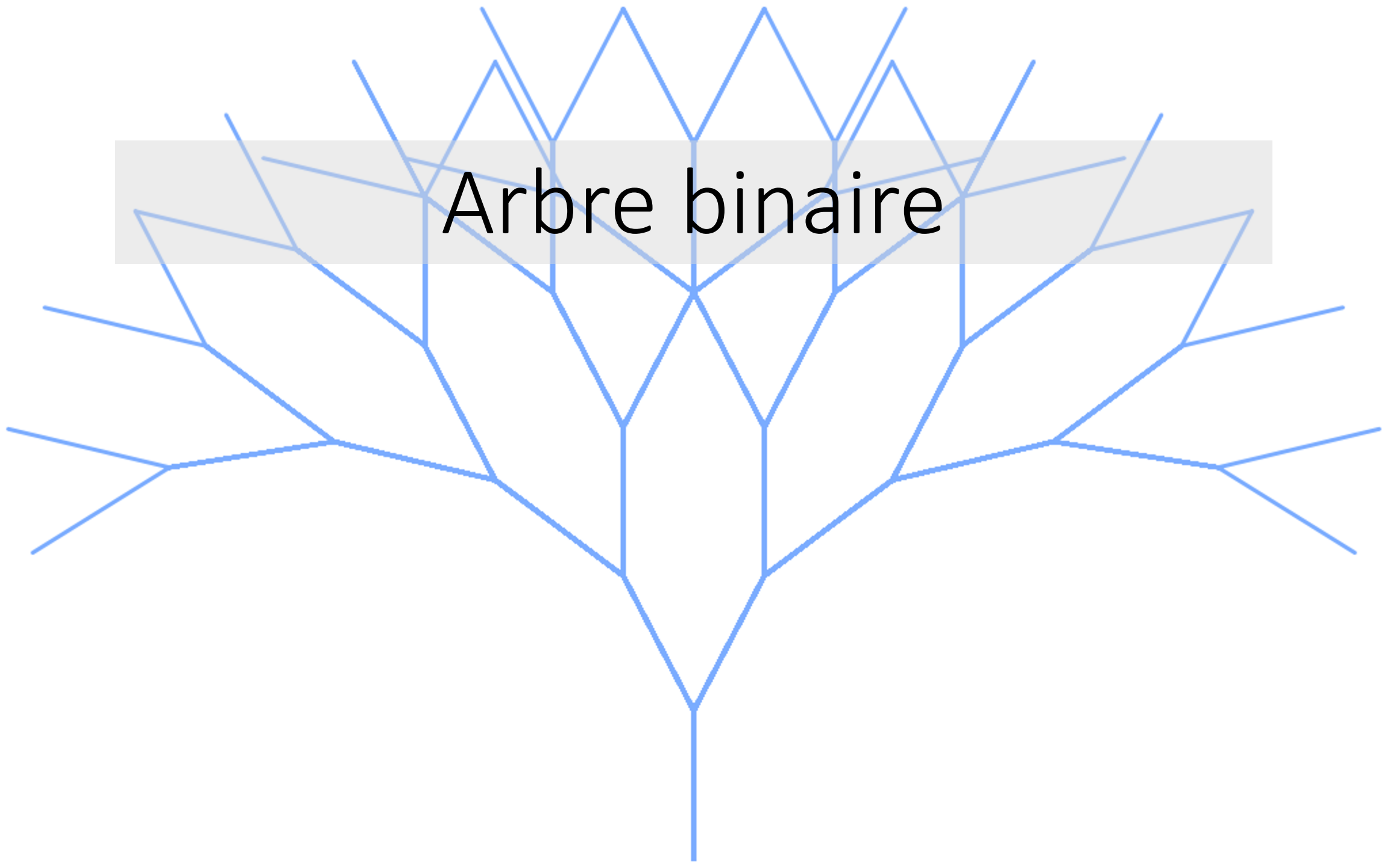


# Arbre binaire

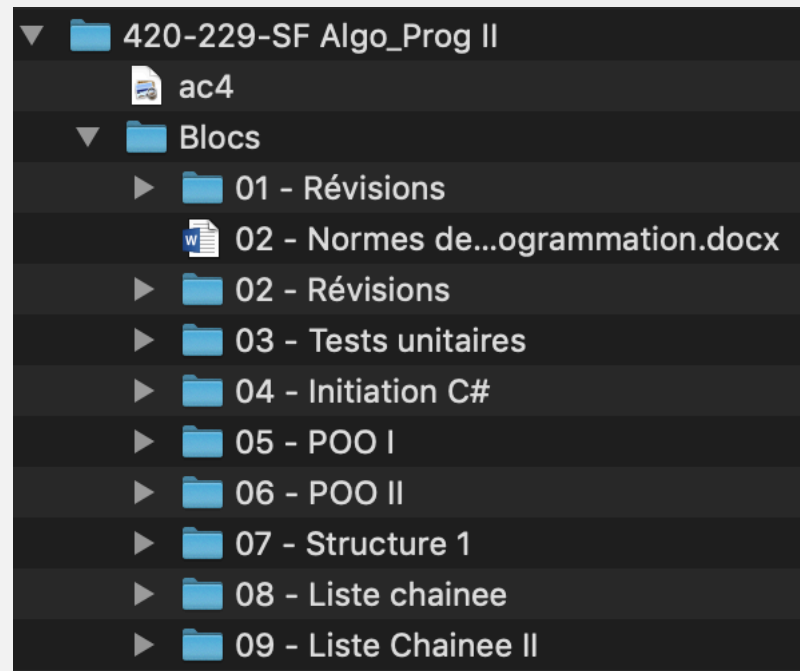


# Objectifs


- Vocabulaire
- Structure des arbres binaires
- Hauteur d'un arbre binaire
- Arbre binaire de recherche
- Parcours
- Insertion

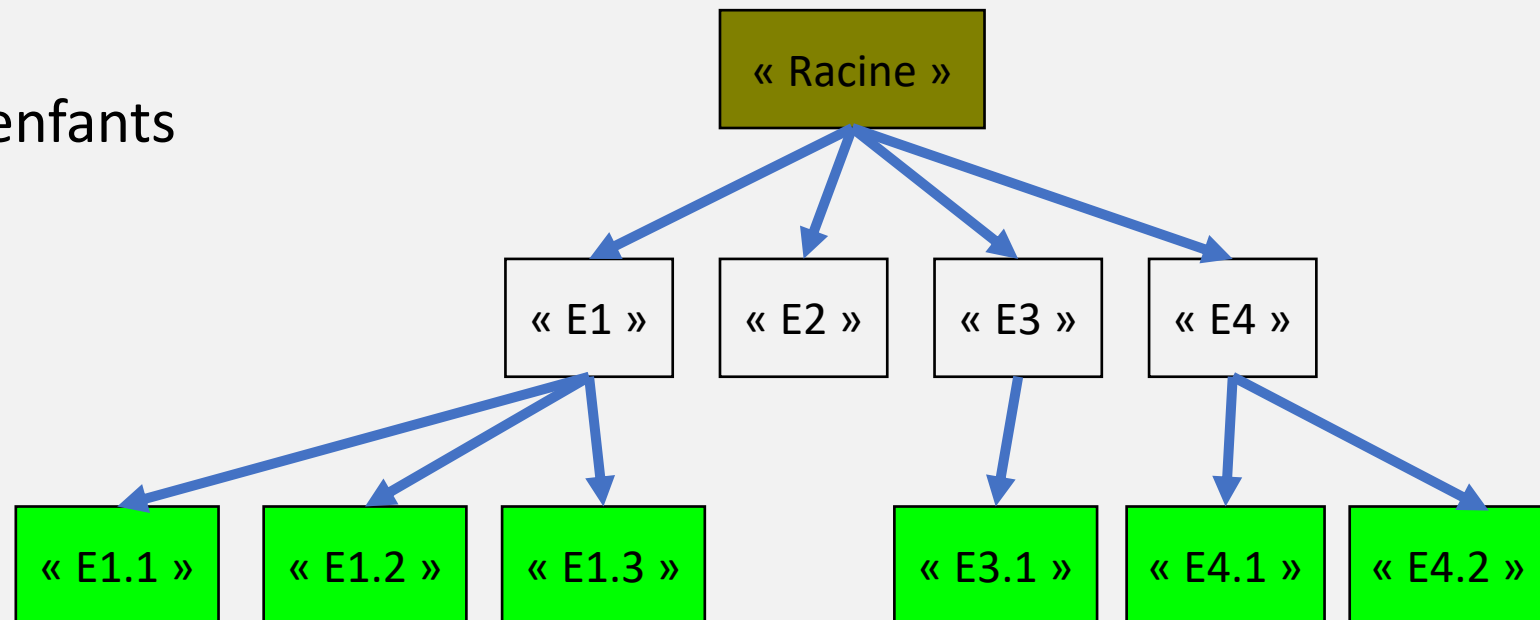
# Définition

- Un arbre est une structure de données qui permet de représenter des éléments sous forme hiérarchique
  - Exemple la structure d'un répertoire



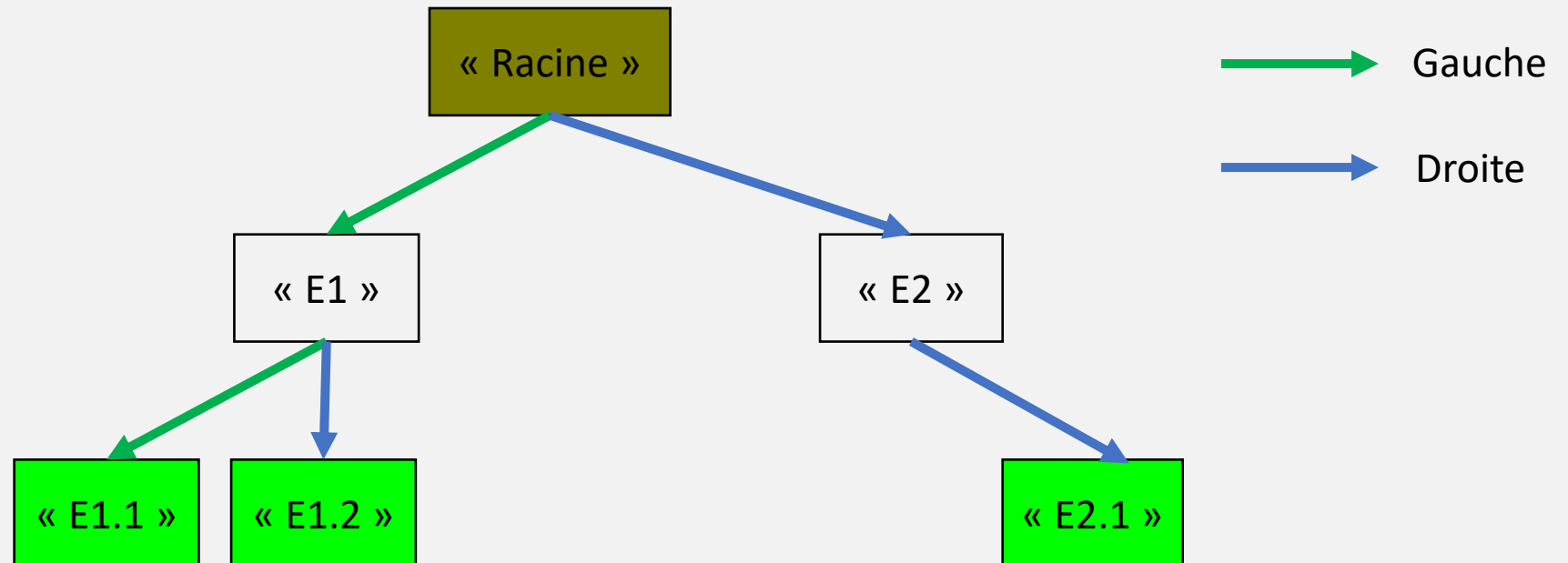
# Arbre - Définition

- Un arbre n-aire est un ensemble de nœuds liés par une relation de parenté
- Un nœud porte une valeur et ses liens vers ses enfants 
- Chaque nœud a un parent
- Chaque arbre a un nœud particulier qui n'a pas de parent.
  - Il est appelé **racine**
- Certains nœuds n'ont pas d'enfants
  - Ils sont appelés **feuilles**



# Arbre binaire – Définition

- Un arbre binaire est un arbre n-aire particulier :
  - Chaque nœud à **au plus deux enfants**
  - Ces deux enfants sont généralement appelés Gauche et Droite



# Exercice 1 – Structure ArbreBinaire

- Écrivez la classe « NoeudArbreBinaire » qui permet de représenter les liens vers les différents enfants et qui permet de stocker une valeur d'un type paramétré
- Écrivez la classe « ArbreBinaire » qui contient simplement le nœud racine

# Exercice 1 – Structure ArbreBinaire – Solution

```
public class NoeudArbreBinaire<TypeElement>
{
    public NoeudArbreBinaire<TypeElement> Gauche { get; set; }
    public NoeudArbreBinaire<TypeElement> Droit { get; set; }

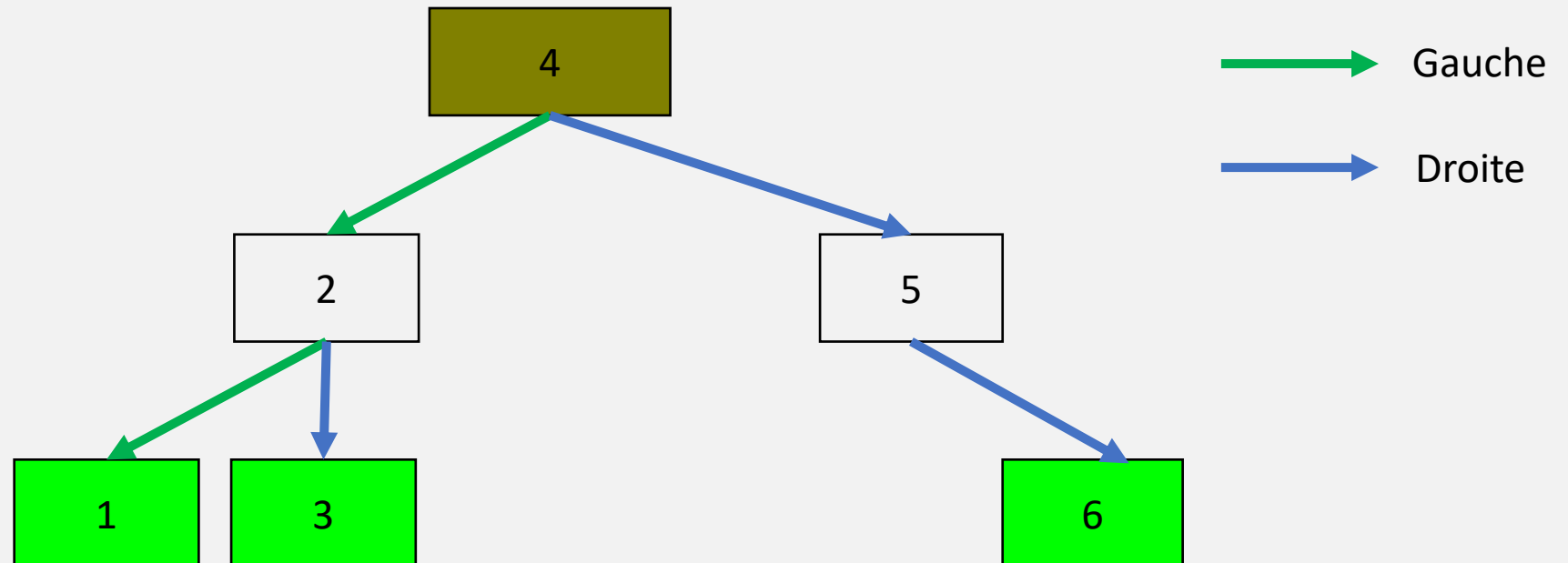
    public TypeElement Valeur { get; set; }
}
```

```
public class ArbreBinaire<TypeElement>
{
    public NoeudArbreBinaire<TypeElement> Racine { get; set; }

    // Méthodes de l'arbre
}
```

## Exercice 2 – Données de tests

- Dans la nouvelle classe « `GenerateurArbreBinaire` », écrivez la méthode statique « `ExempleArbre1` » qui renvoie un arbre binaire d'entiers qui est exactement le suivant :





# Exercice 2 – Données de tests – Solution

```
return new ArbreBinaire<int>() {  
    Racine = new NoeudArbreBinaire<int>() {  
        Gauche = new NoeudArbreBinaire<int>() {  
            Gauche = new NoeudArbreBinaire<int>() {  
                Gauche = null,  
                Droit = null,  
                Valeur = 1  
            },  
            Droit = new NoeudArbreBinaire<int>() {  
                Gauche = null,  
                Droit = null,  
                Valeur = 3  
            },  
            Valeur = 2  
        },  
    },  
};
```

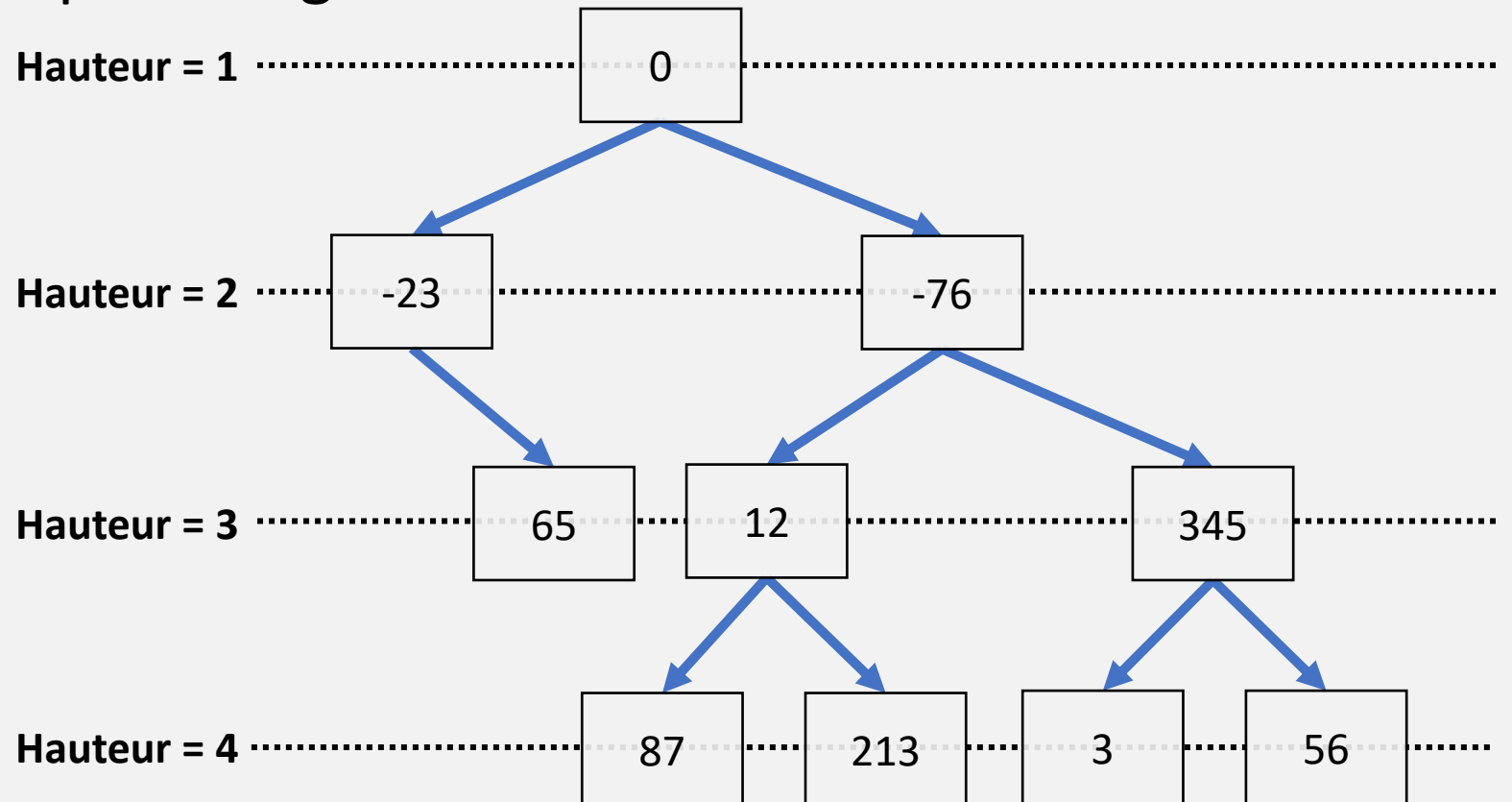
```
        Droit = new NoeudArbreBinaire<int>() {  
            Gauche = null,  
            Droit = new NoeudArbreBinaire<int>() {  
                Gauche = null,  
                Droit = null,  
                Valeur = 6  
            },  
            Valeur = 5  
        },  
        Valeur = 4  
    }  
};
```

# Arbre binaire – Hauteur (ou profondeur)

- C'est le nombre de nœud à traverser en partant de la racine pour atteindre les feuilles les plus éloignées

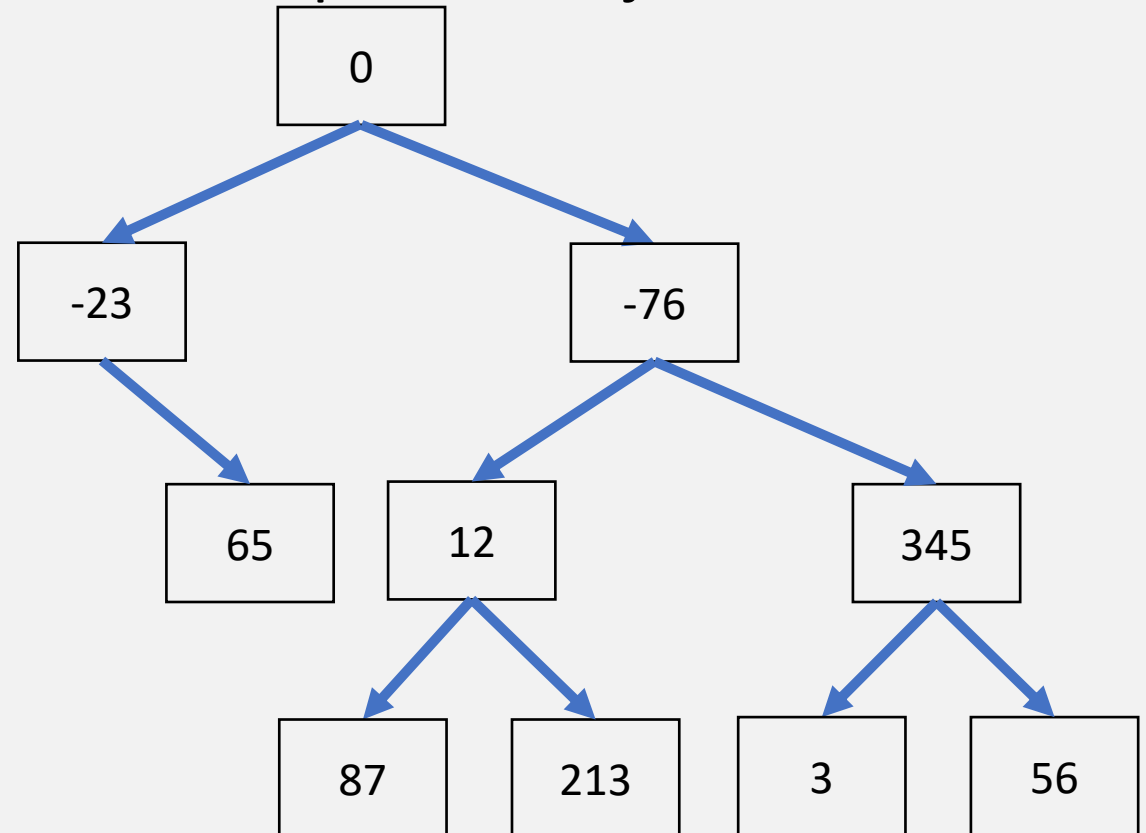
- Exemple :

⇒ Profondeur = 4

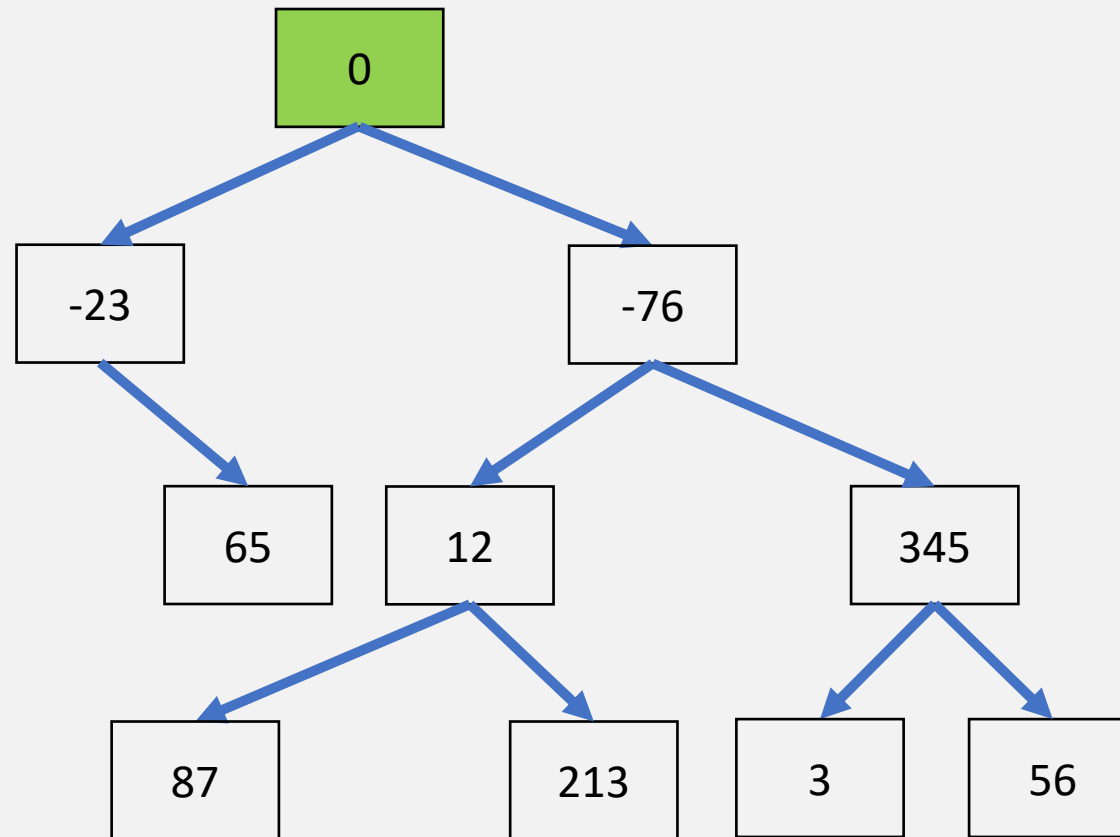


# Arbre binaire – Hauteur (ou profondeur)

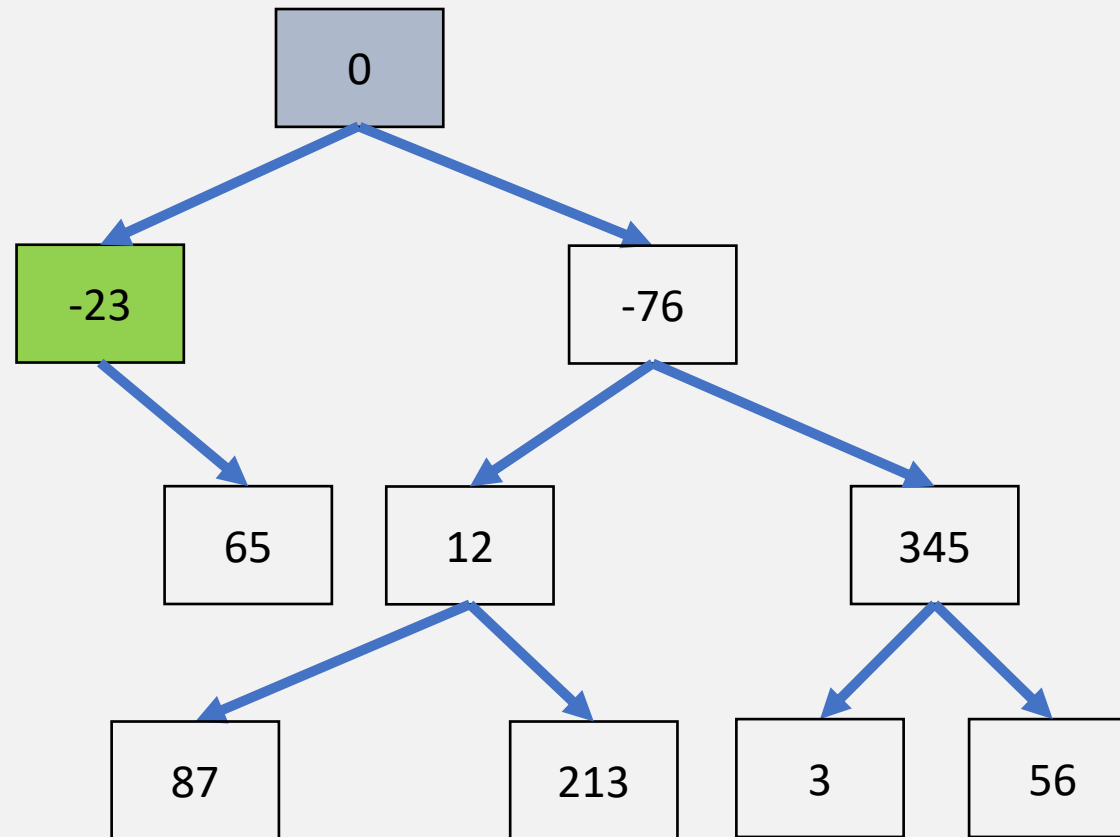
- On obtient la hauteur en parcourant l'arbre et en faisant remonter la hauteur maximale des nœuds enfants à laquelle on ajoute 1
- Les feuilles débutent à 1



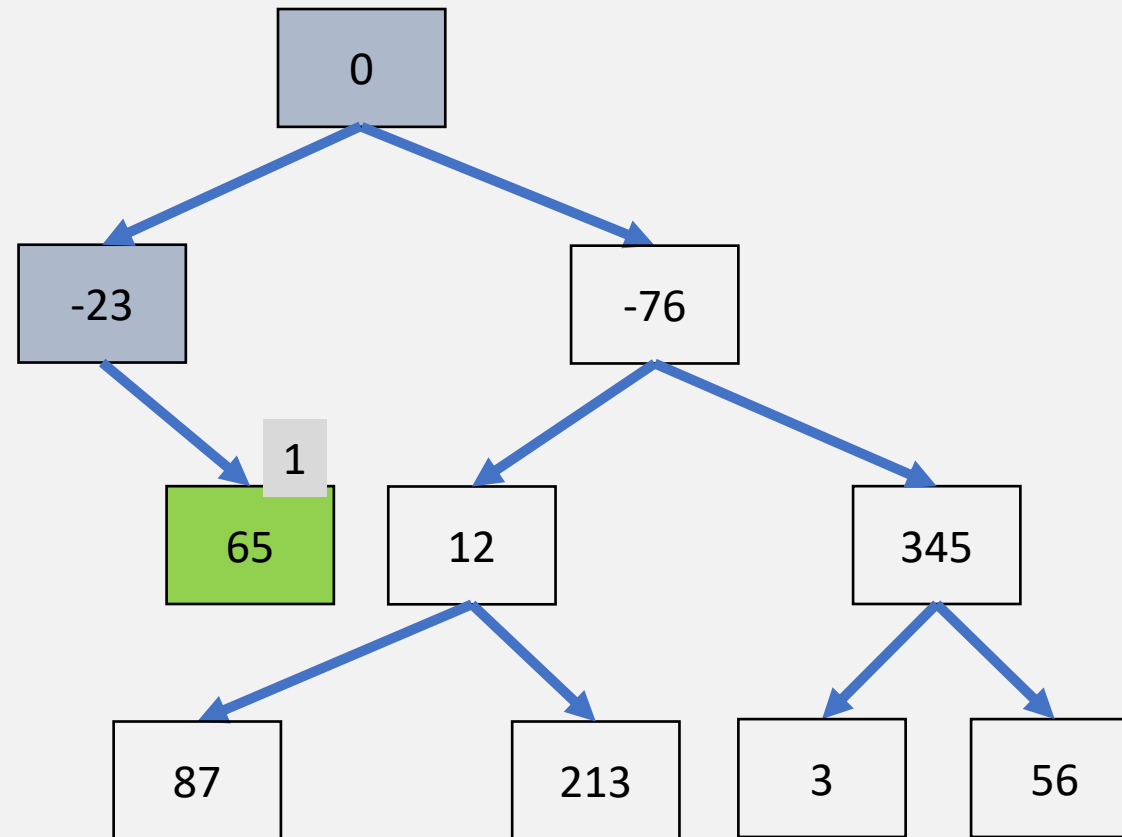
# Arbre binaire – Hauteur (ou profondeur)



# Arbre binaire – Hauteur (ou profondeur)

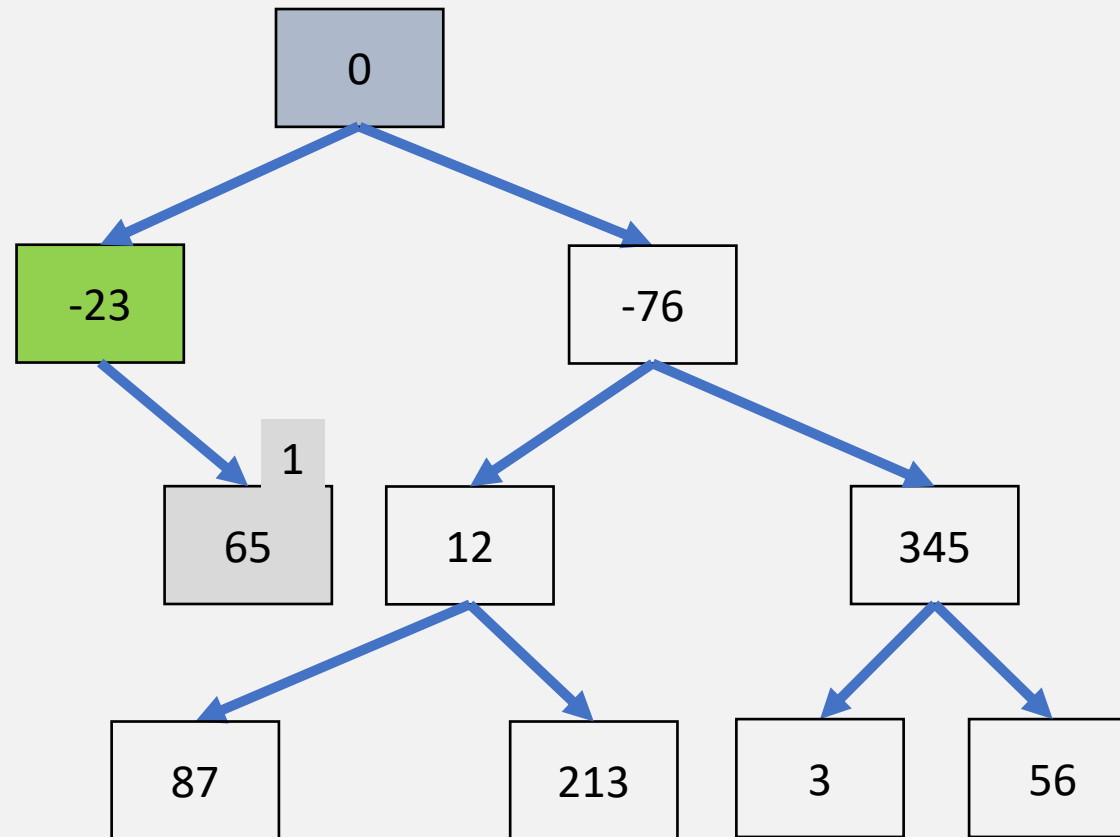


# Arbre binaire – Hauteur (ou profondeur)

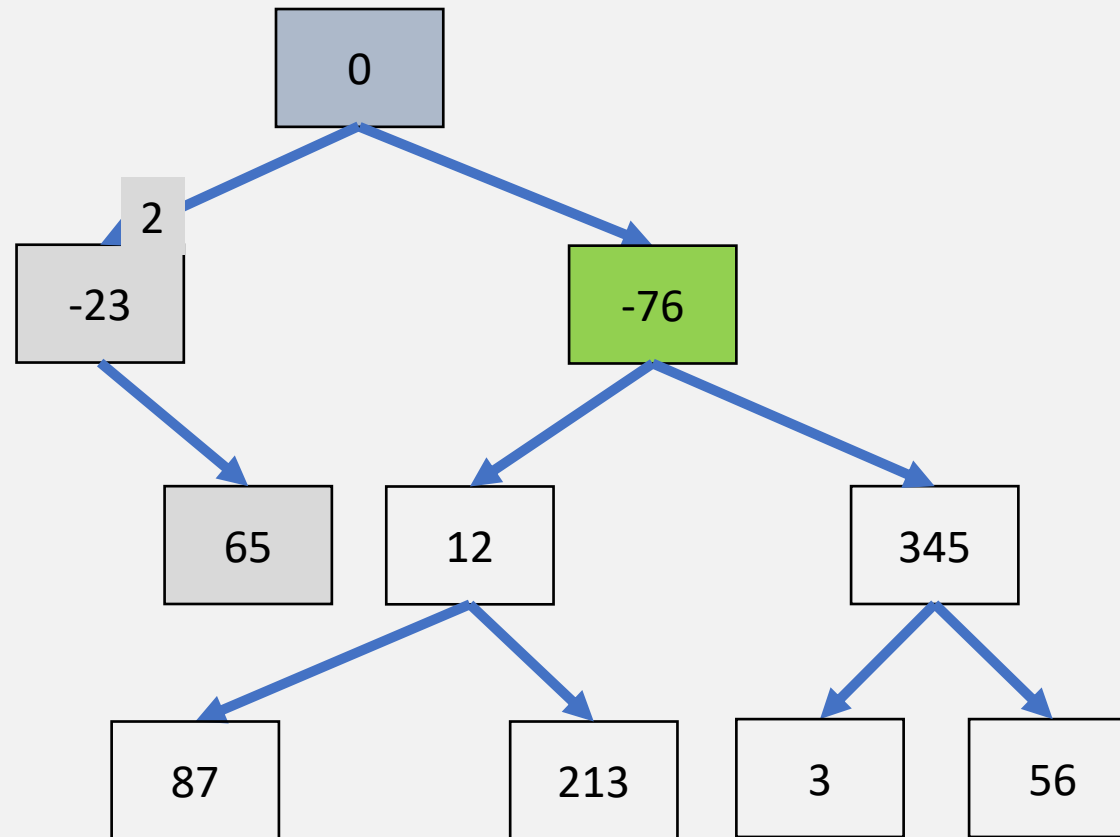


# Arbre binaire – Hauteur (ou profondeur)

$$\text{Max}([1]) + 1 = 2$$

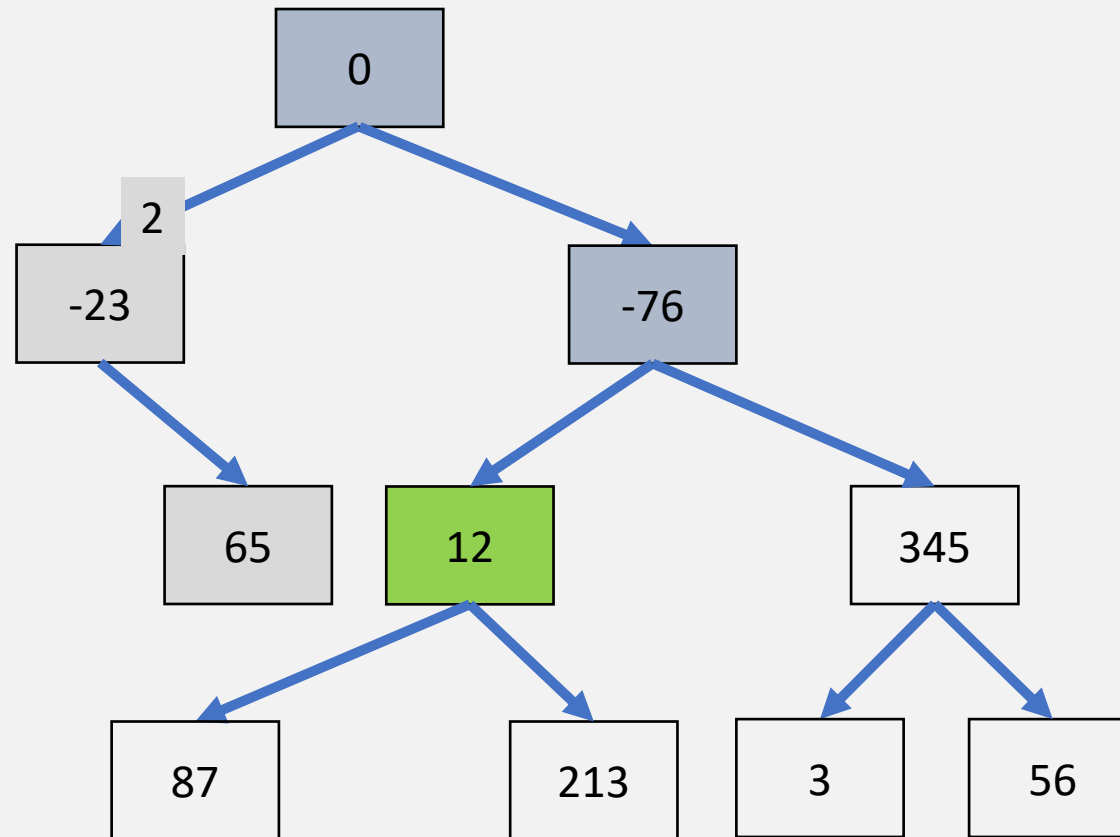


# Arbre binaire – Hauteur (ou profondeur)

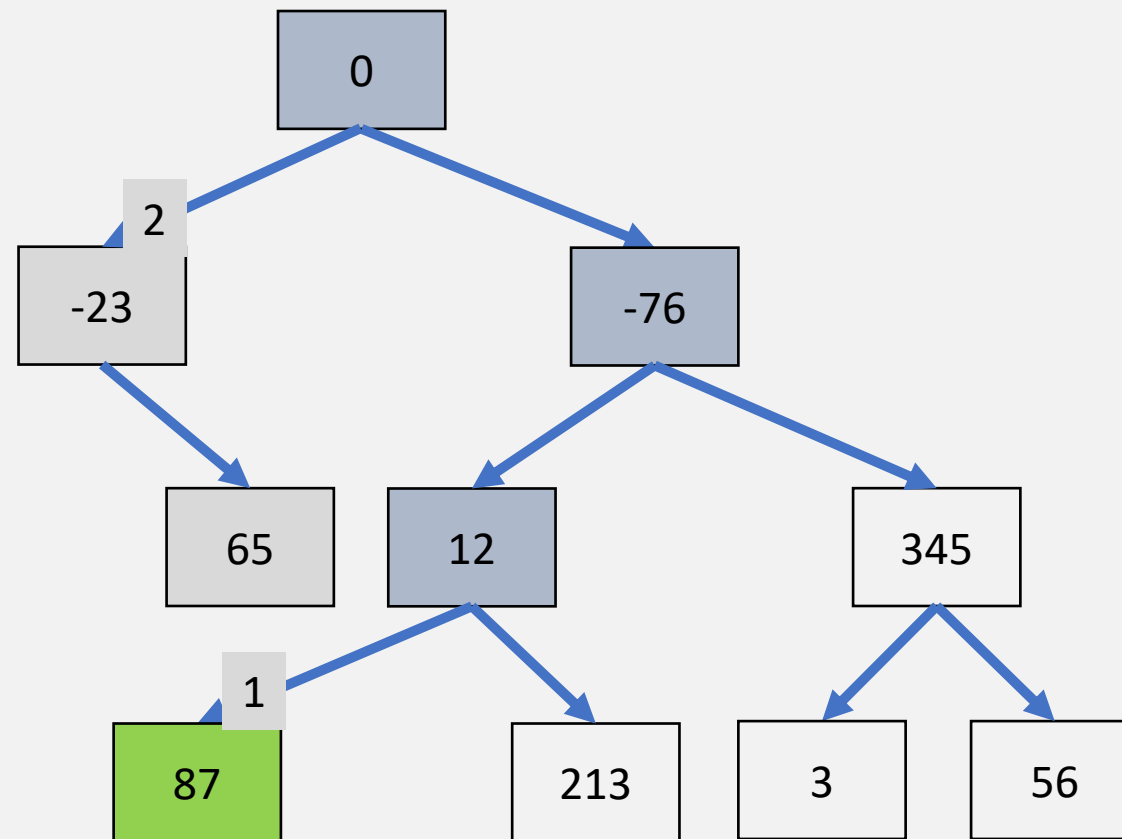




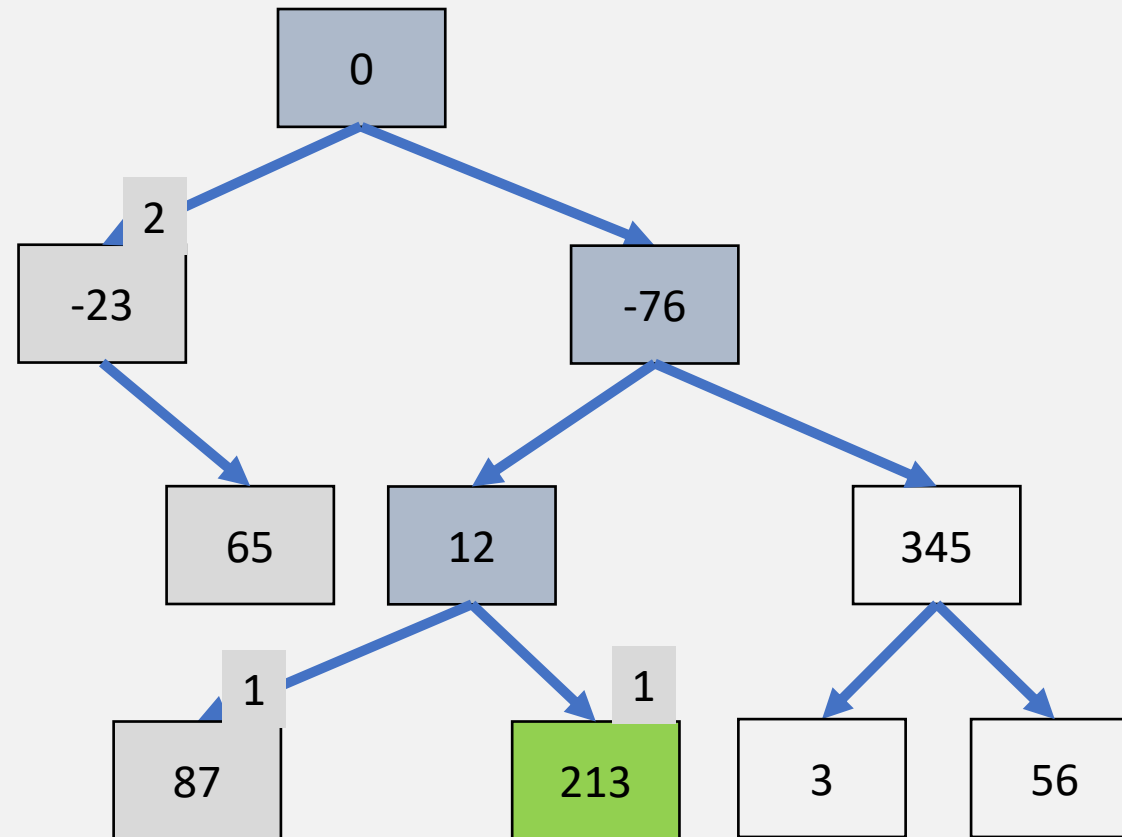
# Arbre binaire – Hauteur (ou profondeur)



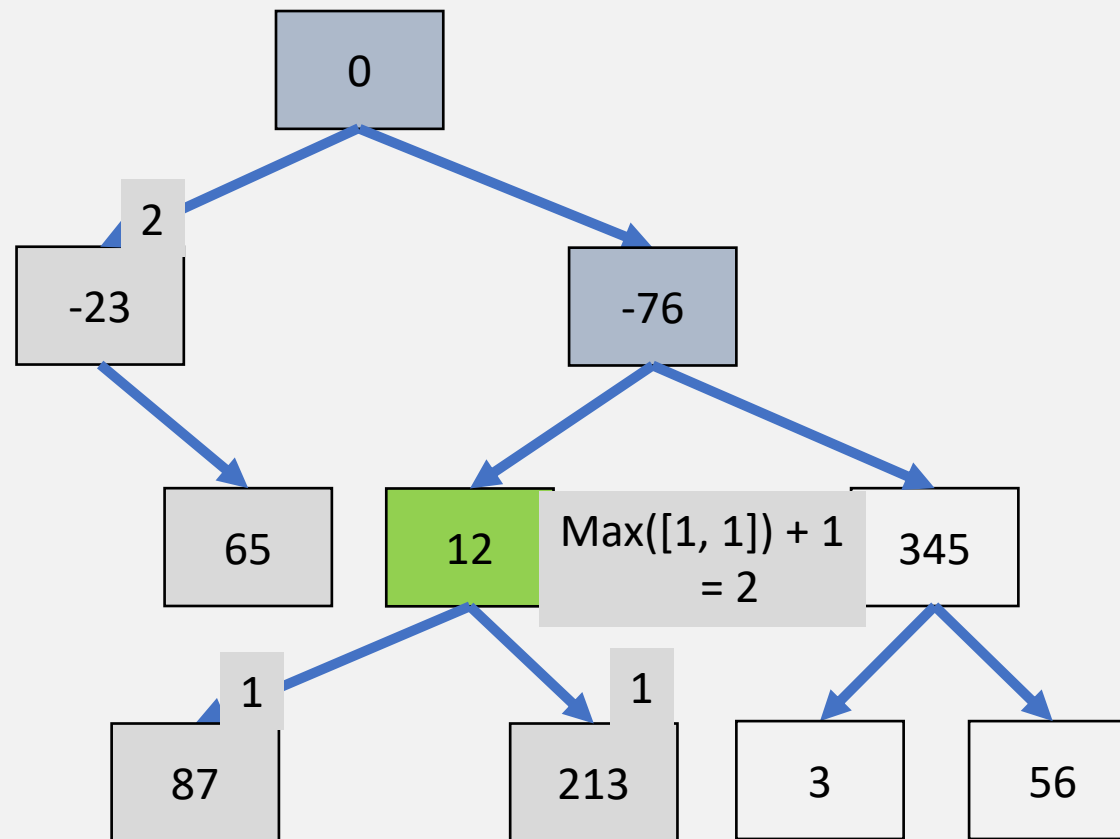
# Arbre binaire – Hauteur (ou profondeur)



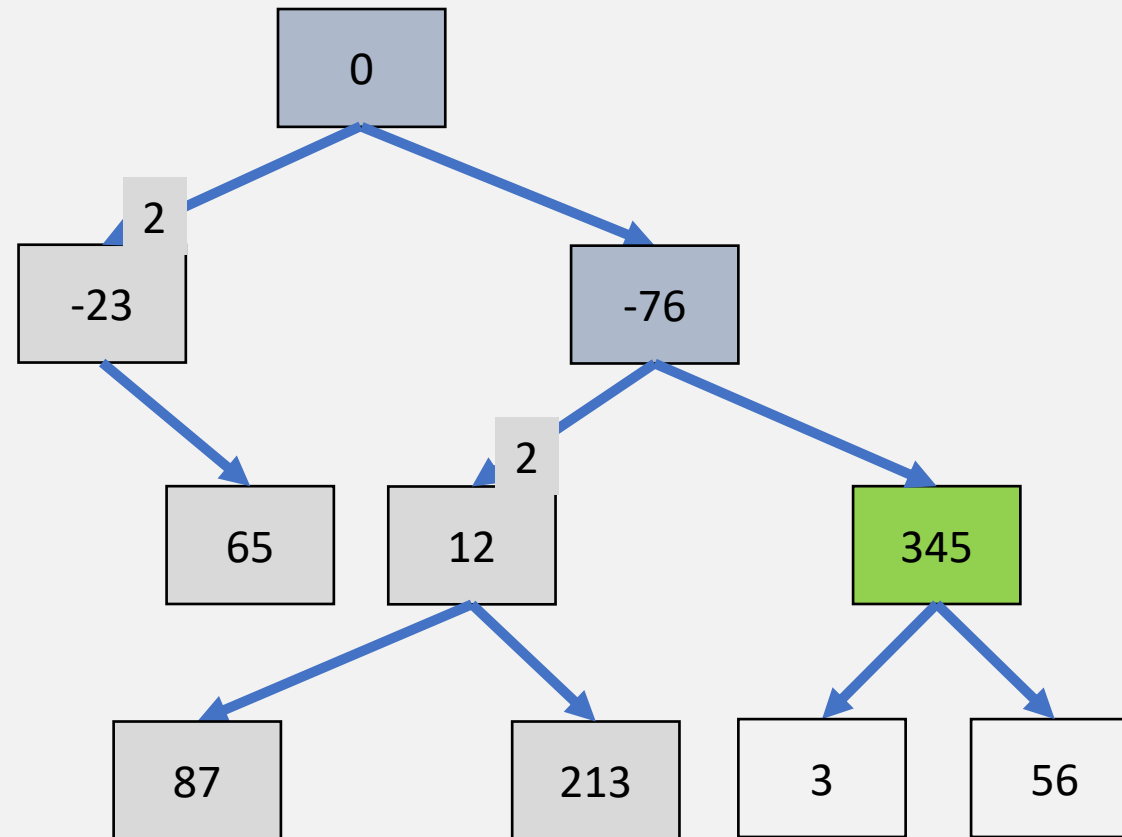
# Arbre binaire – Hauteur (ou profondeur)



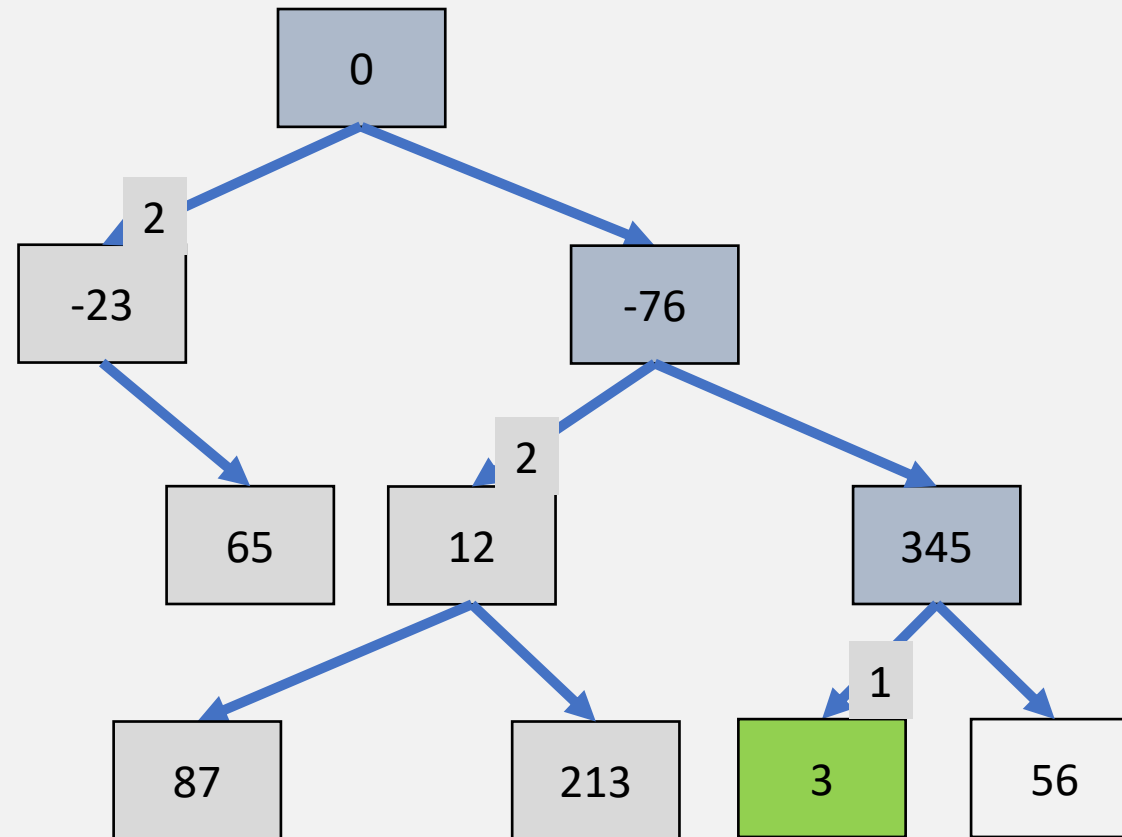
# Arbre binaire – Hauteur (ou profondeur)



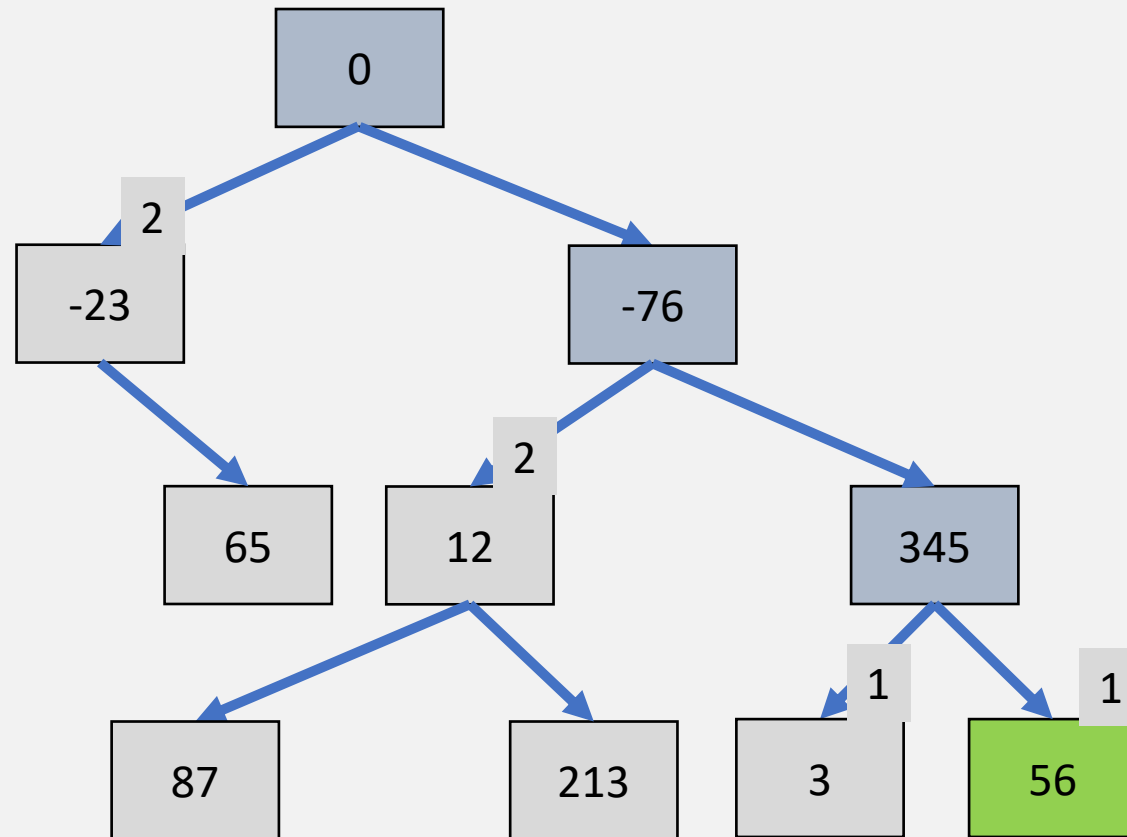
# Arbre binaire – Hauteur (ou profondeur)



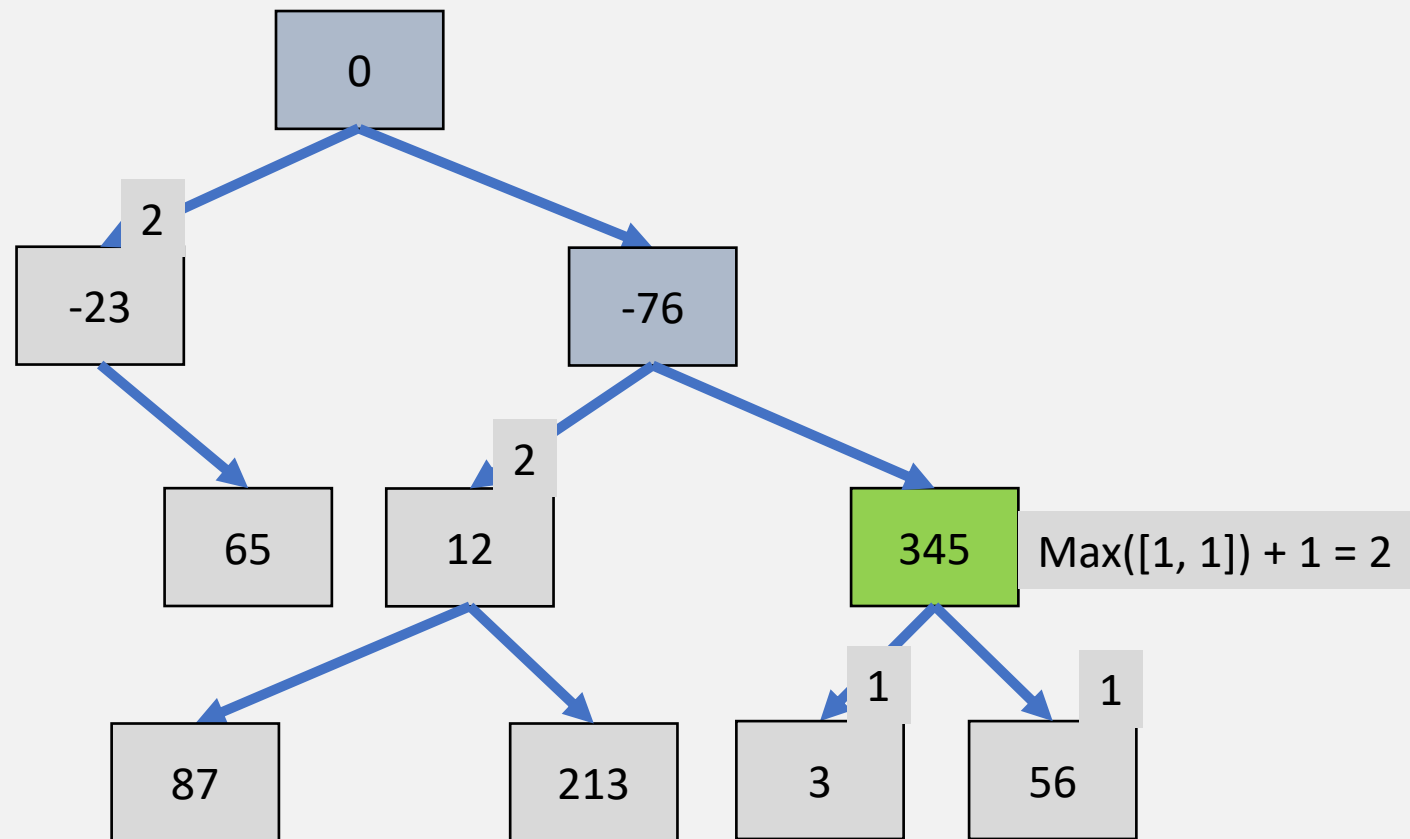
# Arbre binaire – Hauteur (ou profondeur)



# Arbre binaire – Hauteur (ou profondeur)

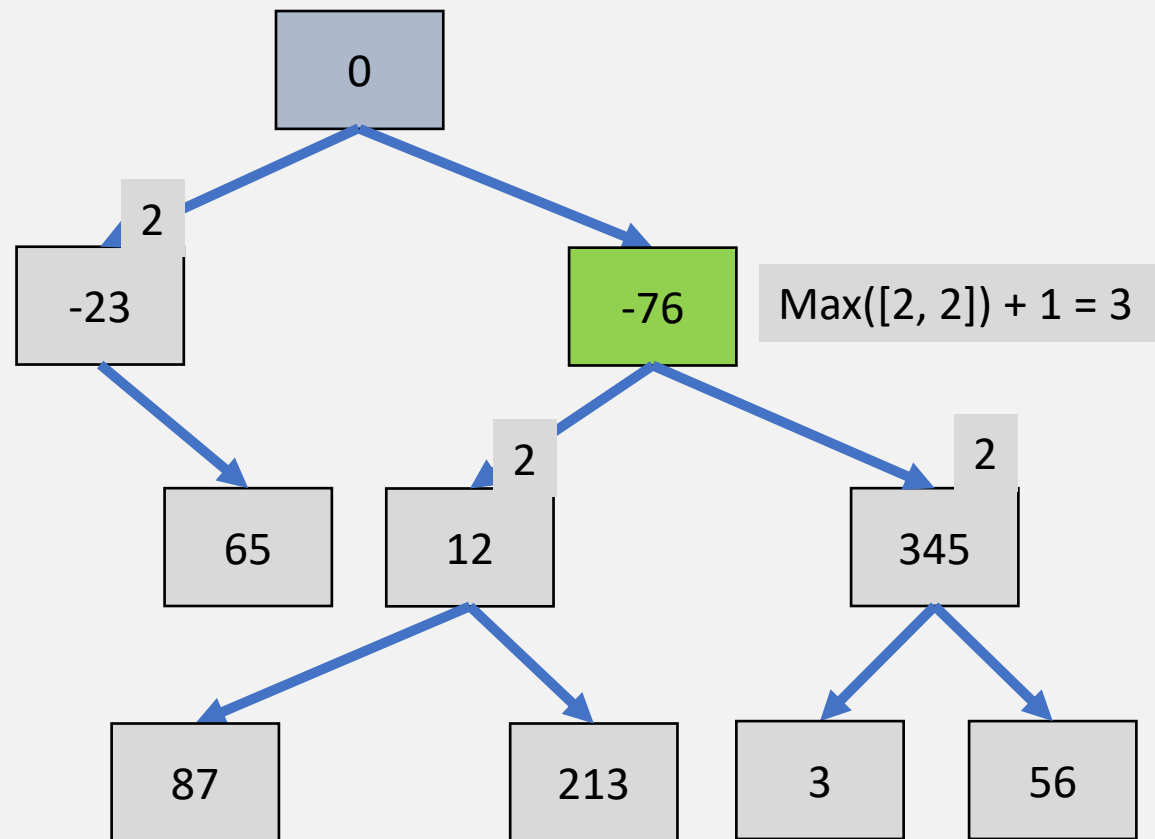


# Arbre binaire – Hauteur (ou profondeur)

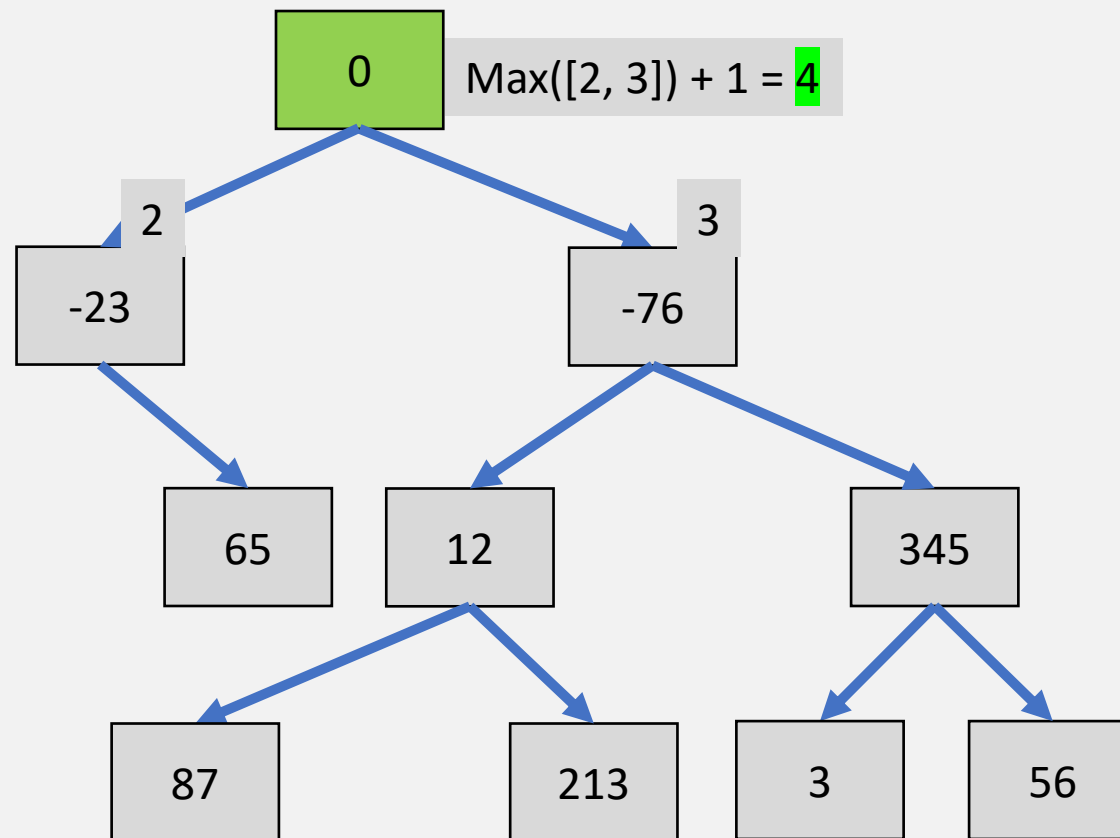




# Arbre binaire – Hauteur (ou profondeur)



# Arbre binaire – Hauteur (ou profondeur)



## Exercice 3 – Hauteur

- Dans la classe « ArbreBinaire », écrivez la méthode « get » de la propriété « Hauteur » qui calcule sa hauteur et la renvoie

# Exercice 3 – Hauteur – Solution

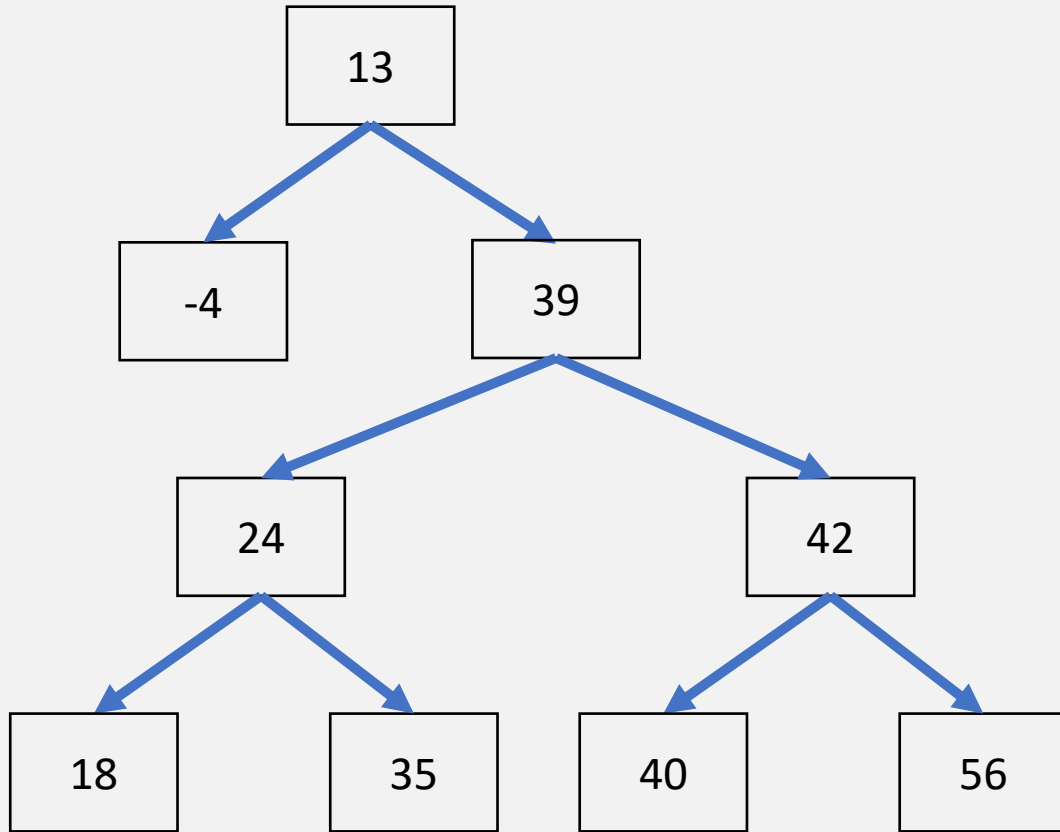
```
entier Hauteur(Noeud p_noeud)
{
    SI (p_noeud == null) ALORS
    {
        renvoyer 0;
    }

    renvoyer 1 + Max(Hauteur (p_noeud.Gauche), Hauteur(p_noeud.Droit));
}
```

# Parcours généralisé sur les arbres binaires

```
aucun Parcours(Noeud p_noeud) {  
    SI (p_noeud != null) ALORS {  
        Traitement_prefixe(p_noeud.Valeur)  
  
        Parcours(p_noeud.Gauche)  
  
        Traitement_infixe(p_noeud.Valeur)  
  
        Parcours(p_noeud.Droit)  
  
        Traitement_postfixe(p_noeud.Valeur)  
    }  
}
```

# ABR – Exemples de parcours



Si le traitement choisi pour un des parcours est un affichage voici les résultats des différents parcours

Parcours préfixe : 13, -4, 39, 24, 18, 35, 42, 40, 56

Parcours infixe : -4, 13, 18, 24, 35, 39, 40, 42, 56

Parcours suffixe : -4, 18, 35, 24, 40, 56, 42, 39, 13

# Exercice 4 – Parcours

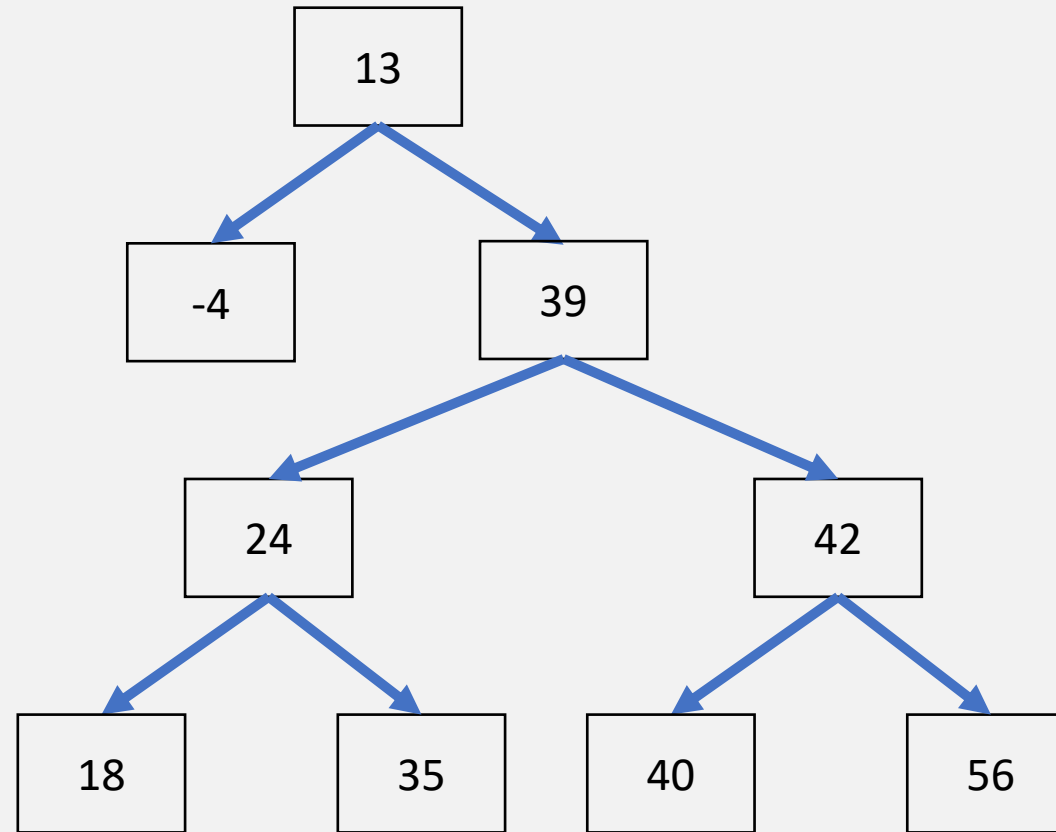
- À partir de l'algorithme précédent, dans la classe « ArbreBinaire », écrivez les méthodes suivantes :
  - ParcoursPrefixe : prend un traitement en paramètre et l'applique sur le nœud courant **avant** de parcourir ses fils
  - ParcoursInfixe : prend un traitement en paramètre et **parcours** son **fils gauche** puis **applique** le traitement sur le nœud courant et **parcours** son **fils droit**
  - ParcoursPostfixe : prend un traitement en paramètre et l'applique sur le nœud courant **après** de parcourir ses fils
- Le traitement est une fonction qui prend la valeur en paramètres et ne renvoie rien
- **Sans exécuter le code**, cherchez sur papier le résultat ces parcours sur l'exemple de l'arbre que vous avez codé
- Validez que vous avez le bon résultat avec votre programme

# Arbre binaire de recherche (ABR) - Définition

- Un arbre binaire de recherche est un arbre binaire dont tous les nœuds respectent les propriétés suivantes pour tous les descendants (enfants, petits enfants, etc.) :
  - Les descendants qui se situent à gauche d'un nœud sont plus petit ou égal au nœud parent
  - Les descendants qui se situent à droite d'un nœud sont plus grand au nœud parent



# ABR - Example



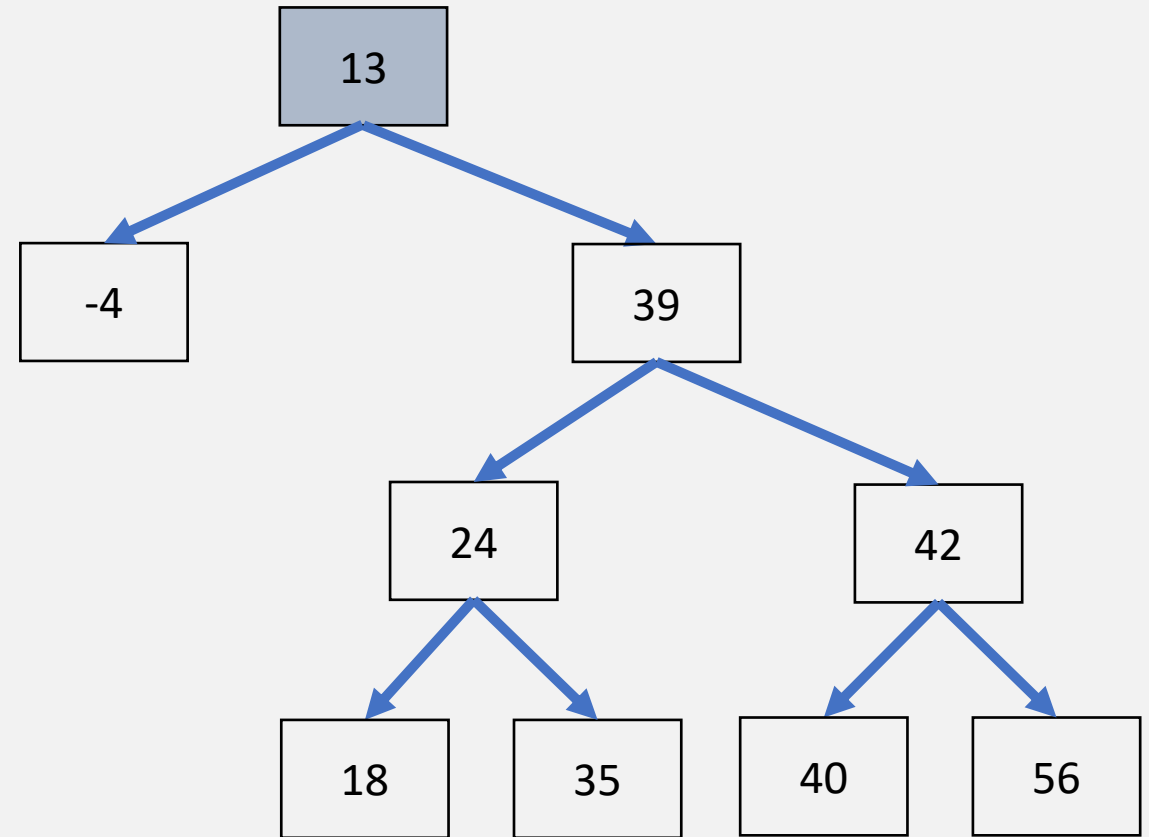
# Exercice 5 – ABR

- Proposez une structure de données générique qui permet de représenter un ABR et codez la
  - Le type d'élément doit implanter l'interface `IComparable<TypeElement>`
- Écrivez la méthode « Minimum » qui renvoie la plus petite valeur
- Écrivez la méthode « Maximum » qui renvoie la plus grande valeur
- Quelles sont les complexités de ces deux méthodes dans le meilleur des cas ?
- Quel type de parcours permet d'afficher les valeurs dans l'ordre croissant ?
- Dans la classe « `GenerateurArbreBinaire` », écrivez la méthode statique « `ExempleArbre2` » qui renvoie un arbre binaire d'entiers qui est exactement celui présenté en exemple

# ABR – Insertion

- Insertion du nombre 0

Nous partons de la racine

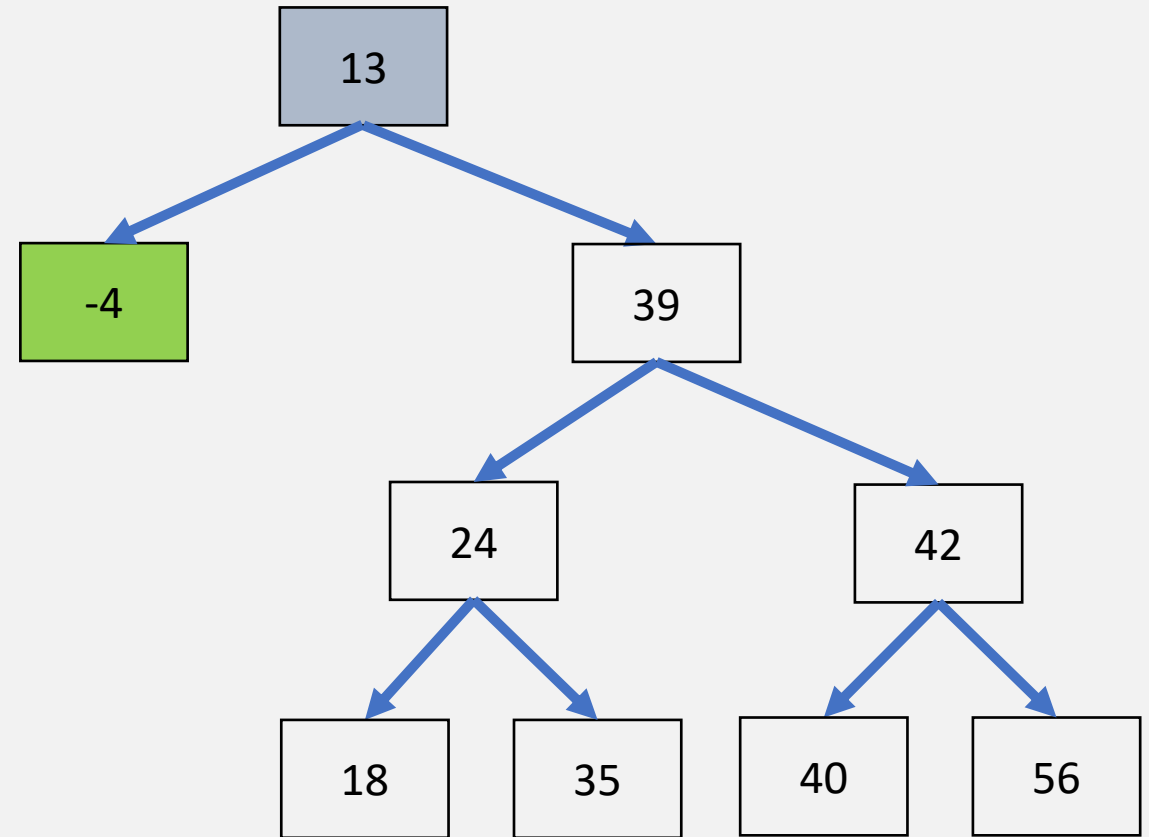


# ABR – Insertion

- Insertion du nombre 0

Nous partons de la racine

$0 < 13 \Rightarrow$  nous regardons l'enfant Gauche.

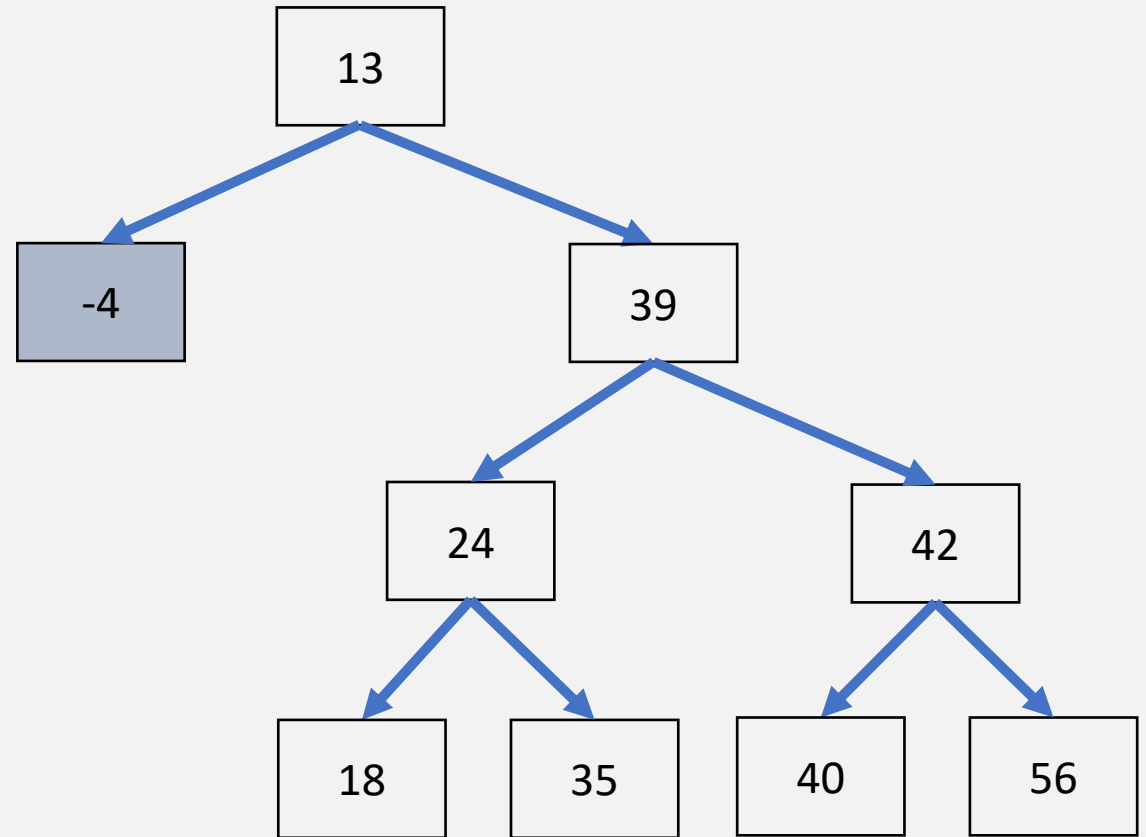


# ABR – Insertion

- Insertion du nombre 0

Nous partons de la racine

$-4 < 0 \Rightarrow$  nous regardons l'enfant Droit.

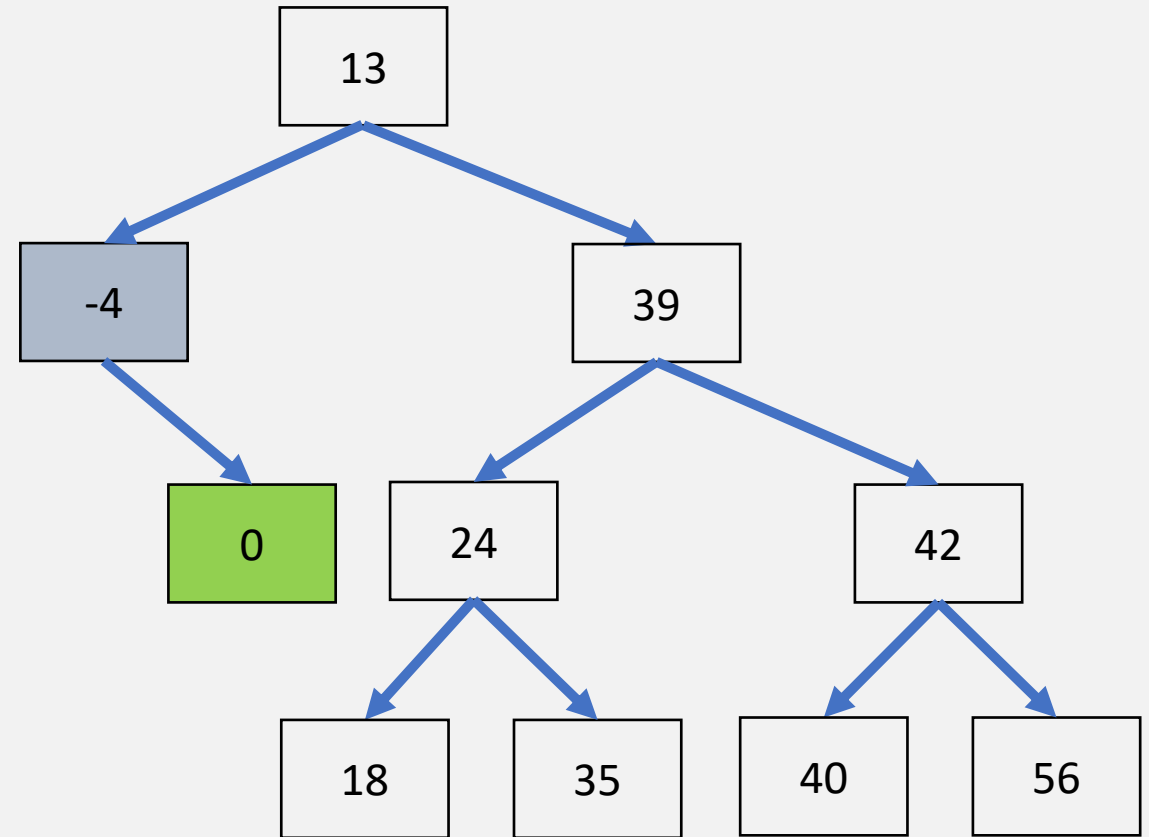


# ABR – Insertion

- Insertion du nombre 0

Nous partons de la racine

L'enfant droit n'existe alors nous allons le créer et l'associer à l'enfant Droit de -4



# Exercice 6 – Opérations

- Écrivez et codez l'algorithme d'insertion d'une valeur dans l'arbre
- Quel problème voyez vous aux ABR ? (Pour vous aider, sur papier, essayez d'insérer les éléments suivants dans l'ordre donné : -42, -10, 0, 15, 23, 42)
- Déduisez :
  - Quelle est la hauteur idéale d'un arbre binaire de recherche ?
  - Quelle est la hauteur dans le pire des cas de l'arbre binaire de recherche ?
- Déterminer sur papier comment rechercher une valeur
- Écrivez et codez l'algorithme