ESISAR3

CS353 - Algorithmique

TP 2

2 Les arbres binaires de recherche standard.

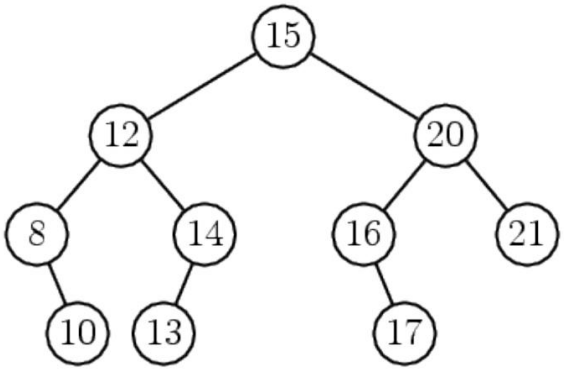
2.1 Introduction

Vous constatez que les performances de votre programme de facturation ne sont pas bonnes. Votre responsable vous demande d'améliorer celui-ci dans les plus brefs délais. Vous décidez alors de modifier votre programme en utilisant les arbres binaires de recherche standard.

A ce point, il est souhaitable de revoir votre cours sur les arbres binaires de recherches.

2.2 Bref rappel sur les arbres binaires.

Un arbre binaire a est un arbre binaire de recherche si, pour tout nœud s de a, les contenus des nœuds du sous-arbre gauche de s sont strictement inférieurs au contenu de s, et que les contenus des nœuds du sous-arbre droit de s sont strictement supérieurs au contenu de s



En conclusion, les arbres binaires de recherche permettent le stockage optimisé d'une liste d'éléments. Les fonctions de recherche, d'insertion et de suppression s'effectuent en O(h), avec h la hauteur de l'arbre.

De plus, vous avez remarqué que dans un arbre binaire de recherche, le parcours infixe fournit les contenus des nœuds en ordre croissant.

2.3 Questions

Vous allez donc résoudre le problème de facturation avec un arbre binaire de recherche.

Pour cela, chaque nœud de votre arbre sera une instance de la structure `Client`. L'identifiant de chaque nœud sera toujours le numéro de téléphone.

Vous allez implémenter progressivement les fonctions d'insertion, de recherche et de suppression dans cet arbre.

Au final, votre programme de facturation aura l'algorithme suivant :

- itération sur les lignes de log
- pour chaque ligne de log
- recherche pour savoir si le client est déjà présent dans l'arbre
- s'il est présent :
- modification des champs de la structure pour incrémenter le prix et le nombre d'appel
- s'il n'est pas présent :
- insertion du client dans l'arbre
- fin itération
- parcours infixe sur l'arbre, ce qui permettra d'obtenir la liste triée par numéro de téléphone croissant, avec le coût pour chaque abonné

Question 1

Reprenez votre code de l'exercice 0. Modifiez votre structure pour avoir une référence vers le fils droit, le fils gauche.

Ecrire la fonction qui permette de créer un client à partir d'un numéro de téléphone, un nombre d'appel, un coût total en centimes d'euros

```
struct Client * createNode(int numeroTel, int nbAppel,int cout) ;
```

Question 2

Ecrire une fonction qui crée un arbre avec les valeurs de l'exemple du paragraphe 2.2

```
struct Client * createSampleTree();
```

et qui retourne un pointeur sur la racine.

Question 3

Ecrire une fonction qui affiche le résultat du parcours infixe

```
void parcoursInfixe(struct Client * abr);
```

A l'aide de votre fonction de la question 2, créez un arbre et vérifiez que le parcours infixe

fournit bien les contenus des nœuds en ordre croissant.

Question 4

Ecrire la fonction de recherche d'un nœud dans l'arbre :

```
struct Client * search(struct Client * abr,int numeroTel);
```

Question 5

Ecrire la fonction d'insertion d'un nœud dans l'arbre :

```
struct Client * insert(struct Client * abr, int numeroTel, int prixAppel)
```

Cette fonction retourne la racine de l'arbre.

Question 6 :

Ecrire la fonction de suppression d'un nœud dans arbre.

```
struct Client * deleteNode(struct Client * abr, int numeroTel);
```

Cette fonction retourne la racine de l'arbre.

Question 7 :

Implémentez le programme de facturation comme expliqué au début du paragraphe 2.3, en vous aidant des fonctions d'insertion et de recherche que vous venez de développer.

Comparer les performances entre le TP1 et le TP2 sur le cas avec 20 000 clients.