un pointeur sur le **`Noeud`** racine de l'arbre.

Les fonctions membres de la classe **`Arbre`** sont rappelées en annexe.

Comme les éléments de l'ABR sont ordonnés, il suffit de placer les éléments à trier dans l'arbre puis de relire l'arbre dans l'ordre croissant des éléments.

Par exemple, l'ABR suivant :



pourra facilement être utilisé pour récupérer la liste des valeurs triées de 1 à 6.

Question 1 : Donner une séquence possible dans laquelle les valeurs de l'arbre précédent ont été ajoutées à partir d'un arbre vide.

Pour obtenir l'arbre précédent, les valeurs peuvent avoir été ajoutées dans l'ordre : 3 1 2 5 4 6

(Toute réponse ok tant que le 3 est en premier, le 1 avant le 2, le 5 avant le 4 et le 6).

Question 2 : Rappeler le code C++ de la fonction membre **`insererElement`** de la classe **`Arbre`**, vue en CM et TP, plaçant dans l'ABR l'élément passé en paramètre. Si vous avez besoin de créer une fonction intermédiaire, donner également son code.

```
void Arbre::insererElement (ElementA e) {
    insererElementDepuisNoeud(e, adRacine);
}


void Arbre::insererElementDepuisNoeud(ElementA e, Noeud * & n) {
    if (n == nullptr) {
        n = new Noeud;
        n->info = e;
        n->fg = nullptr;
        n->fd = nullptr;
    }
    else  {
        if (estInferieurElementA(e, n->info)) insererElementDepuisNoeud(e, n->fg);
        if (estSuperieurElementA(e, n->info)) insererElementDepuisNoeud(e, n->fd);
        // on ne fait rien si élément déjà présent
    }
}
```

Dans cette UE, nous avons vu 4 parcours d'arbre : le parcours préfixe, le parcours infixe, le parcours postfixe, et le parcours en largeur.

Question 3 : Donner l'ordre de visite des éléments de l'arbre de l'exemple de la première page pour chacun de ces 4 parcours.

Parcours préfixe : 3 1 2 5 4 6

Parcours infixe : 1 2 3 4 5 6

Parcours postfixe : 2 1 4 6 5 3

Parcours en largeur : 3 1 5 2 4 6

Question 4 : Indiquer quel parcours permet de récupérer la séquence triée des valeurs de l'arbre. Justifier brièvement.

Le parcours infixe permet de récupérer la séquence triée. En effet, pour tout nœud dans un ABR les éléments plus petits sont à gauche et plus grands à droite, et dans le parcours infixe on va d'abord à gauche (éléments plus petits) puis on traite l'élément, puis on va à droite (éléments plus grands). On visitera donc tous les éléments dans l'ordre croissant.

Supposons que les éléments à trier sont stockés dans une instance de **`TableauDynamique`** (cf. annexe pour un rappel des fonctions membres de la classe).

Question 5 : Donner le code C++ d'une nouvelle fonction membre **`sauverDansTabDyn`** de la classe **`Arbre`** qui parcourt l'arbre (avec le parcours choisi à la question précédente) et qui retourne l'adresse d'un nouveau tableau dynamique stocké sur le tas contenant les valeurs des informations des nœuds visités. Si vous avez besoin de créer une fonction intermédiaire, donner également son code.

```
TableauDynamique * Arbre::sauverDansTabDyn () const {
    TableauDynamique * tab = new TableauDynamique;
    sauverDansTabDynDepuisNoeud(adRacine, tab);
    return tab;
}

void Arbre::sauverDansTabDynDepuisNoeud (const Noeud * n, TableauDynamique * tab) const {
    if (n != nullptr) {
        sauverDansTabDynDepuisNoeud(n->fg,tab);
        tab->ajouterElement(n->info);
        sauverDansTabDynDepuisNoeud(n->fd,tab);
    }
}
```

Pour trier un tableau dynamique de valeurs, nous allons donc finalement exécuter les étapes suivantes :

- Créer un ABR vide
- Ajouter un à un dans cet arbre tous les éléments du tableau
- Vider le tableau
- Parcourir l'arbre et récupérer un nouveau tableau trié
- Recopier le tableau et libérer les données plus utilisées

Question 6 : Donner le code C++ d'une nouvelle fonction membre **`triABR`** de la classe **`TableauDynamique`** qui trie le tableau dynamique en suivant la méthode ci-dessus.

```
void TableauDynamique::triABR () {
    Arbre ABR;
    for (int i = 0 ; i < taille_utilisee ; i++) {
       ABR.insererElement(ad[i]);
    }
    vider();
    TableauDynamique * tabtrie = ABR.sauverDansTabDyn();
    for (int i = 0 ; i < tabtrie->taille_utilisee ; i++) {
       ajouterElement(tabtrie->valeurIemeElement(i));
    }
    delete tabtrie;
}
```

Question 7 : Rappeler les complexités dans le pire des cas, en notation O, de toutes les fonctions utilisées dans **`triABR`** et en conclure la complexité de **`triABR`**.

Le constructeur par défaut de Arbre a une complexité constante en O(1) car réalise juste une affectation.
La fonction insererElement de Arbre a une complexité en O(n) car dans le pire des cas (arbre dégénéré) il faut ajouter au bout de la branche unique.
La fonction vider de TableauDynamique est en O(1) car simple libération mémoire puis allocation de 1 emplacement.
La fonction sauverDansTabDyn de Arbre est en O(n) car réalise un parcours de tous les nœuds.
La fonction valeurIemeElement de TableauDynamique est en O(1) car accès direct à l'élément via son indice.
La fonction ajouterElement de TableauDynamique a une complexité amortie en O(1) et O(n) dans le pire des cas (cf. cours).

A la fois insererElement, valeurIemeElement et ajouterElement sont réalisés dans une boucle itérant sur les n éléments du tableau.

La complexité totale de triABR est donc en $O(1+n*n+1+n+n*(1+n))=O(2n^2+2n+2)=O(n^2)$ donc quadratique comme le tri par insertion ou sélection.

Question 8 : Compléter le code C++ du programme principal ci-dessous qui trie un tableau dynamique de valeurs entières.

```
#include "TableauDynamique.h"
#include <iostream>
using namespace std;

int main() {

   // création sur la pile d'un tableau dynamique vide nommé tab
   TableauDynamique tab;

   cout << "Ajouts de ";
   // ajout de 10 valeurs entières aléatoires dans le tableau
   for (unsigned int i = 0; i < 10; i++) {
      // tirage d'un entier entre -31 et 48 (inclus)
      ElementTD e = (rand() % 80) - 31;
      // ajout dans le tableau
      tab.ajouterElement(e);
      // affichage de la valeur insérée
      cout << e << " ";
   }
   cout << endl;

   // tri du tableau par ABR
   tab.triABR();

   // affichage du tableau trié
   tab.afficher();

   return 0;
}
```

Exercice 2 : Suppressions d'éléments

Dans cet exercice nous allons nous intéresser à des fonctions membres dont le but est de supprimer un ou plusieurs éléments dans les structures de données **`File`** et **`Pile`** (cf. annexe pour les fonctions membres déjà existantes).

Question 9 : Donner le code C++ d'une nouvelle fonction membre **`depilerJusquA`** de la classe **`Pile`** qui dépile tous les éléments jusqu'à ce que l'élément en paramètre soit le sommet de la pile.

Procédure depilerJusquA (e : ElementP)
Pré-condition : aucune
Post-condition : les éléments jusqu'à e sont dépilés (e exclus). Si e n'est pas dans la pile, la pile est entièrement vidée.
Paramètres en mode donnée : e

```
void Pile::depilerJusquA (ElementP e) {
    while (!estVide() && consulterSommet() != e) depiler();
}
```

Question 10 : Donner le code C++ d'une nouvelle fonction membre **`traiterElement`** de la classe **`File`** qui met les éléments en premières places de la file à la fin et qui supprime l'élément passé en paramètre.

Procédure traiterElement (e : ElementF)
Pré-condition : e est présent dans la file
Post-condition : les éléments apparaissant avant e dans la file sont déplacés à la fin et e est supprimé.
Paramètres en mode donnée : e
Exemple : après un appel sur une file contenant (2,3,1,5,0) avec l'élément 1 en paramètre, la file contient (5,0,2,3). C'est-à-dire que le 0 (premier de la file) et le 5 (deuxième) ont été déplacés en fin de file, et le 1 est supprimé.

```
void File::traiterElement (ElementF e) {
    while (premierDeLaFile() != e) {
       enfiler(premierDeLaFile());
       defiler();
    }
    defiler();
}
```

Annexe : liste des fonctions membres des classes TableauDynamique, Liste, File, Pile et Arbre (constructeurs et destructeurs omis)

Classe TableauDynamique

```
void vider ();
void ajouterElement (ElementTD e);
ElementTD valeurIemeElement (unsigned int indice) const;
void modifierValeurIemeElement (ElementTD e, unsigned int indice);
void afficher () const;
void supprimerElement (unsigned int indice);
void insererElement (ElementTD e, unsigned int indice);
int rechercherElement (ElementTD e) const;
```

Classe Liste

```
void vider ();
bool estVide () const;
ElementL iemeElement (unsigned int indice) const;
void modifierIemeElement (unsigned int indice, ElementL e);
void afficherGaucheDroite () const;
void afficherDroiteGauche () const;
void ajouterEnTete (ElementL e);
void ajouterEnQueue (ElementL e);
void supprimerTete ();
int rechercherElement (ElementL e) const;
void insererElement (ElementL e, unsigned int indice);
void supprimerElement (ElementL e);
```

Classe File

```
void enfiler (ElementF e);
ElementF premierDeLaFile () const;
void defiler ();
bool estVide () const;
unsigned int nbElements () const;
```

Classe Pile

```
void empiler (ElementP e);
ElementP consulterSommet () const;
void depiler ();
bool estVide () const;
```

Classe Arbre

```
bool estVide () const;
void vider ();
void insererElement (ElementA e);
void supprimerElement (ElementA e);
Noeud * rechercherElement (ElementA e) const;
```