TP n°11 - Arbres rouge noir

Notions abordées

- Structure pour les nœuds colorés
- Opération de rotation, de réparation, de scission d'un 4-nœud
- Insertion d'une clé en feuille
- Révision de `struct` et des pointeurs

Toutes les fonctions doivent être commentées et **testées**. Elles doivent notamment être munies d'une description faisant suite à d'éventuelles hypothèses sur leurs arguments.

Ce TP a deux objectifs. Le premier est de voir comment on implémente des arbres en C. Le second est l'implémentation de la structure de donnée `ensemble` par des arbres rouge-noirs. Il s'agit donc de gérer un ensemble de données, chacune étant munie d'une clé (supposée de type `int` ici), de manière à pouvoir efficacement retrouver cette donnée à partir de sa clé, et à pouvoir aussi ajouter de nouvelles données. En raison de la technicité de l'opération de suppression dans les arbres rouge-noirs, on omettra la suppression pour ce TP.

Exercice 1 Structure pour les arbres rouges-noirs

Question 1 ✍

Proposer une structure qui permettrait d'implémenter les arbres rouges-noirs, y compris des arbres vides. Schématiser la représentation en mémoire d'une telle structure. Préciser l'utilité des champs de la structure.

Question 2 💻

Sur cahier de prépa récupérer les fichiers `squelette_ARN.c`, et `test_ARN.c`.
Pendre connaissance du type proposé pour les ARN dans `squelette_ARN.c`.
Prendre aussi connaissance du type `orient`. C'est un type admettant deux valeurs : `gauche` et `droite`. Il permettra de retenir si un nœud est fils gauche ou droit de son père. La construction d'un tel type, aux valeurs finies, se fait en C avec `enum`, c'est l'équivalent d'un type somme en OCaml qui n'admettrait que des constructeurs sans arguments.

Question 3 💻

Compléter le corps de la fonction `cree_feuille` qui crée un ARN réduit à une feuille dont l'unique nœud enregistre la donnée et la clé passées en argument.

Question 4 💻

Compléter le corps de la fonction `est_vide` qui teste si un ARN est vide.

Question 5 💻

Compléter le corps de la fonction `est_feuille` qui teste si un ARN est réduit à une feuille.

Exercice 2 Décomposer l'insertion

Question 1 ✍
Dessiner les arbres rouge-noirs de chaque étape lors de la suite d'insertions ci-dessous. Pour certaines opérations il y a des étapes intermédiaires, les dessiner aussi. Pour chaque étape, préciser quelle opération a été réalisée.

```
arn a12 = cree_feuille (12,"douze");
insere (&a12,13,"treize");
insere (&a12,14,"quatorze");
insere (&a12,15,"quinze");
insere (&a12,16,"seize");
insere (&a12,10,"dix");
insere (&a12,11,"onze");
```

Question 2 ✍
Les opérations d'insertions ci-dessous sont extraites d'un jeu de test du fichier `test_arn.c`.
Vérifier que les arbres attendus et les tests réalisés dans ce jeu de tests sont cohérents.

Question 3 ✍ Point clé pour la suite
Expliquer pourquoi l'arbre est passé par pointeur à la fonction `insere`. *On pourra s'appuyer sur le schéma proposé à la question 1 de l'exercice 1.*

Exercice 3 Rotation

On a écrit en cours le pseudo-code de la rotation *gauche* d'une arête u, v c'est-à-dire qui ne vaut que si v est fils *gauche* de u, et ce sans fixer si u était fils gauche ou droit de son père grâce à la donnée en entrée de l'orientation de l'arête reliant u à son père.

Question 1 ✍
Donner de même le pseudo-code de la rotation *droite*.

Question 2 ✍
Donner le code d'une rotation qui ne fixe pas à l'avance si v est fils gauche ou droit de u, mais qui au contraire prend en entrée l'orientation de cette arête.

Question 3 ✍
Traduire en français les hypothèses de la fonction `rotation_ptr`. D'après sa description, quel appel à `rotation_ptr` permet de faire la rotation de l'arête u, v si v a pour père u et u pour père p ?

Question 4 ✍
Prendre connaissance de la fonction `rotation_pere`. Comparer les appels à `rotation_ptr` qu'effectue cette fonction avec les appels proposés à la question précédente.

Question 5 💻
Remplir le corps de la fonction `void rotation_ptr (arn u, arn v, orient ov, arn* ptr_u)` qui réalise la rotation de l'arête `u,v`. Tester cette fonction en dé-commentant la partie idoine du fichier `test_ARN.c`.

Exercice 4 Scission

Question 1 ✍
Lorsqu'on réalise la scission d'un 4-nœud, à moins que celui-ci ne soit racine, on est amené à "fusionner" son nœud milieu avec son père. Pour cela ce nœud milieu est colorié en rouge, mais il faut ensuite vérifier que son père n'est pas déjà rouge, sans quoi il faut réaliser une ou deux rotations pour réparer. On remarque que les mêmes rotations sont nécessaires lorsque, lors de l'insertion d'un nouveau nœud, on ajoute le nœud par "fusion" avec une feuille déjà existante.
Afin de factoriser la portion de code des fonctions `scinde` et `insere` correspondant à cette réparation, on ajoute une fonction `repare_post_fusion` qui réalise les rotations nécessaires pour deux nœuds rouges consécutifs (le reste de l'ARN étant supposé valide).
Décrire en français les hypothèses de cette fonction.

Question 2 💻
Remplir le corps de la fonction `repare_post_fusion`. Tester cette fonction en dé-commentant la partie idoine du fichier `test_ARN.c`.

Question 3 ✍
Traduire en français les hypothèses de la fonction `scinde_ptr`.

Question 4 💻
Remplir le corps de la fonction `scinde_ptr`. Tester cette fonction en dé-commentant la partie idoine du fichier `test_ARN.c`.

Question 5 💻
Compléter le corps de la fonction `est_4_nd` qui teste si un nœud d'un ARN représente un 4-nœud.
Dans l'optique d'un cheminement descendant de la racine vers les feuilles, on teste en réalité si le nœud de l'ARN pris en argument le nœud le plus haut de 3 nœuds rouges représentant un 4-nœud.

Exercice 5 Insertion

Question 1 💻
Définir une fonction de signature `void insere (arn* ptr_a, int c, void* d)` qui réalise l'insertion de la donnée `d` avec la clé `c` dans l'ARN sur lequel pointe `ptr_a`. Cet arbre doit rester un ARN (contrainte d'équilibre de de recherche). Tester cette fonction par plusieurs exemples.