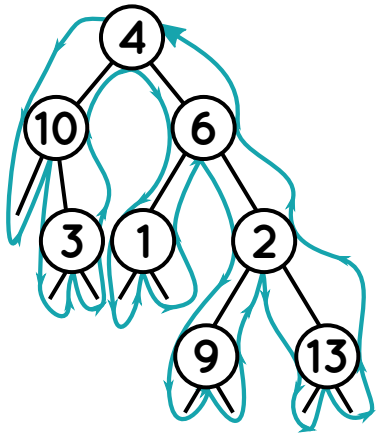
ALGORITHMIE

Parcours en profondeur



Parcours préfixe *[4, 10, 3, 6, 1, 2, 9, 13]*

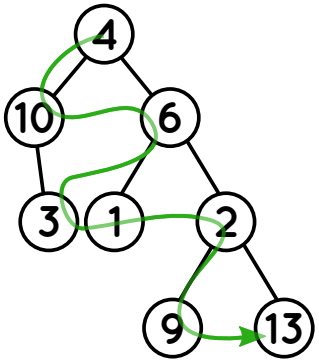
On note la valeur la 1ère rencontre

Parcours infixe *[10, 3, 4, 1, 6, 9, 2, 13]*

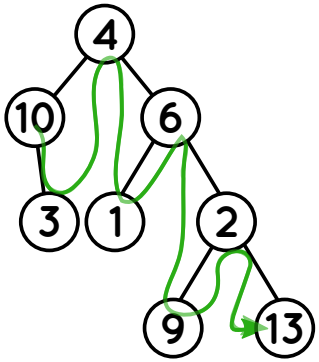
On note la valeur la 2ème rencontre

Parcours suffixe *[3, 10, 1, 9, 13, 2, 6, 4]*

On note la valeur la 3ème rencontre



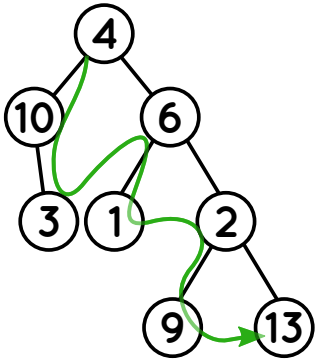
Parcours en largeur
[4, 10, 6, 3, 1, 2, 9, 13]



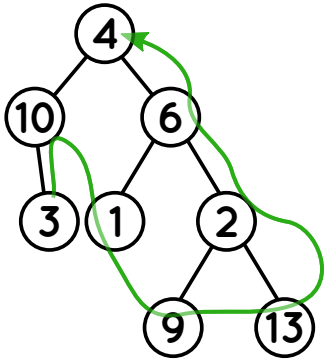
Parcours en profondeur infixe (2ème rencontre)
[10, 3, 4, 1, 6, 9, 2, 13]

Diviser pour régner consiste, pour résoudre un problème de taille N, à :
- **Diviser** : partager le problème en sous-problème (par exemple de taille N/2)
- **Régner** : résoudre ces différents sous-problèmes (généralement récursivement)
- **Combiner** : fusionner les solutions pour obtenir la solution du problème initial

On obtient en général moins d'appels récursifs avec la méthode diviser pour régner qu'avec une récursivité classique. Dans certains cas, la méthode diviser pour régner donne un algorithme de résolution plus rapide.



Parcours en profondeur préfixe (1ère rencontre)
[4, 10, 3, 6, 1, 2, 9, 13]



Parcours en profondeur suffixe (3ème rencontre)
[3, 10, 1, 9, 13, 2, 6, 4]

Implémentation orienté-objet d'un arbre binaire de recherche en python

```python
class ABR:
    def __init__(self, racine = None):
        self.racine = racine

    def est_vide(self):
        return self.racine is None

    def inserer(self, valeur):
        if self.est_vide():
            self.racine = Noeud(valeur)
        else:
            self.racine.inserer(valeur)


class Noeud:
    def __init__(self, valeur, noeud_gauche = None, noeud_droit = None):
        self.valeur = valeur
        self.noeud_gauche = noeud_gauche
        self.noeud_droit = noeud_droit

    def inserer(self, valeur):
        if valeur < self.valeur:
            if self.noeud_gauche is None:
                self.noeud_gauche = Noeud(valeur)
            else:
                self.noeud_gauche.inserer(valeur)
        elif valeur > self.valeur:
            if self.noeud_droit is None:
                self.noeud_droit = Noeud(valeur)
            else:
                self.noeud_droit.inserer(valeur)
```

```python
def rechercher(self, valeur):
    if self.est_vide():
        trouve = False
    else:
        trouve = self.racine.rechercher(valeur)
    return trouve
```

```python
def rechercher(self, valeur):
    if self.valeur == valeur:
        return True
    elif valeur < self.valeur:
        if self.noeud_gauche is None:
            return False
        else:
            return self.noeud_gauche.rechercher(valeur)
    else:
        if self.noeud_droit is None:
            return False
        else:
            return self.noeud_droit.rechercher(valeur)
```

Adrien Taudière