NF16
TP5

Arbres

Printemps 2012

Objectif :

L'objectif de ce TP est de se familiariser avec la notion d'**Arbre Binaire de Recherche (ABR)** : création d'un arbre, insertion de nœuds, suppression de nœuds, recherche d'information et finalement la programmation d'une petite application de gestion utilisant les ABR.

Arbre binaire de recherche : Introduction

On appelle arbre binaire de recherche (ABR) une structure de données qui permet de ranger et de retrouver efficacement des valeurs ordonnées. C'est une structure qui possède la propriété fondamentale suivante :

- tous les nœuds du sous-arbre gauche d'un nœud de l'arbre ont une valeur inferieure à la sienne.
- tous les nœuds du sous-arbre droit d'un nœud de l'arbre ont une valeur supérieure à la sienne.

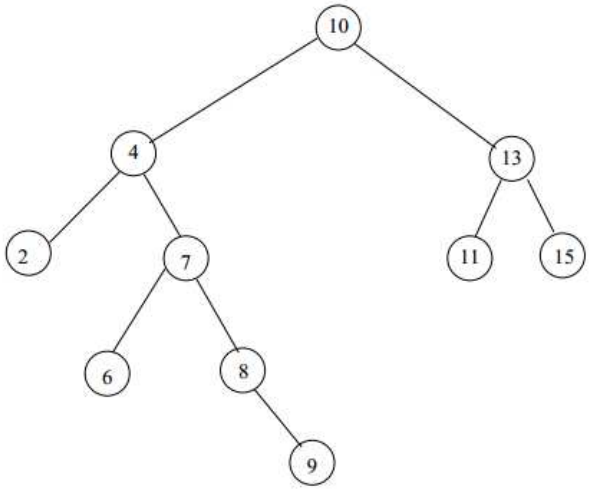

FIGURE 1 ARBRE BINAIRE DE RECHERCHE

Chaque nœud stocke une information appelée valeur ou clé du nœud. Le fils de gauche, s'il existe, porte toujours une valeur inférieure à celle de son père, celui de droite une valeur supérieure. Ainsi, tous les nœuds à gauche d'un nœud valant x portent des valeurs inférieures à x et ceux de droite des valeurs supérieures à x.

Arbres

Printemps 2012

Application :

Il s'agit d'une application pour la gestion complète pour les petites et moyennes entreprises : gestion de la paie et des ressources humaines. Pour cela, un logiciel est optimisé pour gérer ces deux domaines. Ce logiciel possède une base de données contenant toutes les informations concernant un employé à savoir :

- Identifiant unique ;
- Nom et prénom ;
- Sa fonction dans l'entreprise ;
- Son âge ;
- Son salaire ;

Après une analyse, nous avons constaté que la majorité des clients désirent voir les employés triés selon leur salaire, nous avons donc décidé d'utiliser un Arbre Binaire de Recherche (ABR) pour stocker toutes les informations des employés. Chaque nœud de l'arbre contient une liste de pointeurs vers des employés qui ont le même salaire. La clé d'un nœud est donc le salaire des employés. Nous supposons que les salaires ont tous des valeurs entières. La liste des employés ayant le même salaire est représentée par une pile.

Il y a deux structures de Pile. La première structure de pile est utilisée pour stocker des pointeurs vers des employés ayant le même salaire. La deuxième structure de pile stockant des pointeurs vers les nœuds de l'arbre sera utilisée dans le parcours de l'arbre en profondeur.

Structures de données :

```
typedef struct employe{
        int id;                         // l'identifiant de l'employé. Ce champ est unique
        char nom[MAX_STR];              // le nom de l'employé
        char fct[MAX_STR];              // la fonction de l'employé
        int age;
        int salaire;
} Employe;
typedef Employe * pEmploye;

typedef struct pile{
        pEmploye   tabEmploye [MAX_SIZE_PILE];  //tableau des pointeurs pEmploye
int iSommet;
} Pile;
typedef Pile * pPile;
```

```
typedef struct noeud{
        pPile pileEmploye;    //pointeur vers une structure contenant des employés ayant le même
salaire
        int salaire;
        struct noeud * pere;
        struct noeud * gch;
        struct noeud * drt;
} Noeud;
typedef Noeud * pNoeud;

typedef struct pileN{
        pNoeud tabNoeud[MAX_SIZE_PILE];  //tableau des pointeurs pNoeud
        int iSommet;
} PileN;
typedef PileN * pPileN;
```

Fonctions à réaliser :

1. *pNoeud nouveauNoeud(int salaire);*
Cette fonction alloue la mémoire pour un nouveau nœud et l'initialise par le *salaire* passé en paramètre. Elle initialise également sa pile. Les autres champs sont initialisés à NULL. La fonction retourne l'adresse du nœud créé ou NULL en cas d'échec.

2. *pEmploye nouveauEmploye(int num, char * nom, char * fonction, int age, int salaire);*
Cette fonction alloue la mémoire nécessaire à un nouvel employé, l'initialise avec les valeurs passées en paramètres et retourne son adresse. La fonction retourne NULL en cas d'échec.

3. *pNoeud chercherEmploye(pNoeud arbre, int id);*
Cette fonction vérifie l'existence d'un employé dont le *identifiant* est passé en paramètre. Si l'employé existe, elle retourne le pointeur vers le nœud contenant cet employé. Sinon, elle retourne NULL.

4. *pNoeud insererEmploye(pNoeud arbre, pEmploye employe);*
Cette fonction prend un *employe* dont l'adresse est passée en paramètre et l'insère dans l'*arbre*. Si le numéro de l'employé existe déjà, l'employé ne sera pas inséré. L'insertion de l'employé peut occasionner la création d'un nœud. La fonction retourne la racine de l'arbre modifié.

5. *pNoeud supprimerNoeud(pNoeud arbre, pNoeud noeud);*
Cette fonction supprime de l'*arbre* un *nœud* dont l'adresse est passée en paramètre. La fonction retourne la racine de l'arbre modifié. **Toutes les mémoires allouées pour sa pile d'employés seront également libérées.**

6. *pNoeud supprimerEmploye(pNoeud arbre, int id);*

Cette fonction cherche et supprime de l'*arbre* un employé dont l'*identifiant* est passé en paramètre. La fonction retourne la racine de l'arbre modifiée. **Si la pile d'un nœud devient vide après la suppression de l'employé, il faut supprimer le nœud.**

7. *pNoeud modifierEmploye(pNoeud arbre, int id, char * nom, char * fonction, int age, int salaire) ;*
Cette fonction cherche et modifie les informations d'un employé dont l'*identifiant* est passé en paramètre. L'identifiant de l'employé ne sera pas modifié. La fonction retourne la racine de l'arbre modifié.

8. *void afficherEmployes(pNoeud arbre);*
Cette fonction affiche tous les employés par ordre croissant des salaires. **L'ordre d'affichage n'est pas important entre les employés ayant le même salaire.**
Il faudra utiliser la version **non récursive** où la structure pileN sera utilisée.

Exemple d'affichage :

```
-------------------------------------------------------------------------------
| Identifiant |   Nom     | Fonction        | Age    |      Salaire   |
-------------------------------------------------------------------------------
| 1           |    Bruno  | chef de projet  |      35  |        4500 |
| 2           | Thomas    | Manageur        |      45  |        5500 |
-------------------------------------------------------------------------------
```

9. *void supprimerArbre(pNoeud arbre);*
Cette fonction supprime l'arbre et libère toutes les mémoires utilisées.

10. *pNoeud lireFichier(pNoeud arbre, char * f);*
Cette fonction sert à charger des employés depuis un fichier dont le nom est spécifié en paramètre et les insère dans un arbre binaire de recherche. La fonction retourne la racine de l'arbre modifié.

Dans le fichier, chaque ligne contient les caractéristiques d'un employé : ID, Nom, Fonction, Age et Salaire. Les champs sont séparés par une tabulation.

ID	NOM	FONCTION	AGE	SALAIRE
1	Bruno	chef de projet	35	4500
2	Thomas	Manager	45	5500

INTERFACE

Les fonctions qui doivent être présentées sont :

1. Ajouter un nouvel employé à partir du clavier
2. Lire des données à partir d'un fichier
3. Afficher tous les employés

4. Modifier les informations d'un employé
5. Supprimer un employé de la base de données par le numéro
6. Supprimer tous les employés
7. Quitter le programme.

LA STRUCTURE DES DONNÉES