Algorithme 1
Structures de données

Hajer Akid & Sandrine Julia

Semestre 4

Tableaux

- Les éléments d'un tableau sont contigus dans l'espace mémoire. Avec l'indice, on sait donc à combien de cases mémoire se trouve l'élément en partant du début du tableau
- Le temps d'accès à un élément par son indice est constant, quel que soit l'élément désiré

Complexité

	Accès au n^{e}	Recherche	Insertion	Suppression
Tableau	$O(1)$	$O(n)$	$O(1)$	$O(n)$
Tableau trié	$O(1)$	$O(\log(n))$	$O(n)$	$O(n)$

Liste

- Une **liste chaînée** désigne une structure de données représentant une collection ordonnée et de taille arbitraire d'éléments
- L'accès aux éléments d'une liste se fait de manière séquentielle
  - chaque élément permet l'accès au suivant (contrairement au cas du tableau dans lequel l'accès se fait de manière absolue, par adressage direct de chaque cellule dudit tableau)
- **Un élément contient un accès vers une donnée**

Liste

Le principe de la liste chaînée est que chaque élément possède, en plus de la donnée, des pointeurs vers les éléments qui lui sont logiquement adjacents dans la liste

Opérations/syntaxe

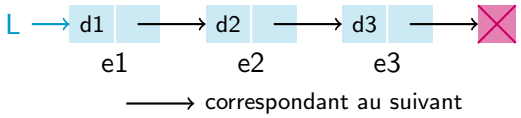
- premier(L) : désigne le premier élément de la liste
- nil : désigne l'absence d'élément

Liste simplement chaînée

- donnée(elt) : désigne la donnée associée à l'élément elt
- suivant(elt) : désigne l'élément suivant elt

Liste simplement chaînée

Représentation



- premier(L) = e1
- donnée(e1) = d1, suivant(e1) = e2
- donnée(e2) = d2, suivant(e2) = e3
- donnée(e3) = d3, suivant(e3) = nil

Complexité

	Accès au n^{e}	Recherche	Insertion	Suppression
Tableau	$O(1)$	$O(n)$	$O(1)$	$O(n)$
Tableau trié	$O(1)$	$O(\log(n))$	$O(n)$	$O(n)$
Liste chaînée	$O(n)$	$O(n)$	$O(1)$	$O(n)$

Pile

- Une pile (en anglais *stack*) est un **type abstrait** fondé sur le principe "dernier arrivé, premier sorti" (ou LIFO pour *Last In, First Out*)
- Les derniers éléments ajoutés à la pile seront les premiers à être récupérés

Exemple

- Pile d'assiettes : on ajoute des assiettes sur la pile, et on les récupère dans l'ordre inverse, en commençant par la dernière ajoutée
- Pile de crêpes

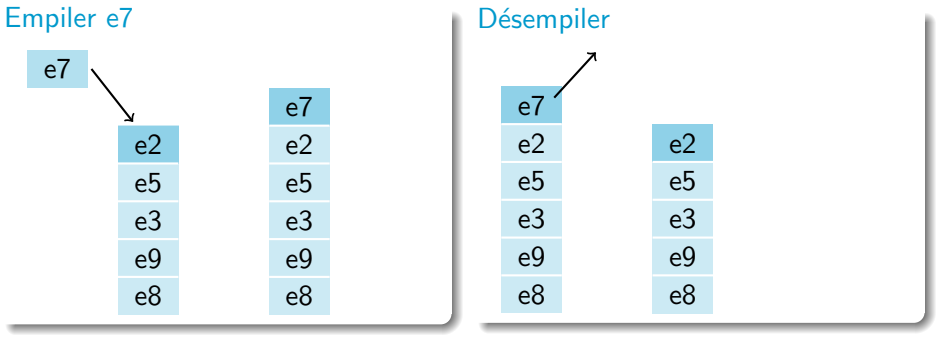
Pile

Opérations

- Sommet(P) : renvoie le dernier élément ajouté et non encore retiré : le sommet (*top*)
- Empiler(P, elt) : comme insérer, place l'élément au sommet de la pile P (*push*)
- Désempiler(P) : comme supprimer, retire de la pile le sommet (*pop*)
- estVide(P) : renvoie vrai si la pile est vide et faux sinon (*empty*)

Pile

Représentation



Pile

Implémentation par un tableau

Une structure composée

- un tableau (T)
- taille courante (s)

Opérations

- Créer(P, n) : créer P.T de taille n; P.s $\leftarrow 0$
- Sommet(P) : retourner P.T[P.s]
- Empiler(P, elt) : P.s $\leftarrow$ P.s $+ 1$; P.T[P.s] $\leftarrow$ elt
- Désempiler(P) : P.s $\leftarrow$ P.s -1
- estVide(P) : retourner P.s $= 0$

Attention

- Empiler(P, elt) : stack overflow = dépassement de la taille de T
- Désempiler(P) : P.s ne doit pas devenir négatif

Complexité

	Accès au n^e	Recherche	Insertion	Suppression
Tableau	$O(1)$	$O(n)$	$O(1)$	$O(n)$
Tableau trié	$O(1)$	$O(\log(n))$	$O(n)$	$O(n)$
Liste chaînée	$O(n)$	$O(n)$	$O(1)$	$O(n)$
Pile (Tableau)	$O(n)$	$O(n)$	$O(1)$	$O(1)$

File

- Une **file** (en anglais *queue*) est un **type abstrait** basé sur le principe "premier arrivé, premier sorti", en anglais FIFO (*First In, First Out*),
- Les premiers éléments ajoutés à la file seront les premiers à être récupérés

Exemple

- Une file d'attente : les premières personnes à arriver sont les premières personnes à sortir de la file

File

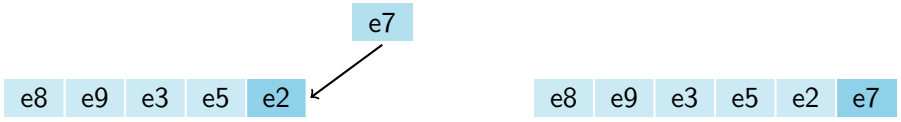
Opérations

- Début(F) : renvoie le premier élément ajouté et non encore retiré : le début ou le premier (*front*)
- Enfiler (F, elt) : comme insérer, place l'élément à la fin de la file F (*enqueue*)
- Dé filer (F) : comme supprimer, retire de la file le premier (*dequeue*)
- estVide(F) : renvoie vrai si la file est vide et faux sinon (*empty*)

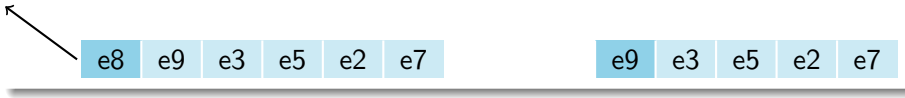
File

Représentation

Enfiler e7



Défiler



File

Liste simplement chaînée : opérations

- initListe (L)
- nombreElements(L)
- premier(L)
- ajouteEnTête(elt , L)
- ajouteEnFin(elt , L)
- supprime(elt , p, L)

Implémentation par une liste

Une liste L simplement chaînée

- Début(F) : renvoyer premier(L)
- Enfiler (F, elt) : ajouteEnFin(elt , L)
- Dé filer (F) : supprime(premier(L), nil , L)
- estVide(F) : renvoyer nombreElements(L) = 0

Complexité

	Accès au n^e	Recherche	Insertion	Suppression
Tableau	$O(1)$	$O(n)$	$O(1)$	$O(n)$
Tableau trié	$O(1)$	$O(\log(n))$	$O(n)$	$O(n)$
Liste chaînée	$O(n)$	$O(n)$	$O(1)$	$O(n)$
Pile (Tableau)	$O(n)$	$O(n)$	$O(1)$	$O(1)$
File (Liste)	$O(n)$	$O(n)$	$O(1)$	$O(1)$

File de priorité

- En informatique, une **File de priorité** est un type abstrait élémentaire qui manipule des éléments, chacun ayant une clé, sur laquelle on peut effectuer trois opérations :
  - insérer un élément
  - lire puis supprimer l'élément ayant la plus grande clé
  - tester si la File de priorité est vide ou pas.
- On ajoute parfois à cette liste l'opération
  - augmenter la clé d'un élément

Tas binaire

1	2	3	4	5	6	7	8	9	10
9	8	3	6	7	2	0	1	4	5

- Un **tas ascendant** ou **tas-max** est un tableau dont les cases sont numérotées de 1 à n vérifiant :
  pour tout i entre 1 et (n div 2) :

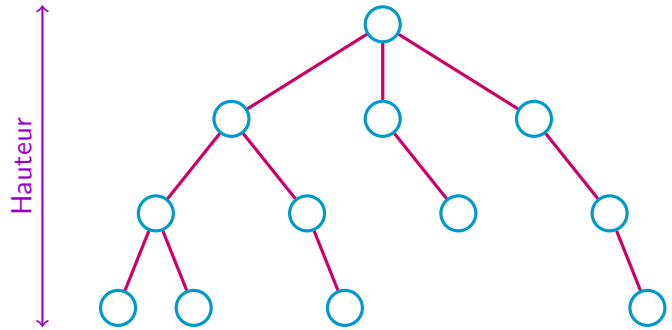
$$\text{si } 2i \leq n : \quad T[i] \geq T[2i]$$
$$\text{si } 2i+1 \leq n : \quad T[i] \geq T[2i+1]$$

Complexité

	Accès au n^e	Recherche	Insertion	Suppression
Tableau	$O(1)$	$O(n)$	$O(1)$	$O(n)$
Tableau trié	$O(1)$	$O(\log(n))$	$O(n)$	$O(n)$
Liste chaînée	$O(n)$	$O(n)$	$O(1)$	$O(n)$
Pile (Tableau)	$O(n)$	$O(n)$	$O(1)$	$O(1)$
File (Liste)	$O(n)$	$O(n)$	$O(1)$	$O(1)$
File de priorité (Tas)	$O(n\log(n))$	$O(n)$	$O(\log(n))$	$O(\log(n))$

Arbre

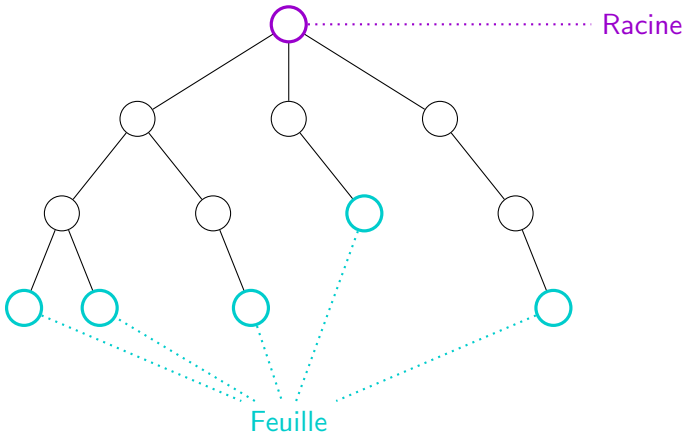
Un arbre est une structure de donnée très utilisée en informatique



Noeuds
Arêtes

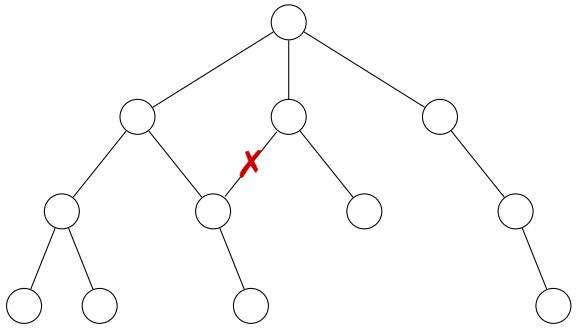
Arbre

Un arbre est une structure de donnée très utilisée en informatique



Arbre

Un arbre est une structure de donnée très utilisée en informatique



Un arbre ne possède pas de cycle

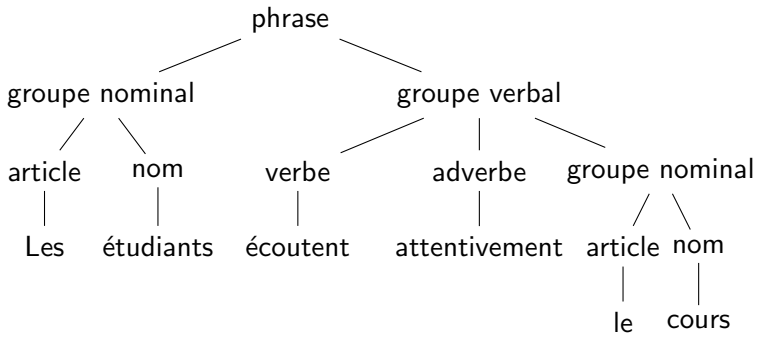
Arbres

Exemple

- Analyse syntaxique
- Expressions algébriques
- Dictionnaire
- Toutes organisations hiérarchiques
- ...

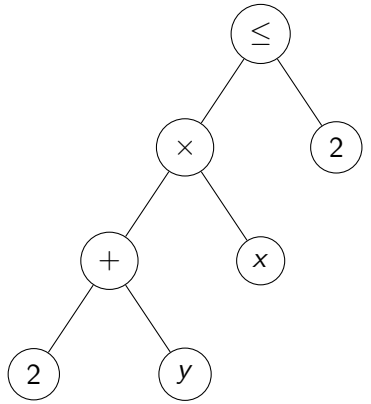
Analyse syntaxique

"Les étudiants écoutent attentivement le cours"



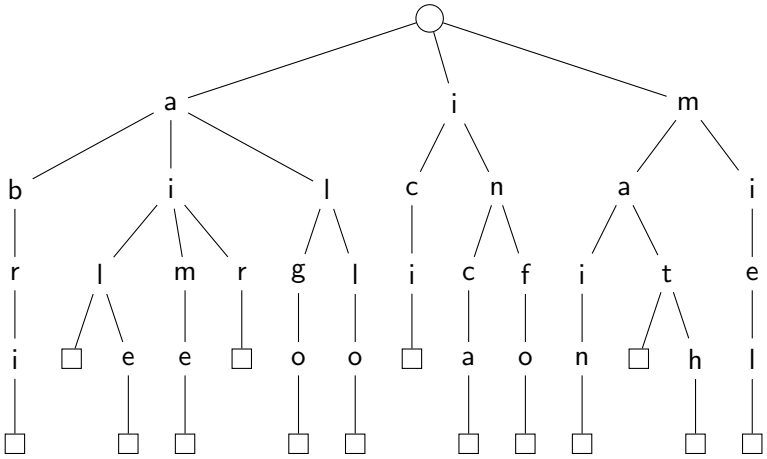
Expression arithmétique

$(2 + y) \times x \leq 2$



Dictionnaire

abri, ail, aile, aime, air, algo, allo, ici, inca, info, main, mat, math, miel



Arbres

Arbre binaire

Un arbre où chaque nœuds a au plus 2 fils

Parcours

Préfixe (RGD)

On traite la racine puis le sous-arbre gauche et finalement le sous-arbre droit

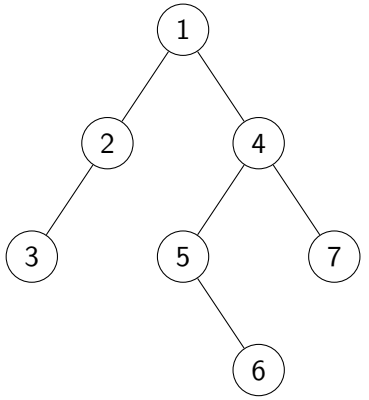
Infixe (GRD)

On traite le sous-arbre gauche puis la racine et finalement le sous-arbre droit

Postfixe (GDR)

On traite le sous-arbre gauche puis le sous-arbre droit et finalement la racine

Parcours



- Préfixe (RGD) : 1 2 3 4 5 6 7
- Infixe (GRD) : 3 2 1 5 6 4 7
- Postfixe (GDR) : 3 2 6 5 7 4 1

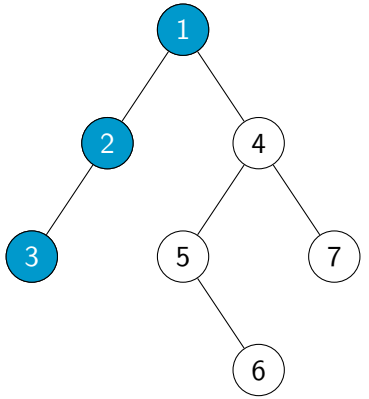
Recherche

En Profondeur (*Depth-First Search*)

Si l'élément recherché n'est pas à la racine

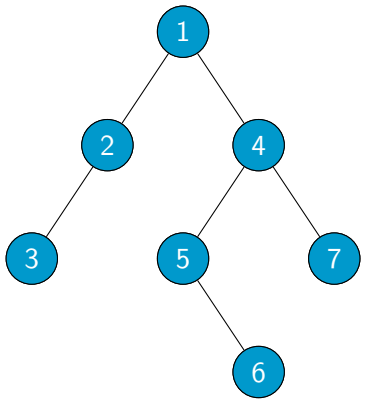
- On cherche dans le sous-arbre gauche
- Si on n'a rien trouvé on cherche dans le sous-arbre droit

Recherche



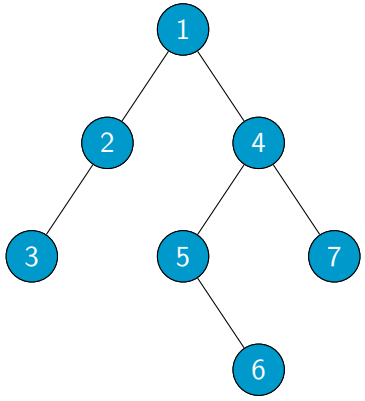
On cherche 3 **Trouvé !**

Recherche



On cherche 7 **Trouvé !**

Recherche



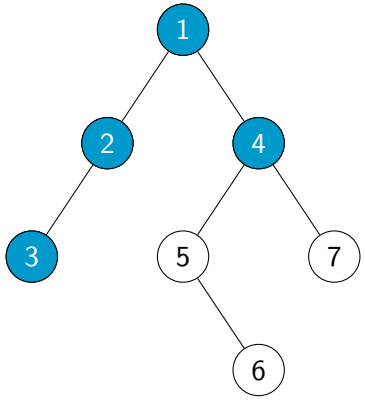
On cherche 10 **N'existe pas !**

Recherche

En Largeur (*Breadth-First Search*)

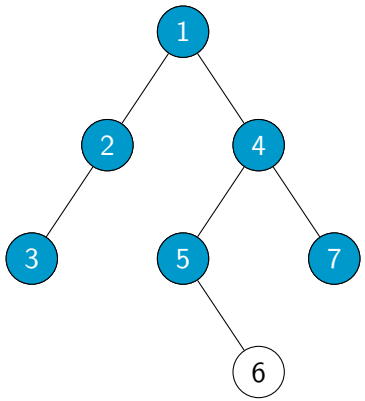
Si l'élément recherché n'est pas à la racine
On cherche niveau par niveau

Recherche



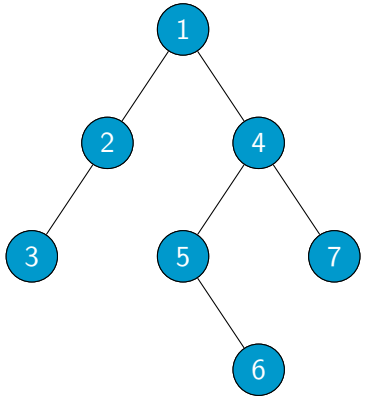
On cherche 3 **Trouvé !**

Recherche



On cherche 7 **Trouvé !**

Recherche



On cherche 10 **N'existe pas !**

BFS

```
BFS (A, x){
  si (A = nil) { retourner FAUX }
  sinon {
    F ← CréerFile()
    enfiler(F, A)
    tant que (non estVide(F)) {
      s ← Défiler(F)
      si (s.val = x) { retourner VRAI }
      si (s.g ≠ nil) { enfiler(F, s.g) }
      si (s.d ≠ nil) { enfiler(F, s.d) }
    }
    retourner FAUX
  }
}
```


Recherche

DFS

- Aucun stockage
- On peut s'attarder trop longtemps dans une branche

BFS

- Par niveau
- Besoin d'un stockage dont la taille augmente à chaque niveau

Complexité

	Accès au n^e	Recherche	Insertion	Suppression
Tableau	$O(1)$	$O(n)$	$O(1)$	$O(n)$
Tableau trié	$O(1)$	$O(\log(n))$	$O(n)$	$O(n)$
Liste chaînée	$O(n)$	$O(n)$	$O(1)$	$O(n)$
Pile (Tableau)	$O(n)$	$O(n)$	$O(1)$	$O(1)$
File (Liste)	$O(n)$	$O(n)$	$O(1)$	$O(1)$
File de priorité (Tas)	$O(n\log(n))$	$O(n)$	$O(\log(n))$	$O(\log(n))$
Arbre binaire	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Arbre binaire de recherche

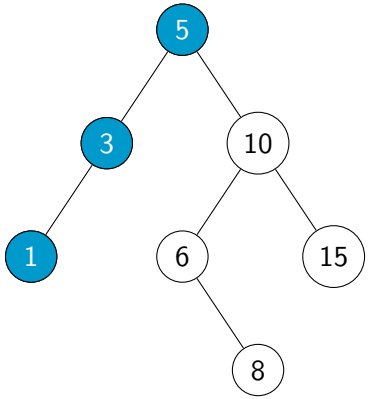
Chaque nœud possède une clé

- Chaque nœud du sous-arbre gauche possède une clé inférieure ou égale à celle du nœud considéré
- Chaque nœud du sous-arbre droit possède une clé supérieure ou égale à celle du nœud considéré

Opérations

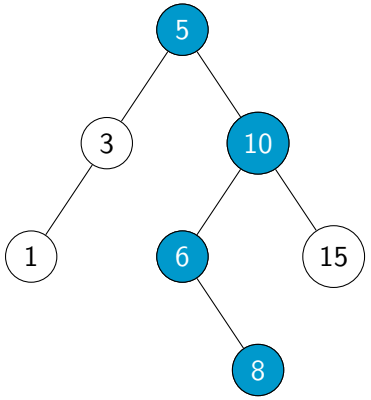
- Rechercher une valeur
- Ajouter un nœud
- Trouver le minimum/maximum
- Trouver le prédécesseur/successeur d'une valeur
- Supprimer un nœud

Recherche



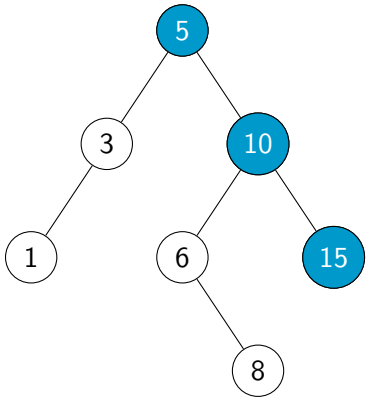
On cherche 1 **Trouvé !**

Recherche



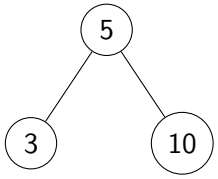
On cherche 8 **Trouvé !**

Recherche



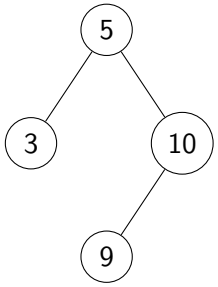
On cherche 12 **N'existe pas !**

Insertion



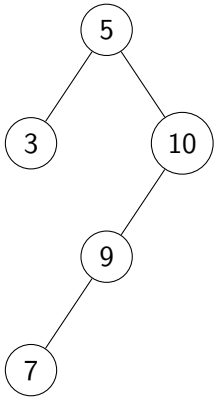
- On ajoute 9

Insertion



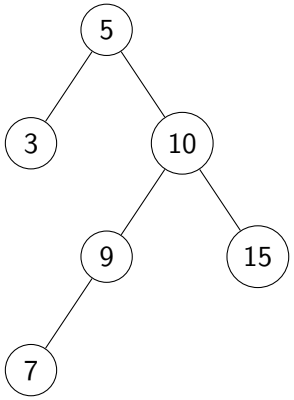
- On ajoute 9
- On ajoute 7

Insertion



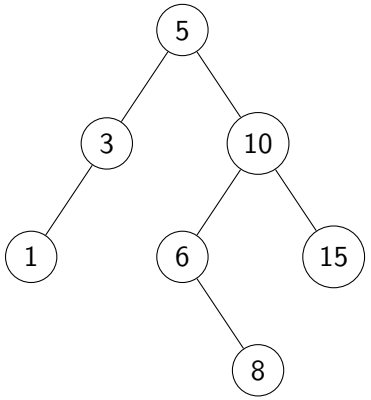
- On ajoute 9
- On ajoute 7
- On ajoute 15

Insertion



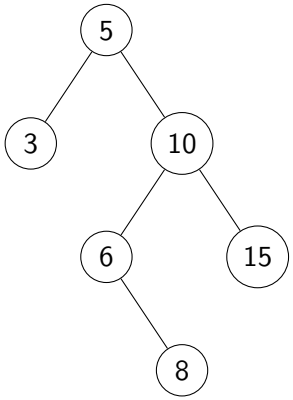
- On ajoute 9
- On ajoute 7
- On ajoute 15

Suppression



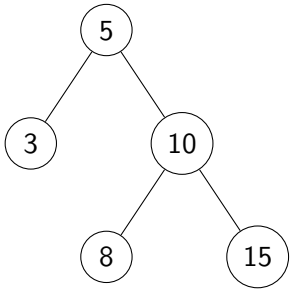
- On supprime 1

Suppression



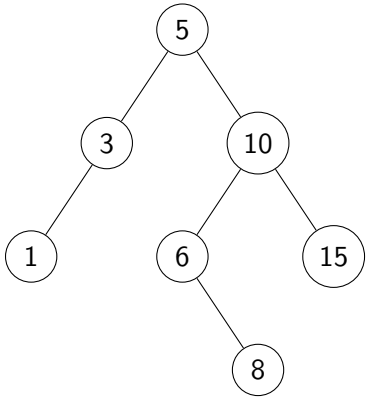
- On supprime 1
- On suppprime 6

Suppression



- On supprime 1
- On suppprime 6

Suppression



- On supprime 5 ?

Complexité

	Accès au n^{e}	Recherche	Insertion	Suppression
Tableau	$O(1)$	$O(n)$	$O(1)$	$O(n)$
Tableau trié	$O(1)$	$O(\log(n))$	$O(n)$	$O(n)$
Liste chaînée	$O(n)$	$O(n)$	$O(1)$	$O(n)$
Pile (Tableau)	$O(n)$	$O(n)$	$O(1)$	$O(1)$
File (Liste)	$O(n)$	$O(n)$	$O(1)$	$O(1)$
File de priorité (Tas)	$O(n\log(n))$	$O(n)$	$O(\log(n))$	$O(\log(n))$
Arbre binaire	$O(n)$	$O(n)$	$O(n)$	$O(n)$
ABR	$O(h)$	$O(h)$	$O(h)$	$O(h)$