FICHE DE SYNTHÈSE
Les Arbres Binaires

NSI Terminale

I. Définitions essentielles

Arbre : Structure de données hiérarchique constituée de nœuds reliés par des arêtes, avec un nœud racine unique et sans cycles.

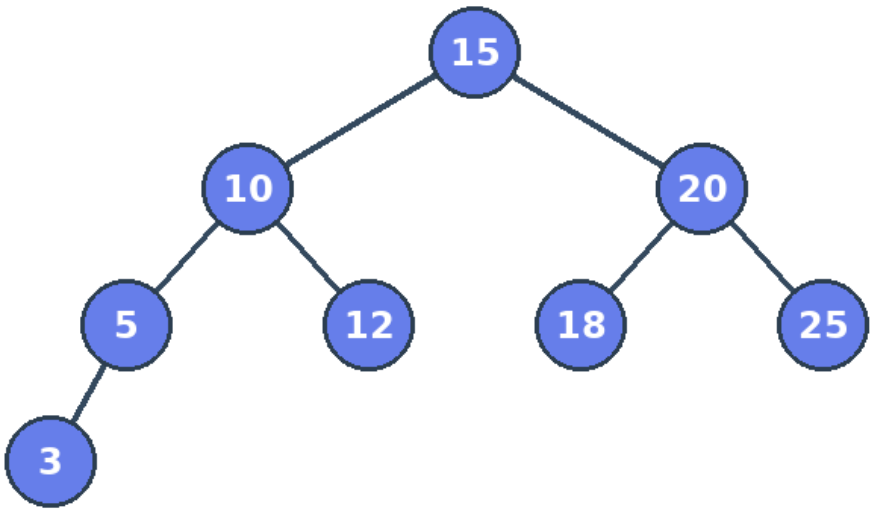
Arbre binaire : Arbre où chaque nœud possède au plus deux enfants (fils gauche et fils droit).

Arbre binaire de recherche (ABR) : Arbre binaire vérifiant : pour tout nœud, valeurs du sous-arbre gauche < valeur du nœud ≤ valeurs du sous-arbre droit.

II. Vocabulaire

Terme	Définition
Racine	Nœud de départ de l'arbre, sans parent
Feuille	Nœud sans enfant
Hauteur	Longueur du plus long chemin de la racine à une feuille
Taille	Nombre total de nœuds dans l'arbre
Profondeur	Distance d'un nœud à la racine (niveau)

III. Exemple d'arbre binaire de recherche



→ **Racine :** 15 **Hauteur :** 3 **Taille :** 7 **Feuilles :** 3, 12, 18, 25

IV. Parcours d'arbres

A. Parcours en profondeur (récursifs)

- **Préfixe :** Racine → Gauche → Droit → 15 10 5 3 12 20 18 25
- **Infixe :** Gauche → Racine → Droit → 3 5 10 12 15 18 20 25 (ordre croissant !)
- **Suffixe :** Gauche → Droit → Racine → 3 5 12 10 18 25 20 15

B. Parcours en largeur (itératif avec file)

Niveau par niveau, de gauche à droite → 15 10 20 5 12 18 25 3

V. Algorithmes en pseudocode

1. Calculer la taille

```
Fonction taille(arbre)
    Si arbre est vide Alors
        Retourner 0
    Sinon
        Retourner 1 + taille(gauche) + taille(droit)
    Fin Si
Fin Fonction
```

2. Calculer la hauteur

```
Fonction hauteur(arbre)
    Si arbre est vide Alors
        Retourner -1
    Sinon
        h_gauche ← hauteur(gauche)
        h_droit ← hauteur(droit)
        Retourner 1 + max(h_gauche, h_droit)
    Fin Si
Fin Fonction
```

3. Parcours infixe

```
Fonction parcours_infixe(arbre)
    Si arbre n'est pas vide Alors
        parcours_infixe(gauche)
        Afficher(valeur)
        parcours_infixe(droit)
    Fin Si
Fin Fonction
```

4. Parcours en largeur

```
Fonction parcours_largeur(arbre)
    Si arbre est vide Alors Retourner
    file ← nouvelle File()
    enfiler(arbre, file)
    Tant que file n'est pas vide Faire
        nœud ← défiler(file)
        Afficher(nœud.valeur)
        Si nœud.gauche existe Alors
            enfiler(nœud.gauche, file)
        Si nœud.droit existe Alors
            enfiler(nœud.droit, file)
    Fin Tant que
Fin Fonction
```

5. Recherche dans un ABR

```
Fonction recherche(arbre, valeur)
    Si arbre est vide Alors Retourner Faux
    Si arbre.valeur = valeur Alors Retourner Vrai
    Sinon Si valeur < arbre.valeur Alors
        Retourner recherche(gauche, valeur)
    Sinon
        Retourner recherche(droit, valeur)
    Fin Si
Fin Fonction
```

6. Insertion dans un ABR

```
Fonction inserer(arbre, valeur)
    Si arbre est vide Alors
        Retourner nouveau_nœud(valeur)
    Si valeur < arbre.valeur Alors
        arbre.gauche ← inserer(arbre.gauche, valeur)
    Sinon
        arbre.droit ← inserer(arbre.droit, valeur)
    Retourner arbre
Fin Fonction
```

VI. Complexité des opérations sur ABR

Opération	Arbre équilibré	Arbre dégénéré
Recherche	`O(log n)`	`O(n)`
Insertion	`O(log n)`	`O(n)`
Suppression	`O(log n)`	`O(n)`

VII. Points clés à retenir

- Un arbre binaire : chaque nœud a au plus 2 enfants
- Hauteur d'un arbre vide : -1, arbre réduit à une feuille : 0
- ABR : sous-arbre gauche < racine ≤ sous-arbre droit
- **Parcours infixe d'un ABR → valeurs en ordre croissant**
- Complexité des opérations sur ABR : O(h) où h = hauteur
- Arbre équilibré → O(log n), arbre dégénéré → O(n)
- L'ordre d'insertion influence la forme et les performances