Arbres binaires et ABR

À la fin de ce chapitre, je sais :

- ☞ définir un arbre binaire de manière inductive
- ☞ calculer la hauteur et la taille d'un arbre binaire en utilisant la définition inductive
- ☞ parcourir en profondeur un arbre binaire dans un ordre préfixe, infixe ou postfixe
- ☞ calculer la complexité d'un parcours en profondeur
- ☞ parcourir un arbre de recherche pour trouver un élément
- ☞ insérer et supprimer un élément dans un arbre de recherche
- ☞ donner les complexités associées aux opérations sur un arbre binaire de recherche

■ **Définition 1 — Type immuable.** Type de données que l'on ne peut pas modifier en mémoire.

Les listes chaînées et les arbre constituent les deux types **immuables** au programme de l'option informatique. À la différence d'une liste chaînée qui est une structure ordonnée séquentielle, un arbre est une **structure de données hiérarchique**. Elle permet d'implémenter différents types abstraits : des dictionnaires, des tas binaires et des files.

A Des arbres

■ **Définition 2 — Arbre.** Un arbre est un graphe connexe, acyclique et enraciné.

(R) La racine d'un arbre $\mathcal{A}$ est un sommet r particulier que l'on distingue : le couple (A, r) est un nommé arbre enraciné. On le représente un tel arbre verticalement avec la racine placée tout en haut comme sur la figure 1. Dans le cas d'un graphe orienté, la représentation verticale permet d'omettre les flèches.

(R) On confondra par la suite les arbres enracinés et les arbres.

■ **Définition 3 — Nœuds.** Les nœuds d'un arbre sont les sommets du graphe associé. Un nœud qui n'a pas de fils est une **feuille** (ou nœud externe). S'il possède des descendants, on parle alors de **nœud interne**.

■ **Définition 4 — Descendants, père et fils.** Si une arête mène du nœud i au nœud j, on dit que i est le **père** de j et que j est le **fils** de i. On représente l'arbre de telle sorte que le père soit toujours au-dessus de ses fils.

■ **Définition 5 — Arité d'un nœud.** L'arité d'un nœud est le nombre de ses fils.

■ **Définition 6 — Feuille.** Un nœud d'arité nulle est appelé une feuille.

■ **Définition 7 — Profondeur d'un nœud.** La profondeur d'un nœud est le nombre d'arêtes qui le sépare de la racine.

■ **Définition 8 — Hauteur d'un arbre.** La hauteur d'un arbre est la plus grande profondeur d'une feuille de l'arbre.

■ **Définition 9 — Taille d'un arbre.** La taille d'un arbre est le nombre de ses nœuds.

(R) Attention, la taille d'un graphe est le nombre de ses arêtes... Un arbre possède toujours $n-1$ arêtes si sa taille est n.

■ **Définition 10 — Sous-arbre.** Chaque nœud d'un arbre $\mathcal{A}$ est la racine d'un arbre constitué de lui-même et de ses descendants : cette structure est appelée sous-arbre de l'arbre $\mathcal{A}$.

(R) La notion de sous-arbre montre qu'un arbre est une structure intrinsèquement récursive ce qui sera largement utilisé par la suite !

■ **Définition 11 — Arbre recouvrant.** Un arbre recouvrant d'un graphe G est un sous-graphe couvrant de G qui est un arbre.

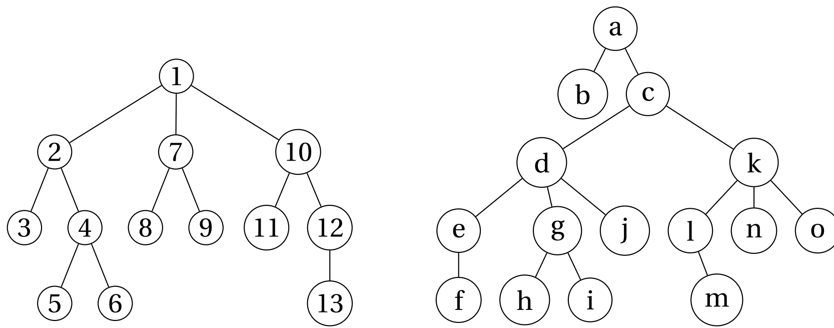


FIGURE 1 – Exemples d'arbres enracinés.

B Arbres binaires

On peut facilement transformer un arbre n-aire en un arbre binaire, beaucoup plus facile à coder. C'est pourquoi on s'intéresse tout particulièrement aux arbres binaires.

■ **Définition 12 — Arbre binaire.** Un arbre binaire est un arbre tel que tous les nœuds ont une arité inférieure ou égale à deux : chaque nœud possède au plus deux fils.

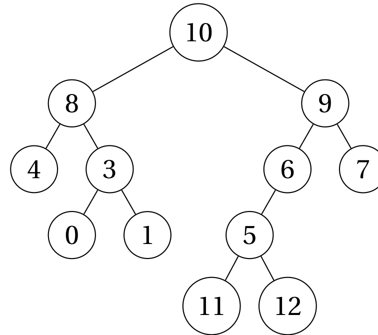


FIGURE 2 – Arbre binaire

■ **Définition 13 — Arbre binaire strict.** Un arbre binaire strict est un arbre dont tous les nœuds possèdent zéro ou deux fils.

■ **Définition 14 — Arbre binaire parfait.** Un arbre binaire parfait est un arbre dans lequel tous les niveaux sauf le dernier doivent être totalement remplis. Si le dernier n'est pas rempli totalement alors il doit être rempli de gauche à droite.

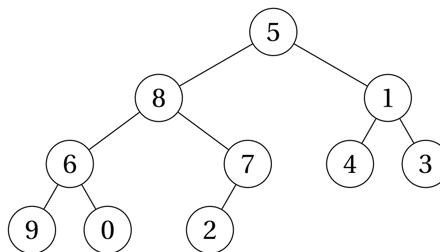


FIGURE 3 – Arbre binaire parfait

■ **Définition 15 — Arbre binaire équilibré.** Un arbre binaire est équilibré si sa hauteur est minimale, c'est à dire $h(a) = O(\log|a|)$.

(R) Un arbre parfait est un arbre équilibré.

Théorème 1 — La hauteur d'un arbre parfait de taille n vaut $\lfloor \log n \rfloor$. Soit a un arbre binaire parfait de taille n. Alors on a :

$$h(a) = \lfloor \log n \rfloor \tag{1}$$

Démonstration. Soit a un arbre binaire parfait de taille n. Comme a est parfait, on sait que tous les niveaux sauf le dernier sont remplis. Ainsi, il existe deux niveaux de profondeur $h(a)-1$ et $h(a)$. On peut encadrer le nombre de nœuds de a en remarquant que chaque niveau k possède 2^k nœuds, sauf le dernier. On a donc :

$$1+2+\ldots+2^{h(a)-1} < |a| \leqslant 1+2+\ldots+2^{h(a)} \tag{2}$$

$$\sum_{k=0}^{h(a)-1} 2^k < |a| \leqslant \sum_{k=0}^{h(a)} 2^k \tag{3}$$

$$2^{h(a)}-1 < |a| \leqslant 2^{h(a)+1}-1 \tag{4}$$

$$2^{h(a)} \leqslant |a| < 2^{h(a)+1} \tag{5}$$

On en conclut que $\lfloor \log_2 |a| \rfloor - 1 < h(a) \leqslant \lfloor \log_2 |a| \rfloor$ et donc que $h(a) = \lfloor \log_2(n) \rfloor$. ■

C Définition inductive des arbres binaires

La plupart des caractéristiques et des résultats importants liés aux arbres binaires peuvent se démontrer par induction structurelle. Cette méthode est une généralisation des démonstrations par récurrences sur $\mathbb{N}$ pour un ensemble défini par induction (cf. figure 4).

■ **Définition 16 — Étiquette d'un nœud.** Une étiquette d'un nœud est une information portée au niveau d'un nœud d'un arbre.

■ **Définition 17 — Définition inductive d'un arbre binaire.** Soit E un ensemble d'étiquettes. L'ensemble $\mathcal{A}_E$ des arbres binaires étiquetés par E est défini inductivement par :

1. VIDE est un arbre binaire appelé arbre vide (parfois noté $\emptyset$),
2. Si $e \in E$, $f_g \in \mathcal{A}$ et $f_d \in \mathcal{A}$ sont deux arbres binaires, alors $(f_g, e, f_d) \in \mathcal{A}_E$, c'est à dire que le triplet (f_g, e, f_d) est un arbre binaire étiqueté par E.

f_g et f_d sont respectivement appelés fils gauche et fils droit.

L'implémentation en OCaml donne :

```
type 'a btree = Vide | Noeud of 'a * 'a btree * 'a btree
```

C'est un type **polymorphe**, c'est-à-dire que les étiquettes sont de type générique `'a`. On peut donc créer des arbres dont les étiquettes sont des `int`, des `char` ou des `float` uniquement à partir de cette définition. Par exemple :

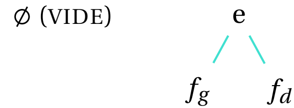


FIGURE 4 – Arbre binaire et définition inductive : arbre vide à gauche, arbre induit par e, f_g et f_d à droite

```
let  arbre = Noeud(3, Vide, Vide)
(* val arbre : int bintree = Noeud (3, Vide, Vide) *)
let  arbre = Noeud(3, Noeud(5, Noeud(7,Vide,Vide), Noeud(9, Vide,Vide)), Noeud(2, Noeud
    (8,Vide,Vide), Noeud(6, Vide,Vide)))
(* val arbre : int bintree =
Noeud (3, Noeud (5, Noeud (7, Vide, Vide), Noeud (9, Vide, Vide)),
Noeud (2, Noeud (8, Vide, Vide), Noeud (6, Vide, Vide))) *)
```

Ces arbres correspondent aux figure ci-dessous :

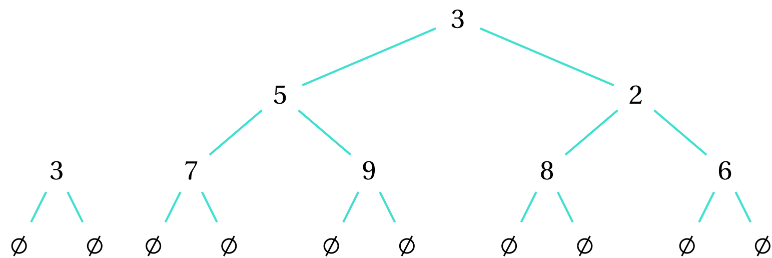


FIGURE 5 – Arbres binaires correspondant aux codes OCaml

■ **Définition 18 — Définition inductive des arbres binaires stricts non vides.** Soit E un ensemble d'étiquettes. L'ensemble $\mathcal{A}_E$ des arbres binaires étiquetés par E est défini inductivement par :

1. LEAF(e) est un arbre binaire appelé feuille qui porte une étiquette e,
2. Si $e \in E$, $f_g \in \mathcal{A}$ et $f_d \in \mathcal{A}$ sont deux arbres binaires, alors $(f_g, x, f_d) \in \mathcal{A}_E$, c'est à dire que le triplet (f_g, x, f_d) est un arbre binaire étiqueté par E.

L'étiquette de la feuille peut-être d'une nature différente de celle des nœuds internes.

En OCaml on peut définir ainsi un arbre binaire strict :

```
type ('a, 'b) sbtree = Feuille of 'a | Noeud of 'b * ('a, 'b) sbtree * ('a, 'b) sbtree
```

On peut ainsi définir les mêmes arbres que sur la figure 5 :

```
let  arbre = Feuille 3
(* val arbre : (int, 'a) sbtree = Feuille 3 *)
let  arbre = Noeud(3, Noeud(5, Feuille 7, Feuille 9), Noeud(2, Feuille 8, Feuille 6))
(* val arbre : (int, int) sbtree =
Noeud (3, Noeud (5, Feuille 7, Feuille 9), Noeud (2, Feuille 8, Feuille 6)) *)
```

On peut également définir des arbres plus complexes :

```
let  arbre = Noeud(3, Noeud(5, Feuille 'a', Feuille 'b'), Noeud(2, Feuille 'c', Feuille '
    d'))
(* val arbre : (char, int) sbtree =
Noeud (3, Noeud (5, Feuille 'a', Feuille 'b'), Noeud (2, Feuille 'c', Feuille 'd')) *)
```

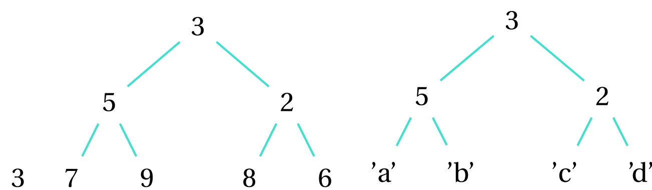


FIGURE 6 – Arbres binaires stricts non vides

(R) On utiliser dans ce qui suit des arbres binaires tels qu'ils sont définis en 17, c'est-à-dire des arbres binaires qui peuvent être vide. On peut naturellement extrapoler ces définitions dans le cas des arbres binaires stricts non vide.

D Démonstration par induction structurelle

■ **Définition 19 — Démonstration par induction structurelle sur un arbre binaire.** Soit $\mathcal{P}(a)$ un prédicat exprimant une propriété sur un arbre a de $\mathcal{A}_E$, l'ensemble des arbres binaires étiquetés sur un ensemble E. On souhaite démonter cette propriété.

La démonstration par induction structurelle procède comme suit :

1. (**CAS DE BASE**) Montrer que $\mathcal{P}(\text{VIDE})$ est vraie, c'est-à-dire que la propriété est vraie pour l'arbre vide,
2. (**PAS D'INDUCTION (Constructeur Noeud)**) Soit $e \in E$ une étiquette et $f_g \in \mathcal{A}$ et $f_d \in \mathcal{A}$ deux arbres binaires pour lesquels $\mathcal{P}(f_g)$ et $\mathcal{P}(f_d)$ sont vraies. Montrer que $\mathcal{P}\big((f_g, e, f_d)\big)$ est vraie.
3. (**CONCLUSION**) Conclure que quelque soit $a \in \mathcal{A}$, comme la propriété est vérifiée pour le cas de base et qu'elle est vérifiée pour le constructeur, $\mathcal{P}(a)$ est vraie.

E Définitions inductives de fonction sur les arbres

■ **Définition 20 — Définition inductive d'une fonction à valeur dans $\mathcal{A}_E$.** On définit une fonction ϕ de $\mathcal{A}_E$ à valeur dans un ensemble $\mathcal{Y}$ par :

1. la donnée de la valeur de $\phi(\text{VIDE})$,
2. en supposant connaître $e \in E$, $\phi(f_g)$ et $\phi(f_d)$ pour f_g et f_d dans $\mathcal{A}_E$, la définition de $\phi((f_g, e, f_d))$.

■ **Exemple 1 — Définition inductive de la hauteur d'un arbre.** Soit $a \in \mathcal{A}$ un arbre binaire. La hauteur $h(a)$ de a est donnée par :

1. $h(\text{VIDE}) = -1$,
2. $h\big((f_g, e, f_d)\big) = 1 + \max\big(h(f_g), h(f_d)\big)$.

R La figure 7 justifie le fait qu'un arbre vide est une hauteur égale à -1 : dans le cas d'une feuilles, on a alors $h = 1 - 1 = 0$. La figure 8 justifie le fait qu'une feuille étiquetée dans une arbre binaire strict non vide possède une hauteur de 0.

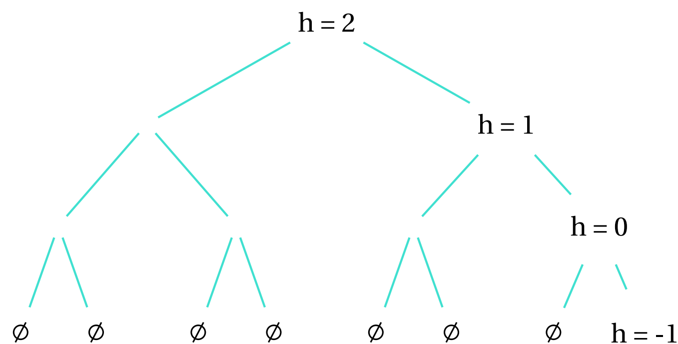


FIGURE 7 – La hauteur d'un arbre vide vaut -1 dans le cas d'un arbre binaire défini par 17

■ **Exemple 2 — Définition inductive de la taille d'un arbre.** Soit $a \in \mathcal{A}$ un arbre binaire. La taille $|a|$ de a est donnée par :

1. $|\text{VIDE}| = 0$,
2. $|(f_g, e, f_d)| = 1 + |f_g| + |f_d|$.

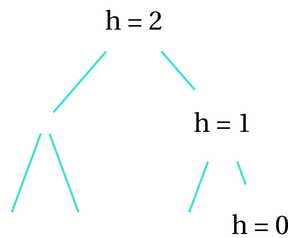


FIGURE 8 – La hauteur d'une feuille étiquetée vaut 0 dans le cas d'un arbre binaire strict non vide définie par 18

F Parcours en profondeur d'un arbre binaire

■ **Définition 21 — Parcours d'un arbre.** Le parcours d'un arbre est l'action de visiter une seule fois chaque nœud. L'intérêt d'un parcours est que l'on peut alors effectuer un calcul sur tous les nœuds de l'arbre : recherche d'une étiquette, compilation d'information ou modification de l'arbre.

■ **Définition 22 — Parcours en profondeur.** Un parcours en profondeur traite en priorité les enfants d'un nœud avant de traiter ses frères.

Les parcours en profondeur se programment naturellement récursivement et sont illustrés sur la figure 9. On distingue les parcours :

préfixe pour lequel l'étiquette du nœud en cours est traitée **avant** celles des deux sous-arbres gauche et droit,

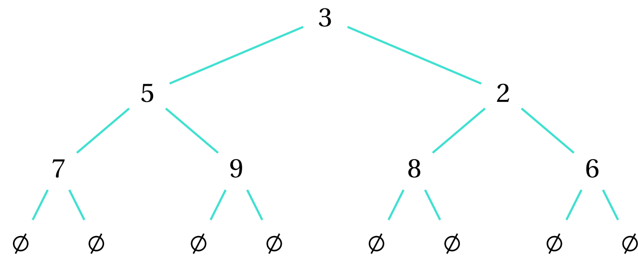
infixe pour lequel l'étiquette du nœud en cours est traitée **entre** celles des sous-arbres gauche et droit,

postfixe pour lequel l'étiquette du nœud en cours est traitée **après** celles des deux sous-arbres gauche et droit.

En OCaml, on peut par exemple coder le parcours préfixe ainsi :

```
let rec prefixe a =
    match a with
    | Vide -> []
    | Noeud (e, fg, fd) -> e :: prefixe fg @ prefixe fd
```

(R) La complexité d'un parcours préfixe dépend de la structure de l'arbre. Si celui-ci est équilibré, moins d'appels récursifs seront nécessaires que s'il est en forme de peigne. On notera cependant que, comme l'opérateur concaténation `@` présente une complexité linéaire, proportionnelle à la longueur de la première opérande, la performance du code ci-dessus n'est pas



Parcours préfixe : 3 5 7 9 2 8 6
Parcours infixe : 7 5 9 3 8 2 6
Parcours postfixe : 7 9 5 8 6 2 3

FIGURE 9 – Parcours en profondeur d'un arbre binaire : on choisit la convention de traiter le fils gauche avant le fils droit.

optimale.

Soit n le nombre de nœuds de l'arbre. Le calcul de la complexité de `(e : : prefixe fg @ prefixe fd)` conduit à :

$$C(n) = 1 + C_g + n_g + C_d$$

car le coût de la concaténation est proportionnel à la longueur de la première liste. Dans le cas d'un peigne à gauche, on aurait trouvé :

$$C(n) = 1 + n - 1 + C(n-1) + C_0 = 1 + n - 1 + C(n-1) + 1 = n + 1 + C(n-1) = \frac{(n+1)(n+2)}{2} = O(n^2)$$

ce qui légitime la seconde approche avec accumulateur !

```
let rec prefixe a acc =
    match a with
    | Vide -> acc
    | Noeud (e, fg, fd) -> e :: prefixe fg (prefixe fd acc)
```

R D'après l'expression `e : : prefixe fg (prefixe fd acc)`, on peut écrire la complexité de la manière suivante : $C(n) = 1 + C_g + C_d$.

Supposons que l'arbre est en forme de peigne à gauche : c'est le cas pour lequel il y aura un maximum d'appels récursifs. Alors on peut simplifier la relation précédente en :

$$C(n) = 1 + C_{n-1}$$

Il s'agit d'une suite arithmétique de raison 1 et de premier terme $C(0) = 1$. On en déduit que $C(n) = n + 1 = O(n)$. La complexité du parcours prédfixe dans le pire des cas est linéaire.

G Proprétés et manipulation des arbres binaires

■ **Exemple 3 — Relation entre la taille et les nœuds.** Soit un arbre binaire à n nœuds et de hauteur h. On se propose de démontrer les propriétés suivantes :

1. $h+1 \leqslant n \leqslant 2^{h+1}-1$
2. cet arbre possède $n+1$ sous-arbres vides.

■ **Exemple 4 — Dénombrer les arbres binaires.** On considère des arbre binaires et on cherche à trouver toutes les structures possibles avec n nœuds.

1. Dénombrer les arbres binaires qui possèdent 0, 1, 2, 3 et 4 nœuds.
2. On peut montrer [a] la suite ainsi formée constitue les nombres de Catalan :

$$C(n) = \frac{1}{1+n}\binom{2n}{n}$$

a. à faire en cours de math ;-)

■ **Exemple 5 — Nombre de feuilles.** On considère un arbre binaire à n nœuds. Soit f le nombre de feuilles de l'arbre. Montrer que $f \leqslant \frac{n+1}{2}$.

H Arbre binaire de recherche (ABR)

■ **Définition 23 — Arbre binaire de recherche.** Soit un ensemble d'étiquettes $\mathcal{E}$ muni d'un ordre total. Un arbre binaire de recherche est un arbre binaire étiqueté par $\mathcal{E}$ dont les nœuds $N(e,g,d)$ vérifient la propriété suivante :

- l'élément e est plus grand que toutes les étiquettes du sous-arbre gauche g,
- l'élément e est plus petit que toutes les étiquettes du sous-arbre droit d.

■ **Exemple 6 — Arbres binaires de recherche.** La figure 10 représentent deux arbres binaires de recherche : l'un est étiqueté avec des entiers et est équilibré, l'autre est étiqueté avec des chaînes de caractères et n'est pas équilibré.

L'implémentation en OCaml s'appuie sur l'arbre binaire défini au chapitre précédent :

```
type 'a bst = Vide | Noeud of 'a * 'a abr * 'a abr
```

C'est un type **polymorphe**, c'est-à-dire que les étiquettes sont de type générique `'a` mais possèdent un ordre total, c'est-à-dire on doit pouvoir comparer n'importe quel élément de l'ensemble avec un autre.

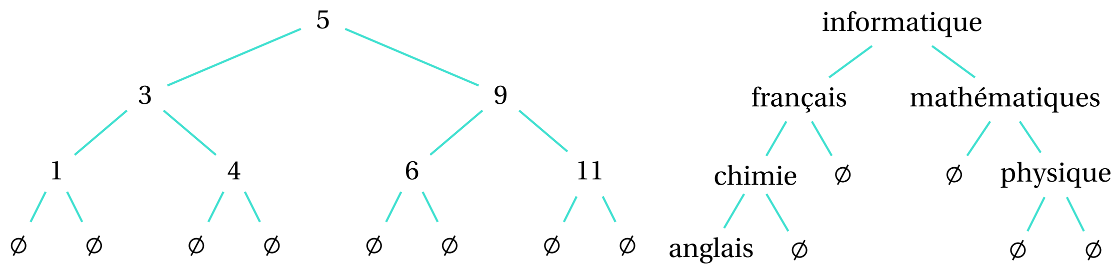


FIGURE 10 – Arbres binaires de recherche équilibré (à gauche) et non équilibré (à droite)

I Opérations sur les arbres binaires de recherche

Les principales opérations sur les arbres de recherche sont :

1. la recherche d'un élément, opération de complexité $O(h)$,
2. l'insertion d'un élément, opération de complexité $O(h)$,
3. la suppression d'un élément $O(h)$.

R Si l'arbre est un peigne, ces opérations deviennent de complexité $O(n)$, si n est le nombre de nœuds de l'arbre. C'est le pire des cas.

Le meilleur des cas se produit lorsque l'arbre est équilibré : ces opérations sont alors donc de complexité $O(\log n)$.

Toute la question est donc d'opérer sur un arbre binaire de recherche et, **simultanément**, de le maintenir dans un état équilibré pour obtenir des performances optimales.

R L'insertion ou la suppression dans un arbre binaire de recherche garantit que la structure d'arbre binaire est respectée à la fin de l'opération. L'équilibre de l'arbre n'est pas garanti.

Les figure 11 et 12 illustrent les opérations d'insertion et de suppression dans un arbre binaire.

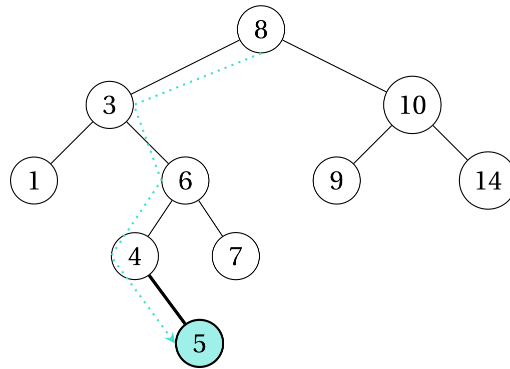
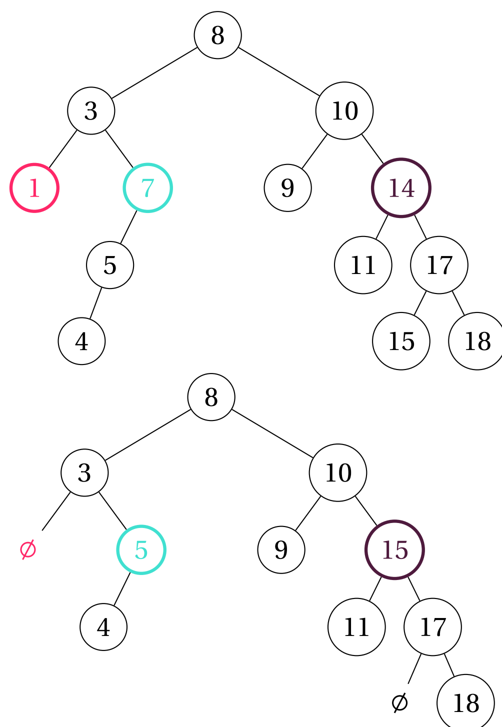


FIGURE 11 – Illustration de l'insertion d'un nœud d'étiquette 5 dans un arbre binaire de recherche



Trois cas sont possibles : le nœud qu'on retire

1. ne possède pas de fils,
2. possède un seul fils,
3. possède deux fils.

FIGURE 12 – Illustration de l'opération supprimer un nœud dans un arbre binaire de recherche

J Maintenir l'équilibre ⇢ HORS PROGRAMME

Le déséquilibre d'un arbre binaire équilibré est engendré par une opération d'insertion ou de suppression. Les techniques d'équilibrage s'appuient généralement sur les opérations d'insertion et de suppression normales suivies d'une manipulation de l'arbre pour lui redonner l'équilibre. Cette manipulation est souvent une rotation des sous-arbres.

■ **Exemple 7 — Arbres binaires de recherche automatiquement équilibrés.** On peut citer notamment :

1. Les arbres AVL,
2. Les arbres rouges et noirs.