TP8 : Arbre binaire de recherche

Les objectifs de ce TP sont les suivants :

- Etre capable d'implémenter les services fondamentaux offerts par un module Arbre Binaire de Recherche,
- Etre capable de mesurer la performance d'un arbre binaire de recherche,
- Comprendre l'influence de la hauteur de l'arbre sur la performance,
- Etre capable d'utiliser des piles et des files pour parcourir itérativement un arbre.

Exercice 1 : Classe Arbre

a. Dans votre répertoire LIFAPSD, créez un répertoire TP8. Placez-y les fichiers ElementA.h, ElementA.cpp et Arbre.h que vous trouverez sur le site de l'UE. Créez un nouveau fichier main.cpp et écrivez-y un programme principal vide pour l'instant. Créez également le Makefile qui permettra de compiler votre projet.

b. Créez le fichier Arbre.cpp en y plaçant les définitions des fonctions membres suivantes :

   - constructeur et destructeur
   - vider et estVide
   - insererElement
   - afficherParcoursInfixe
   - rechercherElement

   Remarque : Pour certaines de ces fonctions membres, vous devrez utiliser une fonction ou procédure auxiliaire, souvent récursive, travaillant sur un sous-arbre, et prenant comme paramètre l'adresse du nœud dans lequel le sous-arbre est enraciné. Ces fonctions auxiliaires restent internes au module Arbre : elles apparaissent donc comme privées dans le .h.

c. Codez ensuite la fonction membre hauteurArbre.

d. Dans le fichier main.cpp, écrivez un programme principal qui insère dans un arbre binaire de recherche 255 entiers aléatoires compris entre 1 et 100 000, puis qui calcule la hauteur de l'arbre. Testez que tout fonctionne bien, en utilisant notamment la procédure d'affichage d'arbre. Vérifiez que vous n'obtenez pas le même arbre si vous exécutez deux fois le programme.

   Rappel : Pour tirer aléatoirement des valeurs, référez-vous à l'annexe B.

e. Complétez le programme principal pour qu'il recherche 100 nombres aléatoires entre 1 et 100 000 dans l'arbre (bien évidemment, parmi ces 100 nombres, certains seront effectivement présents dans l'arbre et d'autres non). En plus de la hauteur de l'arbre, le programme devra afficher le nombre moyen de nœuds visités par opération de recherche.

f. Modifiez le programme principal pour qu'il répète 60 fois le processus complet de création d'arbre + recherche de 100 éléments. Vous devez obtenir comme affichage 60 lignes, avec sur chaque ligne la hauteur et le nombre moyen de nœuds visités par opération de recherche (n'affichez plus les arbres eux-mêmes). Quelle relation constatez-vous entre les deux quantités ?

Vous pouvez également afficher la hauteur et le nombre de nœuds visités pour des arbres qui ont été remplis avec des tableaux triés (produisant des arbres dégénérés) et des tableaux organisés afin de créeer des arbres équilibrés.

Cet exercice devrait vous avoir convaincu de l'intérêt d'équilibrer les arbres. Les algorithmes d'équilibrage sont au programme de LIFAP6.

Exercice 2 : Parcours en largeur (version itérative utilisant une file)

a. Copiez dans votre répertoire TP8 vos fichiers ElementL.h, ElementL.cpp, Liste.h, Liste.cpp, File.h et File.cpp du TP7.

b. Dans Arbre.h, ajoutez la déclaration d'une procédure **itérative** d'affichage **en largeur**. Dans Arbre.cpp, écrivez la définition de cette procédure. Ce code utilisera bien sûr une variable locale de type File. Vous devrez donc ajouter un #include "File.h" au début du fichier Arbre.cpp.

c. Modifiez le programme principal pour qu'il affiche deux fois un arbre rempli : d'abord en utilisant la procédure d'affichage dans l'ordre croissant (parcours infixe), puis en utilisant votre parcours en largeur.

Exercice 3 : Parcours postfixe (version itérative utilisant deux piles)

a. Copiez dans votre répertoire TP8 vos fichiers ElementTD.h, ElementTD.cpp, TableauDynamique.h, TableauDynamique.cpp, Pile.h et Pile.cpp du TP7.

b. Dans Arbre.h, ajoutez la déclaration d'une procédure **itérative** d'affichage **postfixe**. Dans Arbre.cpp, écrivez la définition de cette procédure avec l'algorithme suivant, qui utilise deux piles d'adresses de nœuds, qu'on appellera pileA et pileB :

- on place la racine dans la pileB,
- tant qu'on n'a pas épuisé la pileB, on déplace le sommet de la pileB vers la pileA, et on empile son fils gauche (s'il existe) puis son fils droit (s'il existe) dans la pileB, finTantQue
- une fois la pileB vide, on dépile la pileA jusqu'à la vider, en affichant au fur et à mesure les éléments contenus dans les nœuds.

c. Modifiez le programme principal pour qu'il affiche aussi le contenu de l'arbre par ce troisième parcours.