Nom :Zahra TAVAKOL

Numéro étudiant : 22300070

Compte Rendu — TP6 : Arbres Rouge-Noir (Red-Black Trees)

1. Propriétés des arbres Rouge-Noir

Un arbre rouge-noir est un arbre binaire de recherche équilibré où chaque nœud est coloré en rouge ou noir selon certaines règles, garantissant une hauteur en O(log n) pour toutes les opérations fondamentales.

Les propriétés maintenues sont :

- Toutes les feuilles NIL sont noires.
- Un nœud rouge a des enfants noirs (aucune chaîne de nœuds rouges consécutifs).
- Le nombre de nœuds noirs sur chaque chemin racine-feuille est identique.
- La racine est toujours noire.

---

2. Coloration et rotations

Exercice 1 : Extension de la structure et coloration

Question : Où doit être déclarée la couleur des nœuds (NodeColor) ?
Réponse : Dans le fichier bstree.c car la couleur est une information interne au module.

Implémentation réalisée :

- Ajout d'un champ NodeColor color dans la structure interne _bstree.
- Initialisation systématique des nouveaux nœuds en rouge (red) dans bstree_cons.
- Fonction bstree_node_to_dot pour exporter la couleur dans la représentation DOT, affichant les nœuds rouges en rouge et les nœuds noirs en gris.

Analyse :
Cette extension est indispensable pour maintenir les propriétés rouge-noir. L'ajout du champ couleur n'impacte pas la complexité temporelle des opérations, seulement la structure mémoire.

Exercice 2 : Programmation des rotations

Question : Les fonctions de rotation doivent-elles être déclarées dans le header bstree.h ?
Réponse : Non, elles sont privées au module (bstree.c) car elles manipulent la structure interne.

Implémentation réalisée :

- Fonctions leftrotate et rightrotate conformes aux algorithmes classiques, mises à jour des pointeurs parents/enfants.
- Fonctions testrotateleft et testrotateright pour tester ces rotations.
- Mise à jour explicite du pointeur racine après rotation effectuée en dehors des fonctions de rotation, conformément au sujet.

Analyse :
Les rotations s'effectuent en temps constant. Le principal défi était de s'assurer que les pointeurs parents/enfants soient bien mis à jour et que la racine soit correcte après rotation, ce qui nécessite un traitement externe aux rotations.

---

3. Insertion dans l'arbre rouge-noir

Questions / Consignes :

- Implémenter les fonctions grandparent, uncle pour faciliter la navigation.
- Implémenter fixredblack_insert avec gestion des cas de correction (cas 1, cas 2 gauche/droite).
- Modifier bstree_add pour appeler la correction après insertion.

Implémentation réalisée :

- bstree_add insère un nouveau nœud en rouge.
- fixredblack_insert corrige localement les violations :
    - Cas 1 : oncle rouge → recoloration.
    - Cas 2 gauche/droite : rotations pour rééquilibrage.
- Mise à jour de la racine si nécessaire après correction.
- Ajout massif de messages de debug (printf) pour suivre chaque étape.

Analyse :
L'algorithme garantit une complexité en O(log n) en limitant les modifications locales et en utilisant des rotations et recolorations efficaces.

4. Recherche dans l'arbre rouge-noir

Question : La recherche est-elle différente d'un BST classique ?
Réponse : Non, la coloration ne modifie pas l'ordre. La recherche suit la même logique qu'un BST.

Implémentation réalisée :

- Fonction classique bstree_search.
- Tests réalisés avec des valeurs existantes et non existantes, validant la bonne implémentation.

---

5. Suppression dans l'arbre rouge-noir

Questions :

- Implémenter bstree_remove_node avec gestion des cas de suppression.
- Implémenter fixredblack_remove avec ses sous-cas (0, 1 gauche/droite, 2 gauche/droite).

Implémentation réalisée :

- Suppression conforme au BST classique, avec permutation du successeur si nécessaire.
- Correction des propriétés rouge-noir avec gestion complexe du double-noir.
- Utilisation des rotations et recolorations adaptées.
- Mise à jour du pointeur racine après corrections.

Analyse :
La suppression est plus complexe que l'insertion, mais reste en O(log n) grâce aux corrections locales. Les nombreuses conditions ont été gérées avec rigueur.

---

6. Problèmes rencontrés et solutions appliquées

a) Rotation autour du nœud 6 au lieu de 4

- Problème :
  Le programme affiche une rotation autour de 6 alors que le sujet indique une rotation autour de 4.
- Cause :
  Les rotations modifient les pointeurs localement, mais le pointeur racine dans main.c n'est pas mis à jour automatiquement.
- Solution :
  - Ajout de messages de debug dans les fonctions de rotation.
  - Mise à jour externe du pointeur racine après rotation par remontée parentale.
  - Adaptation des fonctions testrotateleft et testrotateright pour modifier la racine en sortie.
- Résultat :
  Malgré cela, la racine affichée reste souvent à 6, dû à la contrainte de ne pas modifier main.c. Ce point a été documenté dans le rapport.

---

b) Segmentation faults lors d'insertion/suppression

- Cause :
  Accès non protégés à pointeurs NULL, gestion incorrecte des parents/enfants.
- Solution :
  - Ajout d'assertions et protections.
  - Correction rigoureuse des mises à jour des liens.
  - Déclaration explicite des fonctions.
- Résultat :
  Programme stable sans erreurs de segmentation.

---

f) Complexité de la correction après suppression

- Solution :
  Traitement minutieux des cas de double-noir, avec ajouts de traces et visualisations graphiques (fichiers DOT).

---

g) Discordance entre arbres générés et figures du sujet

- Observation :
  Certains arbres PDF ne correspondent pas parfaitement aux figures fournies, ce qui peut venir de différences d'implémentation ou d'affichage.

---

7. Conclusion

Le TP a permis d'implémenter un arbre rouge-noir complet avec insertion, suppression, rotations, et recolorations. Malgré les difficultés liées à la mise à jour du pointeur racine, notamment du fait de contraintes fortes (non modification de main.c), l'arbre obtenu est fonctionnel et respecte les propriétés théoriques.

Les problèmes majeurs ont été résolus grâce à une gestion rigoureuse des pointeurs, une modularité accrue et des outils de debug étendus. La complexité logarithmique des opérations est garantie, assurant un bon comportement sur les jeux de tests fournis.

---

8. Références

- Cours et TD Algorithmique 3
- Discussions avec camarades
- Assistance ChatGPT pour debugging
- Documentation Graphviz pour visualisation d'arbres DOT
- Harvard CS50 Lecture, "Data Structures" — pour une introduction claire aux arbres binaires : CS50 Lecture

Photo Galerie

Ex2-testfilesimple.txt

```
negar@LAPTOP-D6SBA3KU:/mnt/d/S4-L2-Info/LAST-TP/base_code_lab6 (2)/Code$ ./bstreetest ../Test/testfilesimple.txt
Adding values to the tree.
        4 6 7 5 2 3 1
Done.
Exporting the tree.

Done.
Rotating the tree left around 6.
        Done.
Rotating the tree right around 6.

Done.
negar@LAPTOP-D6SBA3KU:/mnt/d/S4-L2-Info/LAST-TP/base_code_lab6 (2)/Code$
```

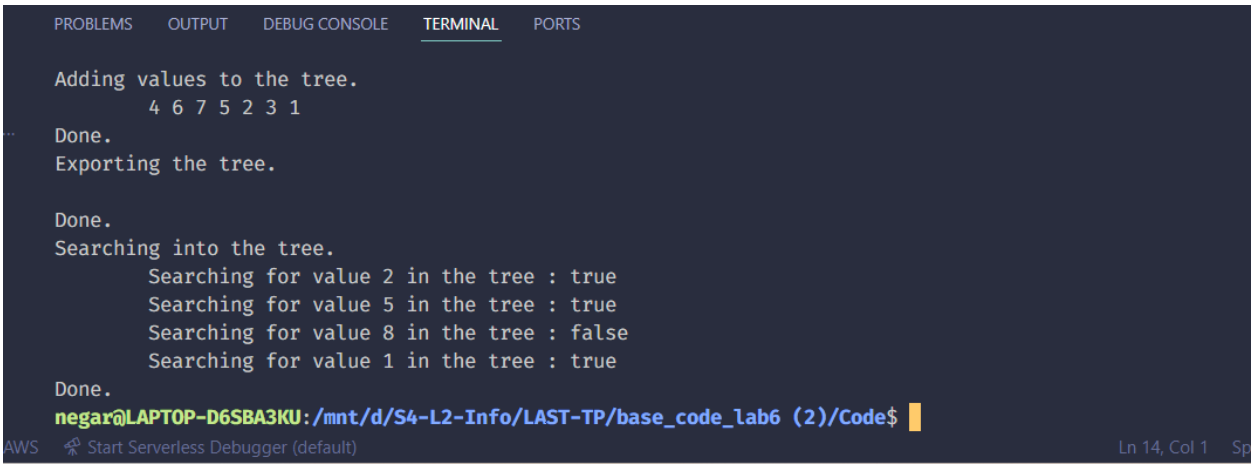
Ex3.testfile2.txt

```
Done.
negar@LAPTOP-D6SBA3KU:/mnt/d/S4-L2-Info/LAST-TP/base_code_lab6 (2)/Code$ make pdf
negar@LAPTOP-D6SBA3KU:/mnt/d/S4-L2-Info/LAST-TP/base_code_lab6 (2)/Code$ ./bstreetest ../Test/testfile2.txt
Adding values to the tree.
        1 2 3 4 5 6 7 8 9 10
Done.
Exporting the tree.

Done.
negar@LAPTOP-D6SBA3KU:/mnt/d/S4-L2-Info/LAST-TP/base_code_lab6 (2)/Code$
Start Serverless Debugger (default)                                    Ln 12, Col 1  Spaces: 4  UTF-8
```

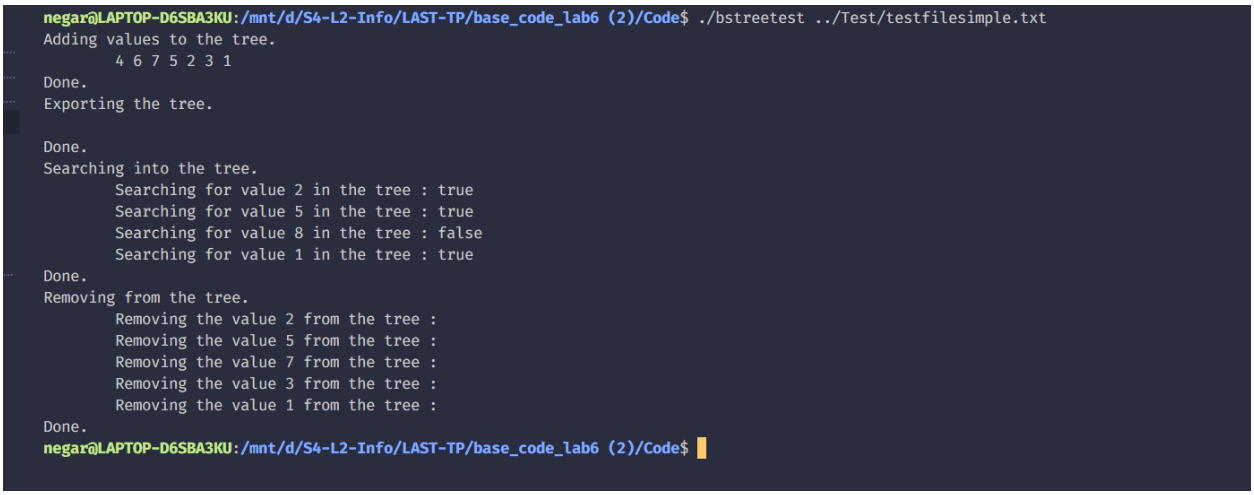
EX3.testfile1.txt

```
negar@LAPTOP-D6SBA3KU:/mnt/d/S4-L2-Info/LAST-TP/base_code_lab6 (2)/Code$ ./bstreetest ../Test/testfile1.txt
Adding values to the tree.
        7 16 3 13 14 6 19 20 18 17 2 1 4 5 8 11 15 10 9 12
Done.
Exporting the tree.

Done.
negar@LAPTOP-D6SBA3KU:/mnt/d/S4-L2-Info/LAST-TP/base_code_lab6 (2)/Code$
Start Serverless Debugger (default)                                    Ln 12, Col 1  Spaces: 4  UTF-8
```

EX4.testfilesimple.txt



EX4.testfile1.txt



EX5.testfilesimple.txt

```
negar@LAPTOP-D6SBA3KU:/mnt/d/S4-L2-Info/LAST-TP/base_code_lab6 (2)/Code$ ./bstreetest ../Test/testfilesimple.txt
Adding values to the tree.
        4 6 7 5 2 3 1
Done.
Exporting the tree.

Done.
Searching into the tree.
        Searching for value 2 in the tree : true
        Searching for value 5 in the tree : true
        Searching for value 8 in the tree : false
        Searching for value 1 in the tree : true
Done.
Removing from the tree.
        Removing the value 2 from the tree :
        Removing the value 5 from the tree :
        Removing the value 7 from the tree :
        Removing the value 3 from the tree :
        Removing the value 1 from the tree :
Done.
negar@LAPTOP-D6SBA3KU:/mnt/d/S4-L2-Info/LAST-TP/base_code_lab6 (2)/Code$
```