Algorithmique et structures de données

TD / TP 8

C. Sempé

Les objectifs de cette séance de TD / TP sont :

- Concevoir puis utiliser les Arbres Binaires de recherche
- Utiliser la programmation récursive

TD

1.Conception d'un arbre binaire de recherche.

On reprend l'arbre binaire du dernier TD.

- Quelle est la différence entre un arbre binaire et un arbre binaire de recherche ?

Nous avons vu en cours comment insérer un nouvel élément dans un ABR.
La méthode ***insereTo()*** se charge de placer correctement la valeur selon la relation d'ordre de l'arbre.

2.Méthode insereTo

- Ré-écrire *InsereTo* sans l'aide du cours. (rappel : absence de doublon)

3.Méthode de recherche

- Ecrivez la méthode *rechercherABR* en respectant la relation d'ordre.
  public boolean rechercherABR (Integer element)

4.Méthode Supprimer.

- Quelle stratégie allez-vous utiliser pour supprimer un élément ?
- Ecrire la méthode *Supprimer* (Integer element).

  Rmq : intValeur() retourne l'entier contenu dans un objet de type Integer.
  V1.compareTo(V2) compare deux Integer avec leurs valeurs numériques. Elle retourne un entier >0 si V1>V2 ; =0 si V1=V2 ; <0 si V1<V2

5.Construction d'un arbre équilibré.

- Faites la méthode récursive *arbreBREnTab* qui permet de placer les éléments d'un ArbreBR dans un ArrayList() en ordre croissant.
  public void arbreBREnTab (ArrayList t)

- Faites un nouveau constructeur *ArbreBRCons* qui va construire un ArbreBR équilibré à partir d'un tableau trié dans l'ordre croissant.

  public ArbreBRCons(ArrayList t, int debut, int fin)
  ☞ On applique la stratégie du QS ou de la dichotomie.

  Exemple sur des entiers :

2	5	9	14	15	19
debut		milieu			fin

  On construit un ArbreBR en prenant l'élément de l'indice milieu.
  A sa gauche il y aura un ArbreBR du sous tableau inférieur à milieu.
  A sa droite il y aura un ArbreBR du sous tableau supérieur à milieu.

Deux cas particuliers
- debut=fin
- debut= indice et fin = indice+1

TP

Mise au point de la classe ArbreBR avec des entiers.

- Ouvrez le projet ArbreBREntier

- Lancez le programme puis appuyez sur bouton *exemple*.
  Comme pour le TP précédent, on visualise l'arbreBR construit dans Fenetre.

- Ajoutez des valeurs à l'arbre. Supprimez des valeurs. Regardez le code dans la classe ArbreBR, c'est ce que vous avez étudié en TD.

- Concevez le constructeur *ArbreBRCons* qui à partir d'un ArrayList trié de valeurs permet de construire un arbre équilibré. Un bouton *Equilibrer* vous permettra de tester et d'équilibrer l'arbre à l'écran.
  public ArbreBRCons(ArrayList t, int debut, int fin)

Statistiques jeux vidéos

- Ouvrez le projet *ArbreBRFiche*
- Ouvrer le fichier *Basejeuvideos.txt*. On va charger ces données puis les manipuler avec un ArbreBR.
- Copiez / Collez le fichier *ArbreBR,java* mis au point précédemment dans le dossier src du projet.
- Adaptez le fichier pour qu'il puisse manipuler des objets *Fiche*.
- Faire un test élémentaire dans le *main* afin de valider la création d'une fiche puis l'insertion dans un arbreBR.

- *ChargementDonnees* vous permet de récupérer un *ArrayList* de Fiche à partir du fichier *txt*.
- Créez un ArbreBR en insérant toutes les fiches du *ArrayList*.
- Affichez l'arbre. Quelle relation d'ordre a été choisie pour classer les fiches ?

Equilibre de l'arbre

- Faire rechercherABR qui retourne en combien de fois elle a trouvé l'élément. Elle retourne -1 si elle ne l'a pas trouvé.
  public int rechercherABR (Fiche val, int compteur)
- Testez la méthode en recherchant la fiche de l'index 49 du ArrayList. Pourquoi elle la trouve en 49 fois ?
- Équilibrez l'arbre.
- Refaite le test de la méthode.

Notion de référence

- Levez les commentaires sur le bloc de code "Notion de référence"
- Lancez le test, le numéro hashcode est "la plaque d'immatriculation unique" de l'objet. Comment interpréter le résultat affiché ?

Éviter les redondances

Le *equals* dans Fiche permet de vérifier tous les attributs afin de détecter une égalité de données.

- Levez les commentaires sur le bloc de code "Doit-on insérer un élément déjà présent ?"
- Comprenez les lignes de code et le résultat obtenu.
  rechercheRef permet de rechercher si un élément est déjà dans l'arbre et vous fournit la référence de l'objet qu'il a en stock et que vous pouvez utiliser à la place de l'élément = dictionnaire.

Quel intérêt ?
Si vous utilisez une Fiche dans un main qui est déjà dans le dico alors il ne faut pas réserver de la place inutilement pour une Fiche déjà en mémoire RAM et présente dans le dictionnaire.

Nouvelle relation d'ordre

Faite une copie du projet dans *NetBeans : Clic Droit sur le projet / Copie*
Modifiez cette copie du projet afin d'avoir une relation d'ordre basée *sur les années*.
Testez avec l'affichage de l'arbre, vous devriez avoir un ordre croissant par année.

Comment intégrer les deux relations d'ordre dans le même projet afin d'avoir deux arbresBR selon deux relations d'ordre ? Pas simple car les objet sont typés. Tout se passe avec le compareTo …
Voir le projet *ArbreBRFicheMultiCriteres* pour une possibilité imparfaite.

Les relations d'ordre sont difficiles à implémenter de multiple fois, il existe donc des techniques qui permettent de trier selon une relation d'ordre des objets dans des collections en Java : *Collections, Comparator* . Voir lien sur Moodle