

Arbre binaire de recherche

Christophe Viroulaud

Terminale - NSI

Algo 09

La recherche d'un élément dans un tableau est **linéaire** dans le pire des cas : elle dépend de la taille du tableau.

32	14	98	1	67	19	8
----	----	----	---	----	----	---

Comment obtenir une méthode de recherche efficace
avec les arbres ?

1. Arbre binaire de recherche

1.1 Définition

1.2 Hauteur

1.3 Insertion

1.4 Recherche

On impose une contrainte à chaque nœud d'un arbre binaire :

- ▶ les valeurs du sous-arbre gauche sont plus petites que celle du nœud,
- ▶ les valeurs du sous-arbre droit sont plus grandes que celle du nœud.

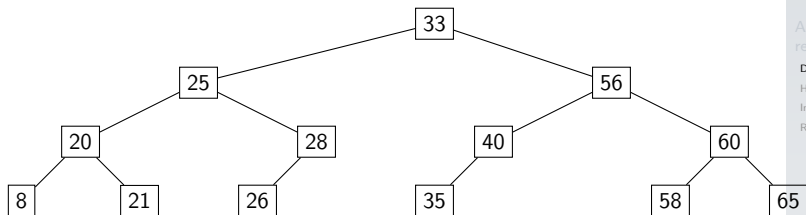


FIGURE 1 – Un Arbre Binaire de Recherche (ABR)

Remarque

On suppose que chaque valeur n'apparaît qu'une seule fois dans l'arbre.

Activité 1 :

1. Placer les valeurs 23, 27, 54, 55 dans l'ABR.
2. Où se trouve la plus grande valeur ? La plus petite ?
3. Effectuer un parcours infixe de l'arbre. Que remarque-t-on ?

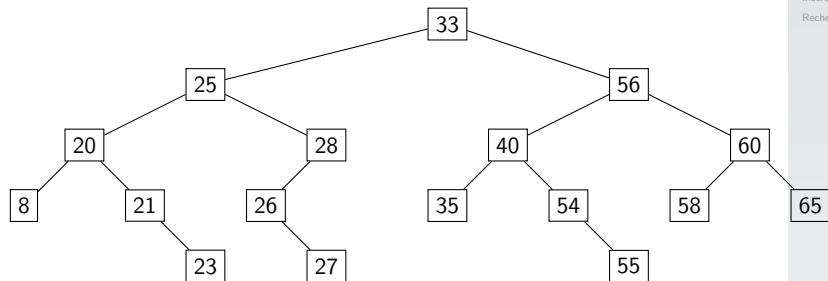


FIGURE 2 – Activité 1

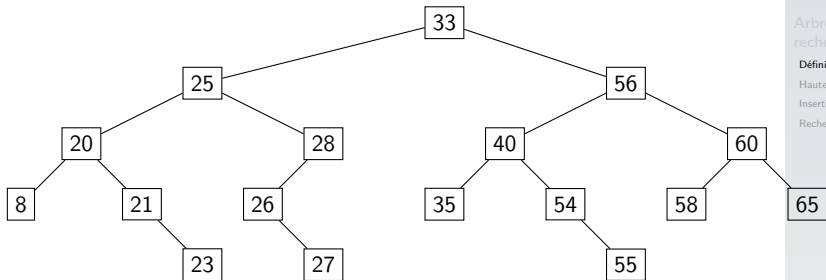
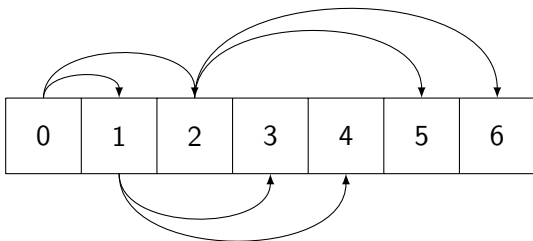


FIGURE 3 – Activité 1

- ▶ La plus petite valeur est dans le nœud le plus à gauche.
- ▶ La plus grande valeur est dans le nœud le plus à droite.
- ▶ parcours infixe : 8 - 20 - 21 - 23 - 25 - 26 - 27 - 28 - 33
- 35 - 40 - 54 - 55 - 56 - 58 - 60 - 65



Activité 2 : Il est judicieux d'utiliser un tableau pour représenter un arbre binaire de recherche.

Construire par compréhension un tableau `arbre` rempli de cent 0. On considérera dans la suite, que la valeur 0 représente un nœud vide.

```
1 arbre = [0 for _ in range(100)]
```

1. Arbre binaire de recherche

1.1 Définition

1.2 Hauteur

1.3 Insertion

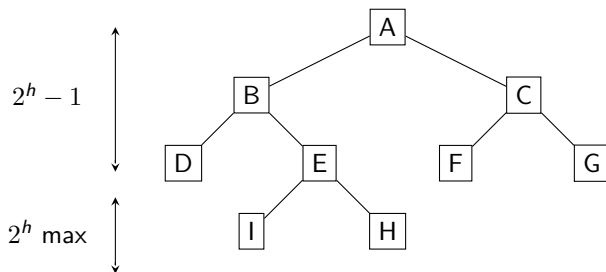
1.4 Recherche

$$h + 1 \leq n \leq 2^{h+1} - 1$$

Code 1 – Propriété des arbres binaires

$$2^h \leq n \leq 2^{h+1} - 1$$

Car il y a entre 1 et 2^h nœuds sur le dernier niveau.



$$2^h \leq n \leq 2^{h+1} - 1$$

$$\Rightarrow 2^h \leq n \leq 2^{h+1}$$

$$\Leftrightarrow h \leq \log_2(n) \leq h + 1$$

À retenir

Dans un arbre binaire **équilibré** :

$$h = \log_2(n)$$

Hors programme

On peut rééquilibrer un arbre en effectuant des **rotations**.

1. Arbre binaire de recherche

1.1 Définition

1.2 Hauteur

1.3 Insertion

1.4 Recherche

Pour insérer un élément dans un arbre binaire de recherche :

- ▶ On part de la racine.
- ▶ On descend dans le sous-arbre gauche si l'élément est inférieur à la racine.
- ▶ On descend dans le sous-arbre droit si l'élément est supérieur à la racine.

On applique récursivement cet algorithme jusqu'à une feuille.

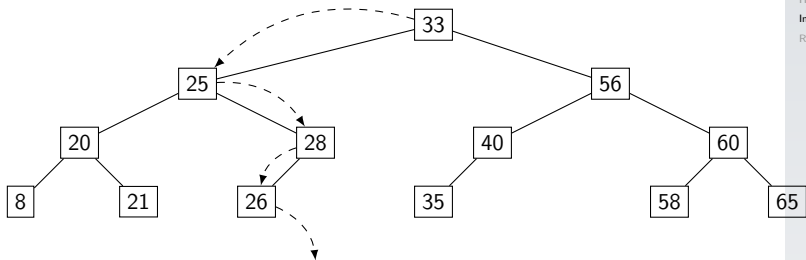


FIGURE 4 – Ajouter 27

Activité 3 :

1. Quelle est la complexité temporelle de cet algorithme ?
2. Écrire la fonction récursive `insérer(val: int, abr: list, i_pere: int) → None` qui insère `val` dans l'arbre de recherche `abr` représenté par un tableau.
3. Insérer dans l'ordre les valeurs : 33, 56, 25, 20, 28, 40, 21, 8, 26, 60, 35, 58, 65.

```
1 def inserer(val: int, abr: list, i_pere: int) -> None:
2     if abr[i_pere] == 0: # cellule vide: cas limite
3         abr[i_pere] = val
4     elif val < abr[i_pere]:
5         inserer(val, abr, 2*i_pere+1) # gauche
6     else:
7         inserer(val, abr, 2*i_pere+2) # droite
```

À retenir

L'insertion a une complexité logarithmique $O(\log_2(n))$.

On parcourt au maximum la hauteur de l'arbre.

```
1 inserer(33, arbre, 0)
2 inserer(56, arbre, 0)
3 inserer(25, arbre, 0)
4 ...
```

Remarque

Selon l'ordre d'ajout, l'arbre produit ne sera pas le même.

1. Arbre binaire de recherche

1.1 Définition

1.2 Hauteur

1.3 Insertion

1.4 Recherche

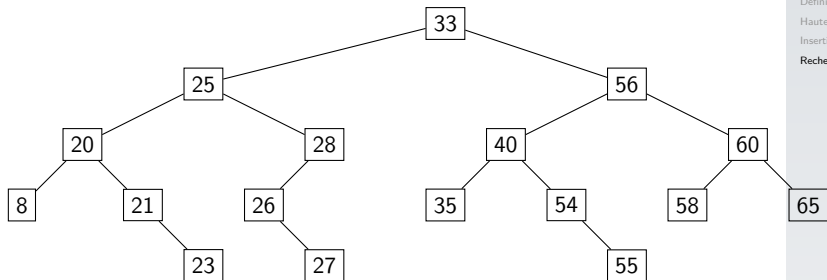
Pour rechercher un élément dans un arbre binaire de recherche :

- ▶ On part de la racine.
- ▶ On descend dans le sous-arbre gauche si l'élément est inférieur à la racine.
- ▶ On descend dans le sous-arbre droit si l'élément est supérieur à la racine.

On applique récursivement cet algorithme jusqu'à trouver l'élément ou bien arriver à une feuille.

1

[33, 25, 56, 20, 28, 40, 60, 8, 21, 26, 35, 58, 65]



Activité 4 :

1. Quelle la complexité temporelle dans le pire des cas de la recherche d'un élément dans le tableau ?
2. Que devient cette complexité pour un arbre binaire de recherche ?

- ▶ Dans un tableau la recherche a une complexité **linéaire**, $O(n)$.
- ▶ Dans un ABR la recherche a une complexité **logarithmique**, $O(\log_2(n))$ dans le pire des cas (c'est à dire quand on ne trouve pas l'élément).

Activité 5 : Écrire la fonction récursive
`rechercher(val: int, abr: list, i_pere: int)`
→ bool qui cherche si `val` est présent dans l'arbre
`abr`.

```
1 def rechercher(val: int, abr: list, i_pere: int) -> bool:
2     if abr[i_pere] == 0: # non trouvé
3         return False
4     elif abr[i_pere] == val: # trouvé
5         return True
6     elif val < abr[i_pere]: # gauche
7         return rechercher(val, abr, 2*i_pere+1)
8     else: # droit
9         return rechercher(val, abr, 2*i_pere+2)
```

- ▶ Considérons un arbre binaire de recherche qui contient 1 million d'éléments.

- ▶ Considérons un arbre binaire de recherche qui contient 1 million d'éléments.
- ▶ Sa hauteur est $h \simeq \log_2(1000000) \simeq 20$

- ▶ Considérons un arbre binaire de recherche qui contient 1 million d'éléments.
- ▶ Sa hauteur est $h \simeq \log_2(1000000) \simeq 20$
- ▶ Il faut seulement 20 étapes pour effectuer une recherche dans l'arbre, contre 1 million dans un tableau.