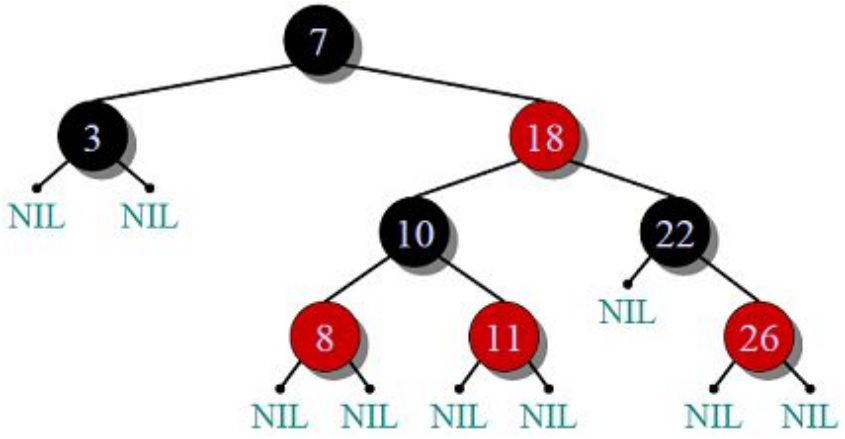
Arbre rouge-noir

Dernière mise à jour: 27-11-2019[1]

Introduction

L'arbre **Rouge**-**Noir** est un arbre de recherche binaire (ABR) auto-équilibré où chaque nœud suit les règles suivantes.



1) Chaque nœud a une couleur **rouge** ou **noire**.

2) La racine de l'arbre est toujours **noire**.

3) Il n'y a pas deux nœuds **rouges** adjacents (un nœud **rouge** ne peut pas avoir un parent **rouge** ou un enfant **rouge**).

4) Chaque chemin d'un nœud (y compris la racine) à l'un de ses nœuds NULL descendants a le même nombre de nœuds noirs.

Pourquoi les arbres rouge-noir?

La plupart des opérations ABR (par exemple, recherche, max, min, insertion, suppression, etc.) prennent un temps $O(h)$ où h est la hauteur de l'ABR. Le coût de ces opérations peut devenir $O(n)$ pour un arbre binaire asymétrique. Si nous nous assurons que la hauteur de l'arbre reste $O(log\ n)$ après chaque insertion et suppression, alors nous pouvons garantir une limite supérieure de $O(log\ n)$ pour toutes ces opérations. La hauteur d'un arbre **Rouge**-**Noir** est toujours $O(log\ n)$ où n est le nombre de nœuds dans l'arbre.

[1] https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/

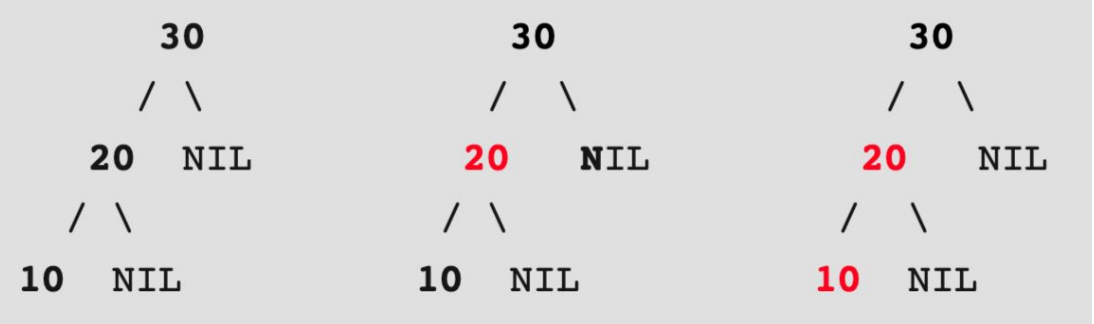
Comparaison avec l'arbre AVL

Les arbres AVL sont plus équilibrés que les arbres **Rouge**-**Noir**, mais ils peuvent entraîner plus de rotations lors de l'insertion et de la suppression. Donc, si votre application implique de nombreuses insertions et suppressions fréquentes, les arbres **Rouge**-**Noir** devraient être préférés. Et si les insertions et suppressions sont moins fréquentes et que la recherche est une opération plus fréquente, alors les **arbres AVL** doivent être préférés aux arbres **Rouge**-**Noir**.

Comment un arbre rouge-noir assure-t-il l'équilibre?

Un exemple simple pour comprendre l'équilibrage est qu'une chaîne de 3 nœuds n'est pas possible dans l'arbre **Rouge**-**Noir**. Nous pouvons essayer n'importe quelle combinaison de couleurs et les voir toutes enfreindre la propriété de l'arbre **Rouge**-**Noir**.

Une chaîne de 3 nœuds n'est pas possible dans les arbres **Rouge**-**Noir**. Les arbres suivants ne sont PAS **Rouge**-**Noir**

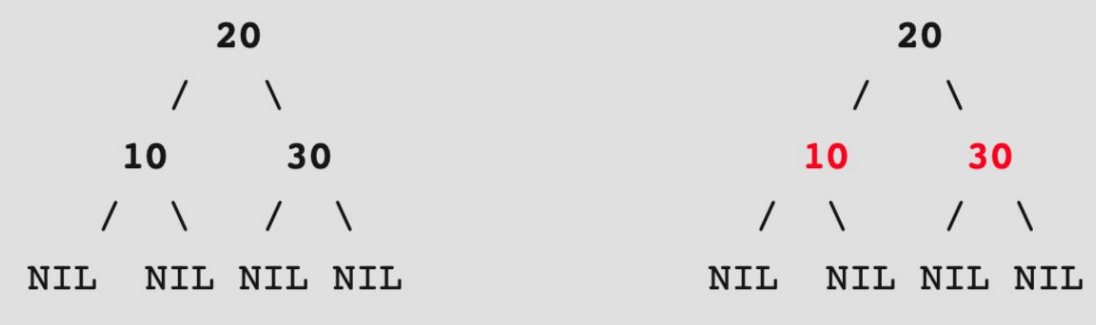


Viole Propriété **4**)

Viole Propriété **4)**

Viole Propriété **3)**

Voici différents arbres rouges-noirs possibles avec plus de 3 clés



À partir des exemples ci-dessus, nous avons une idée de la façon dont les arbres rouge-noir assurent l'équilibre. Voici un fait important sur l'équilibre dans les arbres rouge-noir.

Hauteur noire d'un arbre **Rouge**-**Noir**:

La hauteur noire est le nombre de nœuds noirs sur un chemin allant de la racine à une feuille donnée. Les nœuds feuilles sont également comptés des nœuds noirs. À partir des propriétés 3) et 4) ci-dessus, nous pouvons déduire, ***un arbre*** **Rouge**-**Noir** ***de hauteur h a une hauteur noire> = h/2*** .

Le nombre de nœuds entre un nœud et sa feuille descendante la plus éloignée ne dépasse pas le double du nombre de nœuds à la feuille descendante la plus proche.

Chaque arbre rouge noir avec n nœuds a une hauteur <= $2\ log_2(n+1)$

Cela peut être prouvé en utilisant les faits suivants:

1) Pour un arbre binaire général, soit k le nombre minimum de nœuds sur tous les chemins racine à NULL, alors $n > 2k-1$ (Ex. Si k est égale à 3, alors n est égale à au moins 7). Cette expression peut également être écrite comme $k <= log_2(n+1)$.

2) À partir de la propriété 4) des arbres **Rouge**-**Noir** et de la revendication ci-dessus, nous pouvons dire que dans un arbre **Rouge**-**Noir** avec n nœuds, il existe un chemin de racine à feuille avec au plus $log_2(n+1)$ nœuds noirs.

3) À partir de la propriété 3 des arbres **Rouge**-**Noir**, nous pouvons affirmer que le nombre de nœuds noirs dans un arbre **Rouge**-**Noir** est au moins $\lfloor n/2 \rfloor$, où n est le nombre total de nœuds.

À partir de ces 2 points ci-dessus, nous pouvons conclure que l'arbre **Rouge**-**Noir** à n nœuds a une hauteur $h <= 2log_2(n+1)$.

Tâches à Faire

En vous basant sur vos propres recherches et sur les ressources du cours, proposez un programmes avec les fonctions suivantes:

- insertion dans un arbre **Rouge**-**Noir**,
- suppression dans un arbre **Rouge**-**Noir**,
- recherche dans un arbre **Rouge**-**Noir**,
- transformation d'un arbre binaire de recherche (ABR) en un arbre **Rouge**-**Noir**,
- menu de choix pour effectuer les différentes opérations dans un arbre **Rouge**-**Noir**.

Afin de mieux tester l'utilité des arbres **Rouge**-**Noir**,

1. créer un ABR;
2. faire une fonction itérant sur l'insertion successive de 1000 à 10000 valeurs aléatoires dans l'ABR, avec une intervalle de 1000, c'est-à-dire, vous testez d'abord sur 1000, ensuite 2000, ensuite 3000, …, jusqu'à 10000 (remarque:

vous pouvez aller aussi au delà, rien ne vous empêche de tester avec 100000 noeuds);

3. pour chaque test à la question précédente, appeler une fonction de recherche dans cet ABR, et estimer la durée de calcul;
4. comparer les durées en temps de calcul les différentes cardinalités (1000, 2000, 3000, 4000, 5000, …, 10000);
5. Qu'observez-vous ?
6. Refaire les étapes 2 à 4 cette fois ci avec un arbre **Rouge**-**Noir**.
7. Qu'observez-vous?
8. Comparer les durées obtenues avec les ABR et celles obtenues avec les arbre **Rouge**-**Noir**.
9. Quelles conclusions tirez-vous de ces étapes et comparaisons ?