INF2010 - Structures de données et algorithmes
Hiver 2017
Travail Pratique 4
Arbres binaires

Objectifs:

- ✓ Implémenter un arbre binaire de recherche.
- ✓ Implémenter certaines fonctionnalités d'un arbre rouge-noir.
- ✓ Parcourir un arbre rouge-noir en pré-ordre, post-ordre et par niveaux.

I. Arbre binaire de recherche

Un arbre binaire de recherche est un arbre qui respecte les deux propriétés suivantes:

1. La valeur du fils gauche d'un nœud donné est inférieure à la valeur du nœud courant.
2. La valeur du fils droit d'un nœud donné est supérieure à la valeur du nœud courant.

Exercice 1: Insertion dans un arbre binaire de recherche (2 points)

L'ajout d'un élément dans un arbre binaire se fait dans une feuille, tout en respectant les deux propriétés indiquées ci-dessus. Dans le fichier « Tree.java », complétez les méthodes :

- **insert :** permet d'ajouter un élément dans l'arbre, la racine de cet arbre est l'attribut « root »
- **printPrefixe :** permet d'afficher l'arbre dans l'ordre préfixe;
- **printInfixe :** permet d'afficher l'arbre dans l'ordre infixe;
- **printPostfixe :** permet d'afficher l'arbre dans l'ordre ordre postfixe;
- **getHauteur :** calcule la hauteur d'un arbre qui représente le nombre de nœud entre la racine et la plus distante feuille de l'arbre.

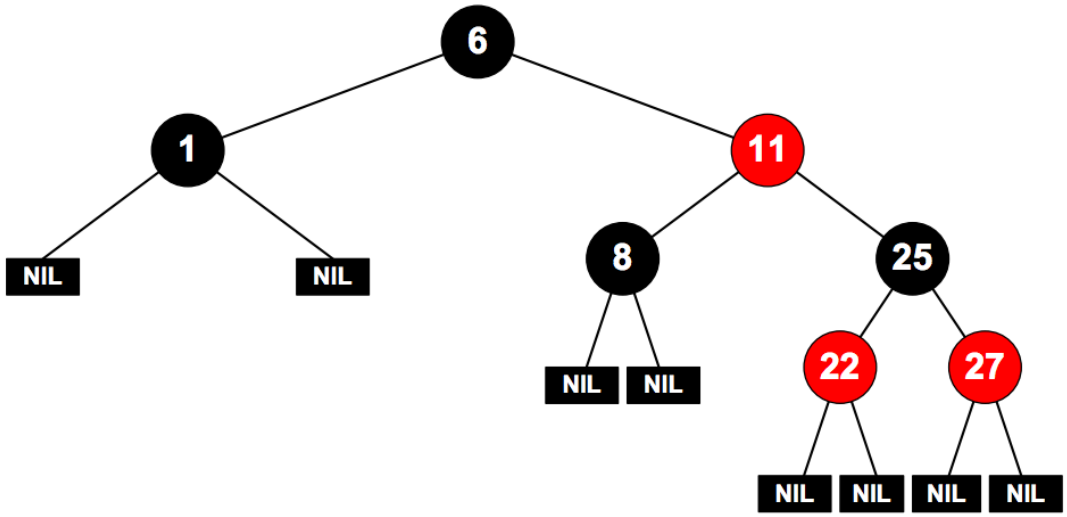
II. Arbre rouge-noir

Un arbre rouge-noir est un arbre binaire de recherche équilibré. En plus des propriétés usuelles d'un arbre binaire de recherche, l'arbre rouge-noir doit respecter les contraintes suivantes :

3. Un nœud est soit rouge ou noir ;
4. La racine est noire ;
5. Toutes les feuilles sont noires (les feuilles sont des nœuds vides représentés ici par NIL) ;
6. Tous les enfants d'un nœud rouge sont noirs ;
7. Tout chemin d'un nœud vers une de ses feuilles contient le même nombre de nœuds noirs.

Ces propriétés font en sorte que l'insertion et la suppression dans un arbre rouge-noir est de complexité O(log n) dans le pire cas, car les contraintes énumérées précédemment font en sorte

que tout chemin reliant un nœud et à une feuille est au plus deux fois plus long que le chemin menant à la feuille la plus proche.



Exercice 1: Implémentation de méthodes utilitaires (0.75 point)

Pour implémenter l'insertion et la suppression, nous aurons besoin de méthodes permettant la manipulation et la recherche de nœuds de l'arbre. Complétez dans RedBlackTree.java les méthodes suivantes, définies dans la classe interne RBNode:

- **grandParent():** retourne le nœud grand-père du nœud courant, null si inexistant;
- **uncle():** retourne le nœud oncle du nœud courant, null si inexistant ;
- **sibling():** retourne le nœud frère du nœud courant, null si inexistant.

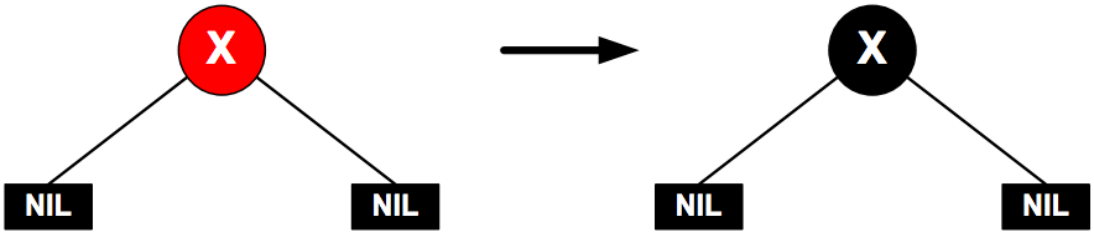
Exercice 2: Insertion et Recherche dans un arbre rouge-noir (5 points)

Dans un arbre binaire classique, l'information est insérée dans l'arbre en tant que feuille d'un nœud. Dans un arbre rouge-noir les feuilles ne contiennent aucune information. L'insertion se fait donc toujours en créant un nœud rouge contenant la valeur à insérer avec deux feuilles NIL.

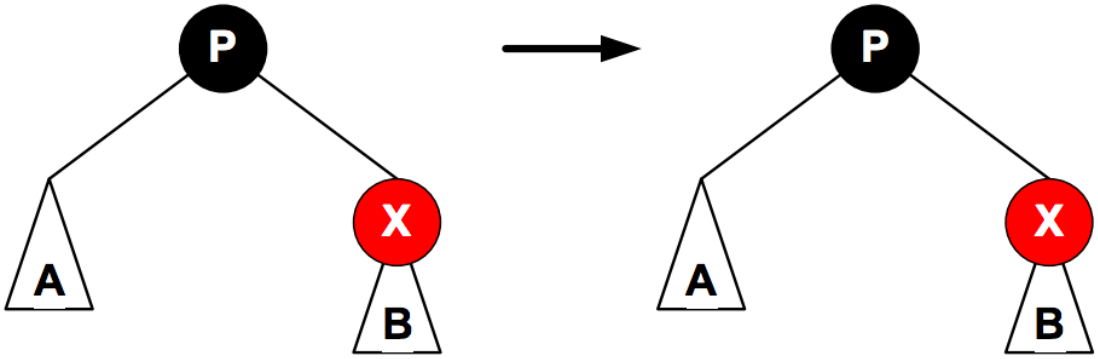
La première étape de l'insertion est d'abord de localiser où le nouveau nœud rouge sera inséré. Ceci est identique à la localisation d'insertion dans un arbre binaire de recherche classique. Par la suite, il faut s'assurer que les propriétés de l'arbre rouge-noir demeurent valides.

Lors de l'insertion, il faut vérifier dans l'ordre si le nœud inséré (N) tombe dans un des 5 cas suivants :

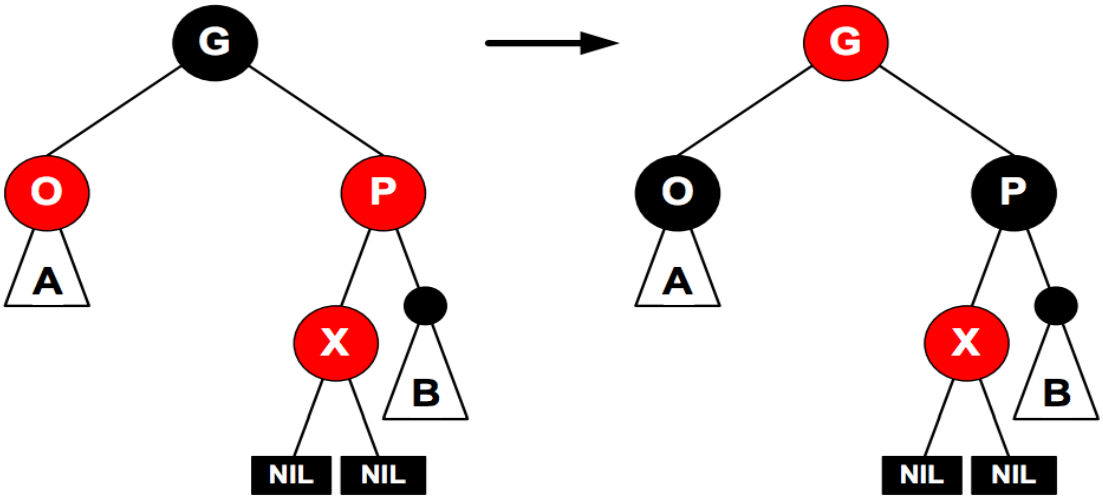
1. Le nœud N est la racine de l'arbre → le nœud devient noir ;



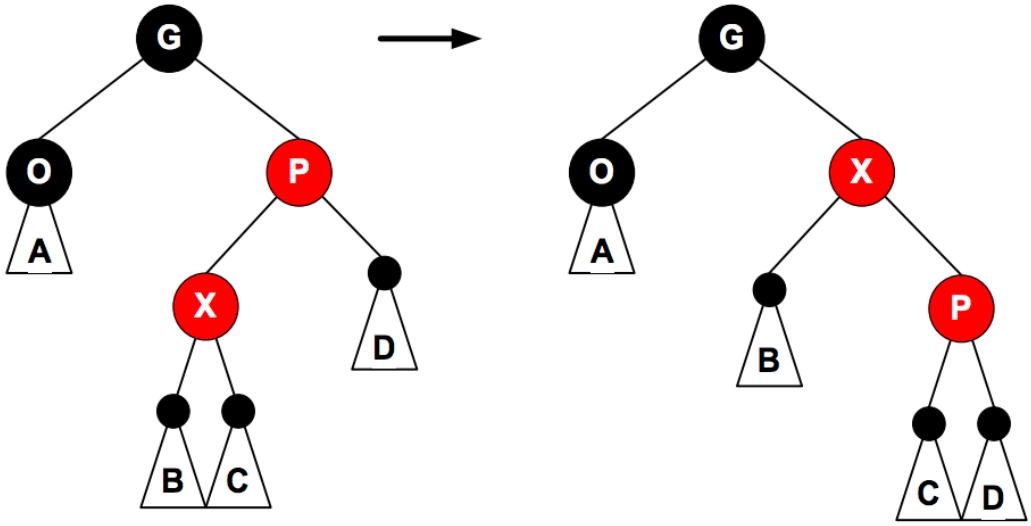
2. Le parent P du nœud est noir → tout est correct, on sort ;



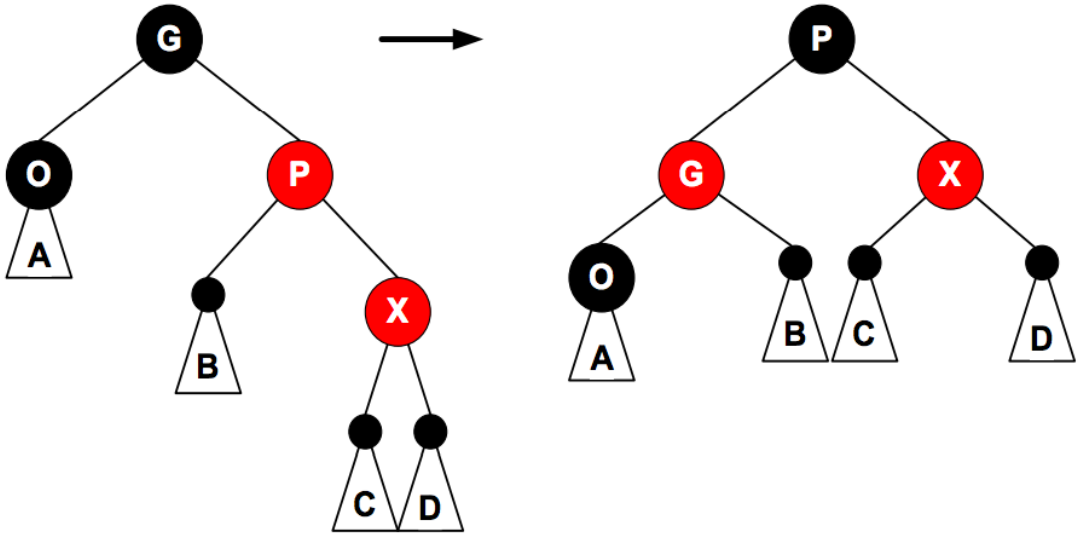
3. Le parent P et l'oncle O du nouveau nœud sont rouges → le parent et l'oncle deviennent noirs, le grand parent G devient rouge; il faut alors vérifier que G respecte les 5 règles dans l'ordre à son tour;



4. Le parent P est rouge, l'oncle est noir, le nœud X est l'enfant de gauche de P et P est l'enfant de droite de G (ou symétriquement, droite-gauche) → on procède à une rotation à droite (resp. à gauche) autour de P et on vérifie le cas 5 sur l'enfant de droite de X (resp. enfant de gauche de X). Sinon (s'il n'y a pas eu de rotation), on vérifie le cas 5 sur X;



5. Le parent P est rouge, l'oncle O est noir, X est l'enfant de droite de P et P est l'enfant de droite de G (ou cas symétrique, gauche-gauche) → on change la couleur de G et de P, on procède à une rotation à gauche (resp. à droite) autour de G et on sort.



Exercice 3: Affichage des valeurs des nœuds (0.25 point)

Une fois que l'insertion fonctionne, implémenter la méthode récursive de recherche **find();**

Exercice 4: Affichage des valeurs des nœuds (1 point)

Dans le fichier RedBlackTree.java, complétez les méthodes récursives :

- **printTreePreOrder() :** affiche le contenu de l'arbre en pré-ordre ;
- **printTreePostOrder() :** affiche le contenu de l'arbre en post-ordre ;
- **printTreeAscendingOrder() :** affiche le contenu de l'arbre en ordre croissant;
- **printTreeDescendingOrder() :** affiche le contenu de l'arbre en ordre décroissant.

Remarque : Référez-vous à l'affichage attendu pour implémenter vos fonctions. Faites bien attention aux virgules, aux espaces, aux parenthèses et aux accolades dans l'affichage, ils sont tous demandés et vous devez respecter l'affichage attendu.

Exercice 5: Affichage par niveau (1 point)

Dans le fichier RedBlackTree.java, complétez la méthode **printTreeLevelOrder()** qui permet d'afficher le contenu de l'arbre par niveaux. Référez-vous à l'affichage attendu pour implémenter vos fonctions. Faites bien attention aux virgules, aux espaces, aux parenthèses et aux accolades dans l'affichage, ils sont tous demandés et vous devez respecter l'affichage attendu.

Instructions pour la remise

Le travail doit être fait par équipe de 2 personnes idéalement et doit être remis via Moodle :

Groupe Mercredi (B2) : mardi 21 mars 2017, 23:55
Groupe Mercredi (B1) : mardi 28 mars 2017, 23:55

Veuillez envoyer vos fichiers .java seulement dans une archive de type ***.zip** qui portera le nom **inf2010_lab4_MatriculeX_MatriculeY** (de sorte que **MatriculeX < MatriculeY**). Les travaux en retard seront **pénalisés de 20 %** par jour de retard. Aucun travail **ne sera accepté après 4 jours** de retard.