Arbre Binaire de Recherche, Pile, File

Introduction

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.
On appelle arbre binaire de recherche (ABR) une structure de données qui permet de ranger et de retrouver efficacement des valeurs ordonnées. Chaque nœud de l'arbre stocke une information appelée valeur ou clé du nœud et possède zéro, un ou deux nœuds fils (descendants), à droite et à gauche. Ainsi, chaque nœud est le père de ses fils et deux nœuds qui ont le même père sont frères. 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.

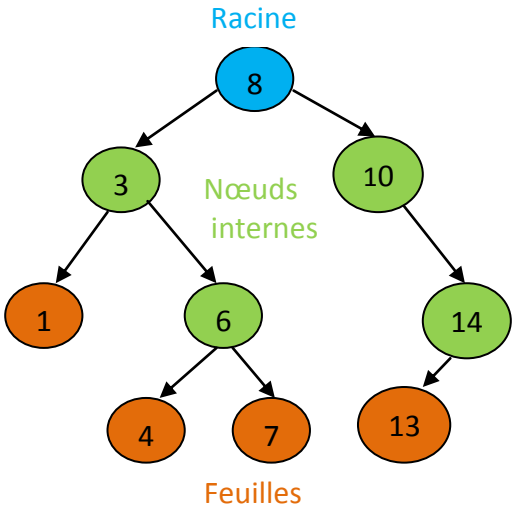

Figure 1. Un Arbre Binaire de Recherche

Description du problème

Contexte

L'entreprise **Sage** propose des solutions de gestion complète pour les petites et moyennes entreprises : gestion de la paie et des ressources humaines. Pour cela, Sage a développé un logiciel optimisé qui gère ces deux domaines. Ce logiciel possède une base de données contenant toutes les informations concernant un employé à savoir :

- Son identifiant
- Son nom et prénom
- Sa fonction au sein de l'entreprise
- Son âge
- Son salaire

Problématique

Malgré l'arrivée tardive de Sage sur ce marché prometteur, elle a déjà un grand nombre de clients grâce notamment à son service optimisé. Sa base de données connaît une grande dynamique : chaque jour plusieurs nouvelles informations sont données par les entreprises sur leurs employés, et plusieurs employés quittent leurs emplois pour différentes raisons (fin de contrat, départ en retraite, etc.) et donc retirés de la base de données. Pour gérer ce dynamisme et garder de bonnes performances lors de la navigation des clients sur son site, l'entreprise décide de revoir le système de gestion de sa base de données.

Spécification et structure de données

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.

La figure 2 présente le schéma de la structure de données.

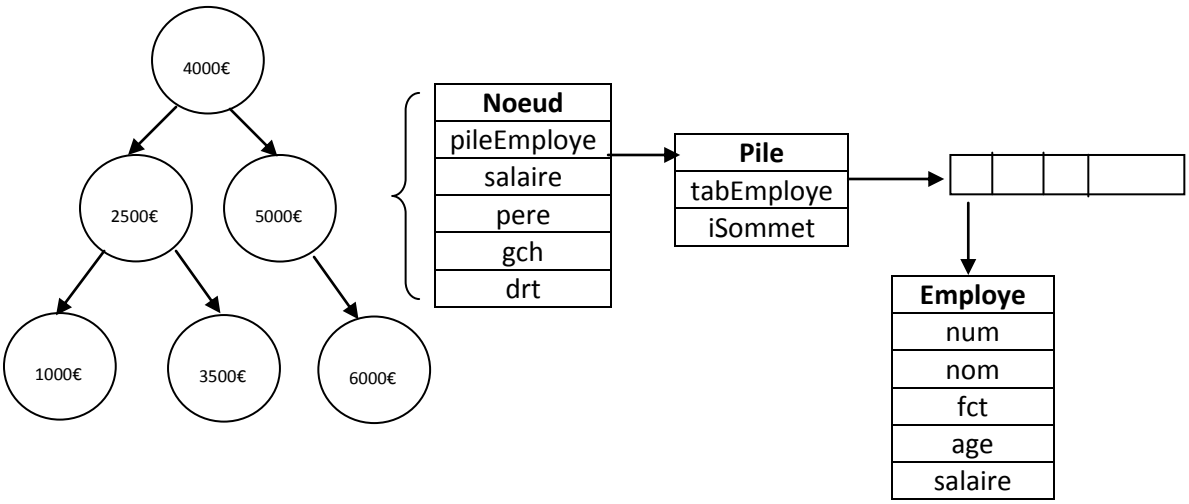

Figure 2. Schéma de la structure des données

```
typedef struct employe{
        int num;                    // 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;
```

Les structures de Piles

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 (comme expliqué dans le paragraphe précédant). La deuxième structure de pile stockant des pointeurs vers les nœuds de l'arbre sera utilisée dans le parcours de l'arbre en profondeur.

Remarque 1 : **les fonctions concernant les opérations basiques sur des piles sont déjà fournies dans les fichiers initiaux ainsi que certaines autres fonctions.**

Les fonctions à développer

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 numero);*
Cette fonction vérifie l'existence d'un employé dont le *numero* 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. Cette fonction vous est fournie dans les fichiers initiaux.**

6. *pNoeud supprimerEmploye(pNoeud arbre, int numero);*
Cette fonction cherche et supprime de l'*arbre* un employé dont le *numero* 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 numero, char * nom, char * fonction, int age, int salaire) ;*
Cette fonction cherche et modifie les informations d'un employé dont le *numero* est passé en paramètre. Le numéro 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

11. *void ecrireFichier(pNoeud arbre, char * f);*
Cette fonction enregistre toutes les données d'un arbre dans un fichier dont le nom est passé en paramètre. Les contenus seront effacés si le fichier existe déjà.

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. Sauvegarder les données dans un fichier
8. Quitter le programme.

Remarque 2 : **Libre à vous de rajouter des fonctions complémentaires que vous jugez nécessaires à l'application.**