ESISAR3

CS315 - Algorithmique

TPs 3 et 4

3 Les arbres rouges et noir

3.1 Introduction

Vous constatez que les performances de votre programme de facturation ont été améliorées. Néanmoins, étant donné le développement de votre société, vous craignez que l'augmentation du nombre de clients conduisent de nouveau à un temps de réaction trop long.

Vous décidez alors de modifier votre programme pour utiliser les arbres binaires de recherche **rouges et noirs.**

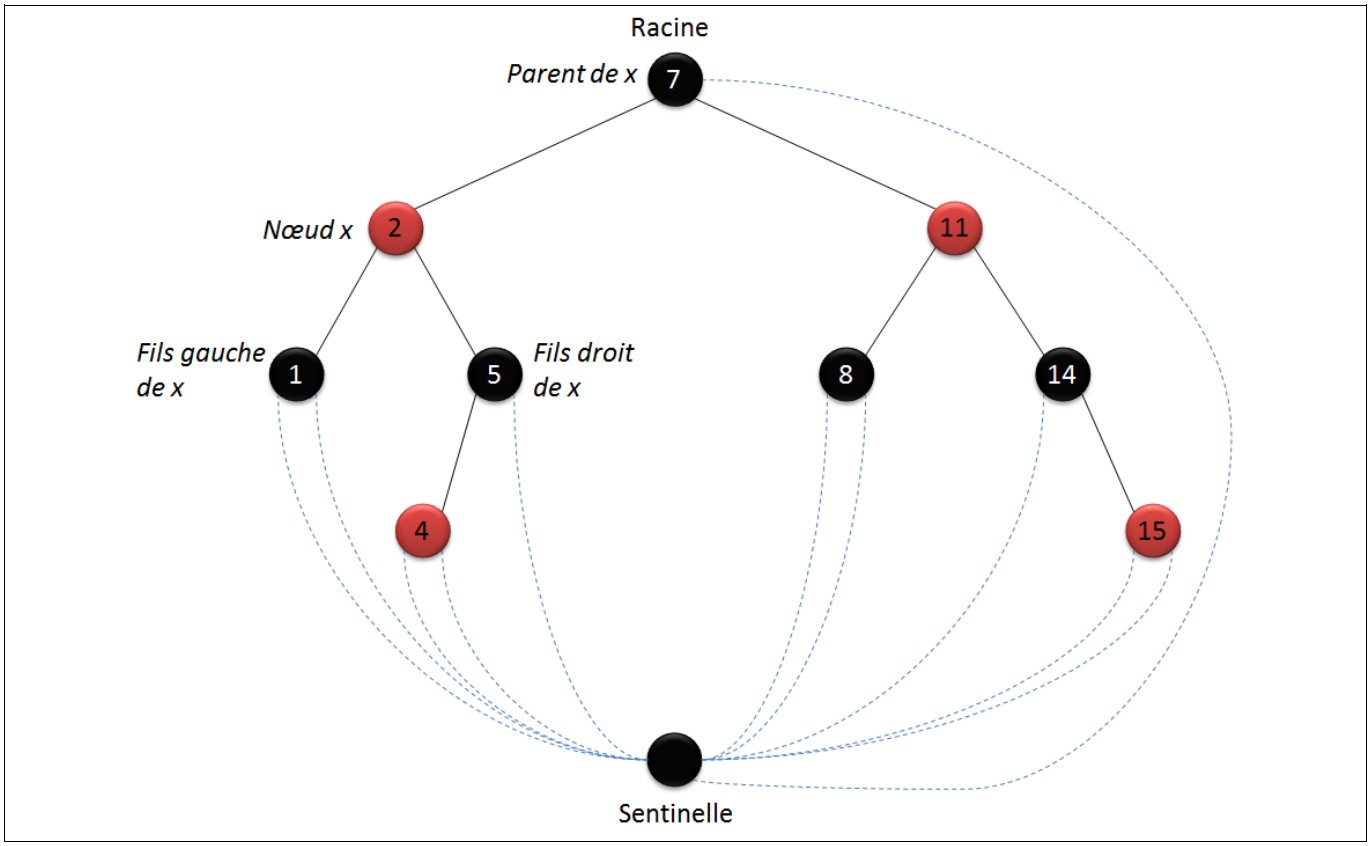
A ce point, il est souhaitable de revoir votre cours sur les arbres rouges et noirs

3.2 Bref rappel sur les arbres rouges et noirs.

En plus de votre cours, une explication sur les arbres rouge et noir est disponible sur

http://www.liafa.jussieu.fr/~carton/Enseignement/Algorithmique/Programmation/RedBlackTree/

Les caractéristiques d'un arbre rouge-noir sont les suivantes :
1. chaque nœud est soit rouge, soit noir
2. la racine est noire
3. chaque feuille est noire
4. si un nœud est rouge, alors ses deux enfants sont noirs
5. pour chaque nœud ; tous les chemins reliant le nœud à des feuilles contiennent le même nombre de nœuds noirs.



Pour ce TP, nous poserons les contraintes d'implémentation suivantes :

- une sentinelle sera déclarée, elle sera le fils de toutes les feuilles et aussi le père de la racine (la racine sera le fils gauche de la sentinelle). Pour connaître la racine de l'arbre, nous utiliserons cette sentinelle (l'arbre sera identifié par la connaissance de cette sentinelle)/
- pour représenter un nœud, nous utiliserons une structure `Client`, qui contient le numéro de téléphone, le nombre d'appel , le prix total, une couleur (ROUGE ou NOIR), et un pointeur vers son père, deux pointeurs vers les fils.

3.3 Questions

Question 1

Modifiez votre structure `Client` pour avoir une référence vers le père et une couleur.

Utilisez le code suivant pour définir les couleurs :

```
typedef enum {RED=0, BLACK=1 } Color;
```

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 et d'une couleur :

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

A noter : le nœud retourné ne fait pas encore parti de l'arbre

Question 2

Ecrire la fonction qui crée un arbre avec les valeurs de l'exemple du paragraphe 3.2

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

et qui retourne la sentinelle de l'arbre

Question 3

Ecrire la fonction d'affichage de l'arbre suivant un parcours préfixé. Cette fonction devra afficher la sentinelle (avec la lettre S), et prend en entrée la sentinelle.

```
void parcoursPrefixe(struct Client * sentinelle);
```

Dans l'exemple du paragraphe 3.2, nous devrons ainsi obtenir :

7(N) 2(R) 1(N) S S 5(N) 4(R) S S S 11(R) 8(N) S S 14(N) S 15(R) S S

L'utilisation d'une sous fonction est bien sûr autorisée.

Entrainez-vous à convertir ce résultat en un dessin de l'arbre (feuille + crayon!).

Ecrire la fonction d'affichage de l'arbre suivant un parcours infixé.

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

Question 4

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

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

Question 5

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

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

Pour cela, vous devrez d'abord implémenter les fonctions :

```
void left_rotate( struct Client *x);
void right_rotate( struct Client *y);
```

Question 6 :

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

```
void deleteNode(struct Client * sentinelle, int numeroTel);
```

Question 7 :

Implémentez le programme de facturation en utilisant cette fois ci les arbres rouges et noirs.

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