

# RAPPORT

**ABR&ARN**



**SALHI WAF AE**

LICENCE 3 INFORMATIQUE

# SOMMAIRE

## *du rapport*

01	<i>L'arbre rouge noir (ARN)</i>	<u>page 3</u>
02	<i>Exemple d'arbre rouge noir (ARN)</i>	<u>page 3</u>
03	<i>Regles qu'un ARN doit respecter</i>	<u>page 3</u>
04	<i>Ajout d'un élément dans l'ARN:</i>	<u>page 4</u>
05	<i>Suppression d'un élément dans l'ARN:</i>	<u>page 5</u>
06	<i>Recherche d'un élément dans l'ARN:</i>	<u>page 6</u>
07	<i>Implementation</i>	<u>page 7</u>
08	<i>Arbre BINAIRE DE RECHERCHE</i>	<u>page 8</u>
09	<i>Comparaison des performances de l'ABR et de l'ARN</i>	<u>page 8</u>

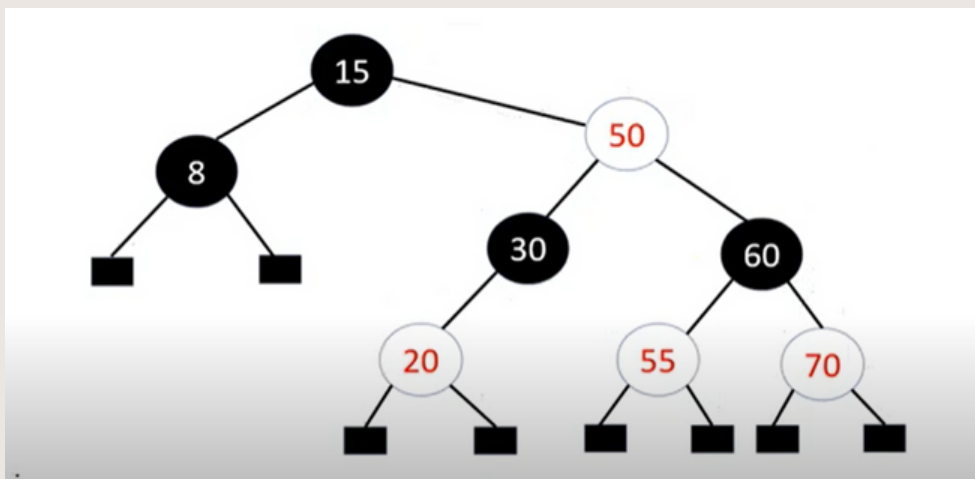
## 01

## L'ARBRE ROUGE NOIR (ARN):

Un arbre rouge-noir (ARN) est un type d'arbre binaire de recherche (ABR) où chaque nœud est soit rouge, soit noir. Si un nœud est rouge, tous ses enfants sont noirs, ce qui empêche d'avoir deux nœuds rouges consécutifs. L'arbre est structuré de sorte que chaque chemin d'un nœud à ses descendants (incluant ceux ayant 0 ou 1 enfant) possède le même nombre de nœuds noirs. Cette structure garantit un équilibre qui permet des opérations de recherche, insertion, et suppression en temps logarithmique.

## 02

## EXEMPLE D'ARBRE ROUGE NOIR (ARN):



## 03

## REGLES QU'UN ARN DOIT RESPECTER :

1. Chaque nœud est soit rouge, soit noir.
2. La racine est noire.
3. Les feuilles  $\blacksquare$  sont noires.
4. Si un nœud est rouge, alors ses deux fils sont noirs.
5. Pour chaque nœud, tous les chemins le reliant à des feuilles contiennent le même nombre de nœuds noirs.

l'ajout se fait par l'ajout classique de l'ABR avec quelques modifications:

- on remplace les null par ☒
- on colorie le nouveau nœud z en rouge
- comme ce coloriage risque de violer certaines propriétés RN, on appelle dans le code : `ajouterCorrection()`

les propriétés que ajouter risque de violer

- 1.OK
- 2.Si l'arbre était vide z devient sa racine. C'est facile à réparer, il suffit de le colorier en noir
- 3.OK
- 4.Si y (le père de z) est rouge, cette propriété est violée.
- 5.OK

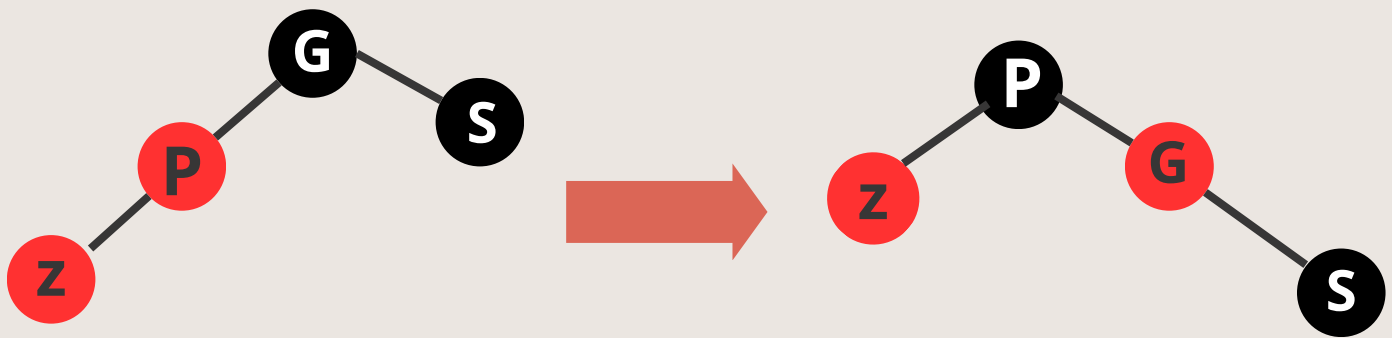
**Pour réparer ces propriétés et obtenir un ARN équilibré, on procède comme suit :**

-Si le parent du nœud inséré est noir et que le nouveau nœud est rouge, alors l'arbre conserve ses propriétés d'arbre rouge-noir.

-Si le nouveau est la racine , il doit être coloré en noir si le nouveau nœud n'est pas une racine dans ce cas on recolorie le père et le grand père et l'oncle du nouveau

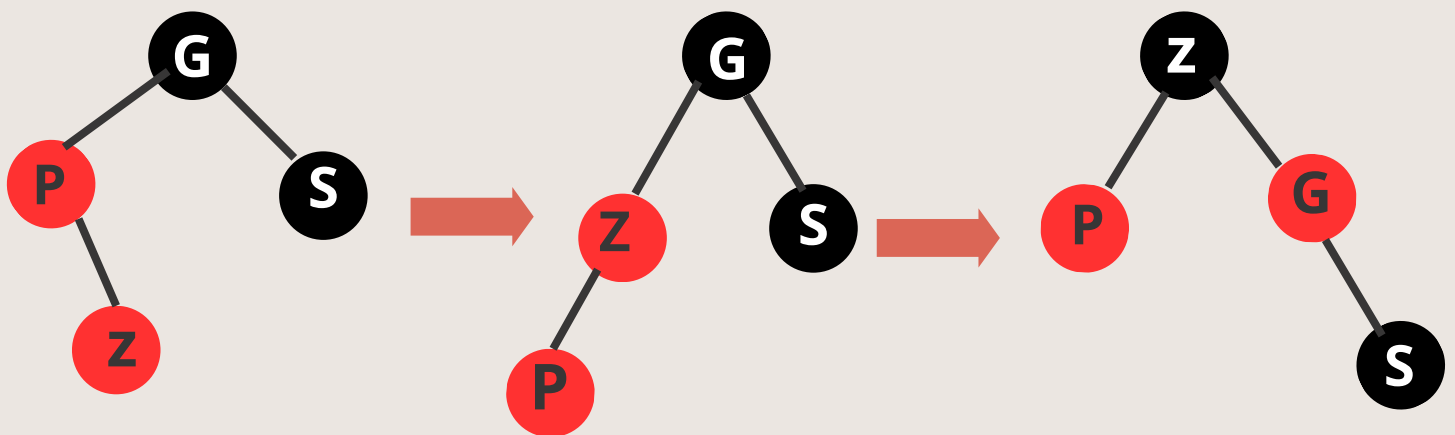


-Si l'oncle du nouveau noeud est noir (sous forme de ligne) on fait une rotation (droite ou gauche) avec une recoloration :



idem, pour l'autre sens

-Si l'oncle du nouveau noeud est noir (sous forme de triangle) on commence réaligner la forme pour appliquer par la suite la démarche du cas précédent



idem, pour l'autre sens

## 05 SUPPRESSION D'UN ÉLÉMENT DANS L'ARN:

Pour supprimer un élément dans un arbre rouge-noir, on commence par procéder comme dans un arbre binaire de recherche. Une fois le nœud supprimé, plusieurs situations peuvent se présenter en fonction de la couleur du nœud et de ses fils. Voici les différents cas à considérer

### Cas 1

Si le nœud supprimé est rouge, aucune modification supplémentaire n'est nécessaire, car les propriétés de l'arbre rouge-noir sont automatiquement préservées.

### Cas 2

Si le nœud supprimé est noir mais possède un fils rouge, il suffit de colorer ce fils en noir. Cela rétablit la hauteur noire et garantit que l'arbre reste équilibré.

### Cas 3

Si le nœud supprimé est la racine et qu'il n'a aucun fils rouges, la hauteur noire de l'arbre diminue de 1.

### Cas 4

Si le nœud supprimé est noir, n'est pas la racine, et n'a aucun fils rouges, l'arbre devient déséquilibré, car une différence de hauteur noire apparaît entre les sous-arbres. Ce déséquilibre nécessite des ajustements spécifiques pour rétablir l'équilibre et respecter les propriétés de l'arbre rouge-noir.

## 06

## RECHERCHE D'UN ÉLÉMENT DANS L'ARN:

La recherche d'un élément dans un arbre rouge-noir suit la même procédure qu'un arbre binaire de recherche. On commence à la racine et compare l'élément recherché avec la valeur du nœud courant, puis on se déplace vers le sous-arbre gauche ou droit selon le résultat. Ce processus se répète jusqu'à trouver l'élément ou arrivée à une feuille. Grâce à la structure équilibrée de l'arbre rouge-noir, la recherche s'effectue en  $O(\log n)$  : car à partir de la racine on va décider quel sous arbre on va parcourir pour trouver l'élément rechercher.

**1a Classe Principale :**

- `T extends Comparable<? super T>` impose que le type `T` doit être comparable, permettant l'ordre des éléments.
- `extends AbstractCollection<T>` permet à la classe de bénéficier des fonctionnalités de base d'une collection Java, comme la gestion de la taille et les itérations.

**Constructeur:**

- Premier constructeur : Crée un arbre rouge-noir vide avec l'ordre naturel par défaut.
- Deuxième constructeur : Crée un arbre rouge-noir vide avec un comparateur personnalisé pour l'ordre des éléments.
- Troisième constructeur : Crée un arbre rouge-noir vide, puis y ajoute les éléments d'une collection donnée, en utilisant le comparateur défini (ou l'ordre naturel si aucun comparateur spécifique n'est fourni).

**MÉTHODES :**

- ➔ Méthode d'Insertion
- ➔ Méthode d'équilibre après ajout : qui permet d'équilibrer l'arbre après l'ajout en cas ou l'une de violence de l'une des règles de l'ARN
- ➔ Méthode de suppression : suis la même logique que celle de l'ABR
- ➔ Méthode d'équilibre après suppression : idem que celle de l'ajout
- ➔ Méthode de rotation : Effectuent des rotations gauche et droite pour rééquilibrer l'arbre.
- ➔ Itérateur: Permet de parcourir l'arbre en ordre croissant.

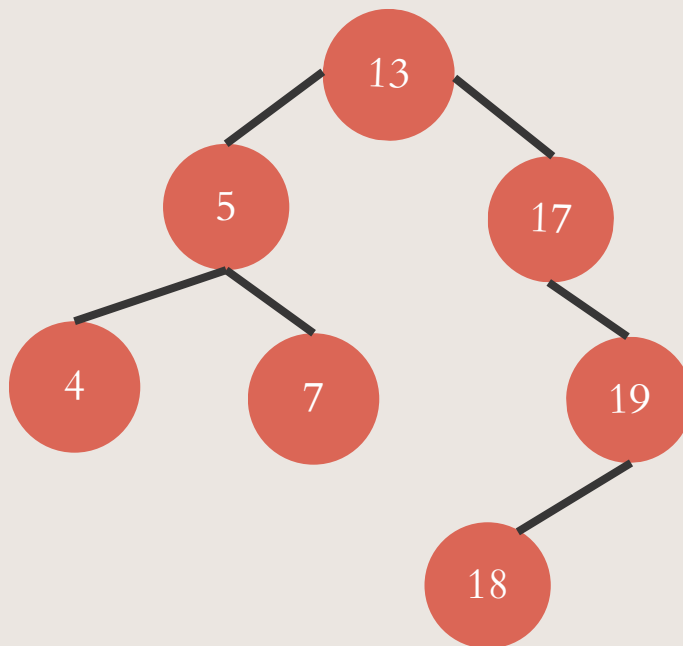
**3. COMPLEXITÉ DES MÉTHODES**

Les opérations:

`rechercher()`, `equilibrerApresAjout(Noeud n)`, `delete(Noeud z)`, `equilibrerApresSuppression(Noeud x)`, `add(T e)`: s'exécutent en  $O(\log n)$

*Un arbre binaire de recherche (ou arbre binaire ordonné) est une structure de données qui est une forme spécialisée d'arbre binaire. Il est constitué de nœuds où chaque nœud contient une valeur et deux références (ou pointeurs) vers ses enfants gauche et droit.*

## ➔ EXEMPLE D'ARBRE BINAIRE



## COMPARAISON DES PERFORMANCES DE L'ABR ET DE L'ARN

### COMPARAISON TEMPS CONSTRUCTION:

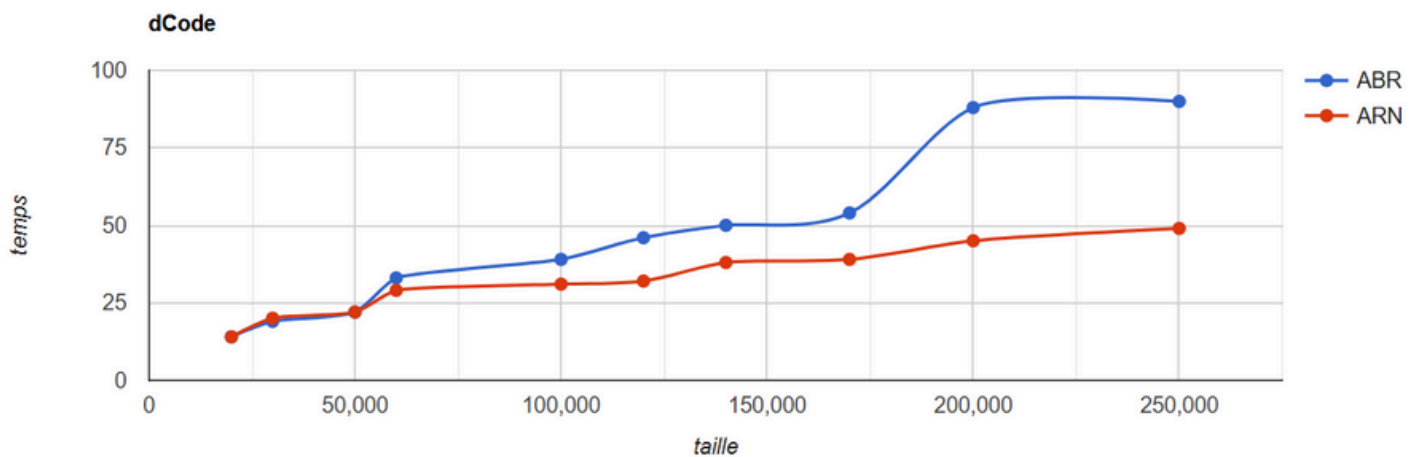
	abscisse x →	ordonnee y ou f(x) ↑
1	20000	14
2	30000	20
3	50000	22
4	60000	29
5	100000	31
6	120000	32
7	140000	38
8	170000	39
9	200000	45
10	250000	49

TABLEAU REPRÉSENTANT LE TEMPS DE CONSTRUCTION DE L'ARN EN FONCTION DE SA TAILLE



	abscisse x →	ordonnee y ou f(x) ↑
1	20000	14
2	30000	19
3	50000	22
4	60000	33
5	100000	39
6	120000	46
7	140000	50
8	170000	54
9	200000	88
10	250000	90

TABLEAU REPRÉSENTANT LE TEMPS DE CONSTRUCTION DE L'ABR EN FONCTION DE SA TAILLE



GRAPHIQUE REPRÉSENTANT LE TEMPS DE CONSTRUCTION DES ARBRES ABR ET ARN EN FONCTION DE LEUR TAILLE



*Les tests réalisés sur des arbres montrent que les arbres Rouge-Noir sont plus rapides à construire que les arbres binaires de recherche. Cependant, cette différence de performance devient vraiment visible seulement lorsque la taille des arbres est très grande.*

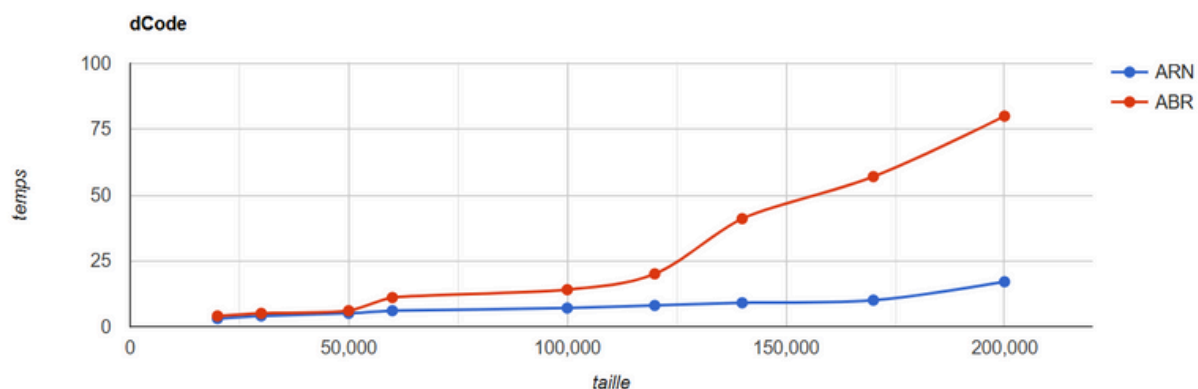
## COMPARAISON TEMPS DE RECHERCHE :

	abscisse x →	ordonnee y ou f(x) ↑
1	20000	4
2	30000	5
3	50000	6
4	60000	11
5	100000	14
6	120000	20
7	140000	41
8	170000	57
9	200000	80

TABLEAU REPRÉSENTANT LE TEMPS DE RECHERCHE DE L'ABR EN FONCTION DE SA TAILLE

	abscisse x →	ordonnee y ou f(x) ↑
1	20000	3
2	30000	4
3	50000	5
4	60000	6
5	100000	7
6	120000	8
7	140000	9
8	170000	10
9	200000	17

TABLEAU REPRÉSENTANT LE TEMPS DE RECHERCHE DE L'ARN EN FONCTION DE SA TAILLE



GRAPHIQUE REPRÉSENTANT LE TEMPS DE RECHERCHE DES ARBRES ABR ET ARN EN FONCTION DE LEUR TAILLE

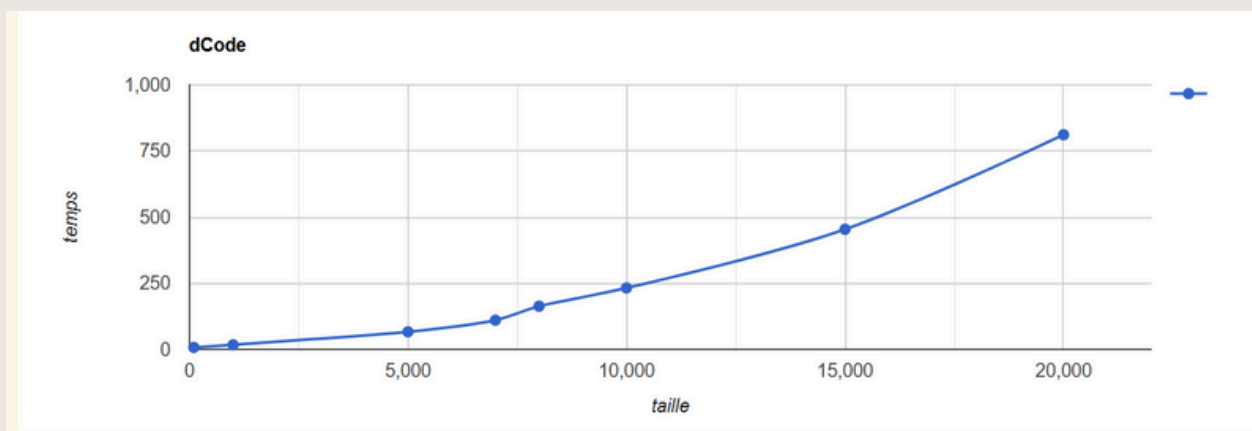


*En comparant les deux courbes, on remarque que la recherche se fait plus rapidement avec l'ARN, même pour des tailles d'arbres relativement petites, tandis qu'avec l'ABR, la recherche peut être plus lente.*

## 2. LE CAS DÉFAVORABLE :

	abscisse x →	ordonnee y ou f(x) ↑
1	100	7
2	1000	17
3	5000	66
4	7000	110
5	8000	163
6	10000	232
7	15000	454
8	20000	812

TABLEAU REPRÉSENTANT LE TEMPS DE CONSTRUCTION DE L'ABR EN CAS DÉFAVORABLE



GRAPHIQUE REPRÉSENTANT LE TEMPS DE RECHERCHE DES ARBRES ABR ET ARN EN FONCTION DE LEUR TAILLE



*lorsque les clés sont ajoutées dans un ordre croissant, du plus petit au plus grand. Selon le graphique et le tableau ci-dessous, on observe que la construction de l'arbre prend plus de temps lorsque les clés sont ordonnées, comparé à un ajout de clés aléatoires.*