DES ARBRES AUX TAS

À la fin de ce chapitre, je sais :

- ☞ définir un tas-min et un tas-max
- ☞ expliquer l'algorithme du tri par tas
- ☞ utiliser un tas pour créer une file de priorité
- ☞ appliquer les files de priorités à l'algorithme de Dijsktra

A Des arbres

■ **Définition 1 — Arbre.** Un arbre est un graphe connexe, acyclique et enraciné.

Ⓡ 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.

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

■ **Définition 2 — 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 3 — 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 4 — Arité d'un nœud.** L'arité d'un nœud est le nombre de ses fils.

■ **Définition 5 — Feuille.** Un nœud d'arité nulle est appelé une feuille.

■ **Définition 6 — Profondeur d'un nœud.** La profondeur d'un nœud est le nombre d'arêtes qui le sépare de la racine.

■ **Définition 7 — Hauteur d'un arbre.** La hauteur d'un arbre est la plus grande profondeur d'une feuille de l'arbre.

■ **Définition 8 — 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 9 — 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 10 — Arbre recouvrant.** Un arbre recouvrant d'un graphe G est un sous-graphe couvrant de G qui est un arbre.

ă

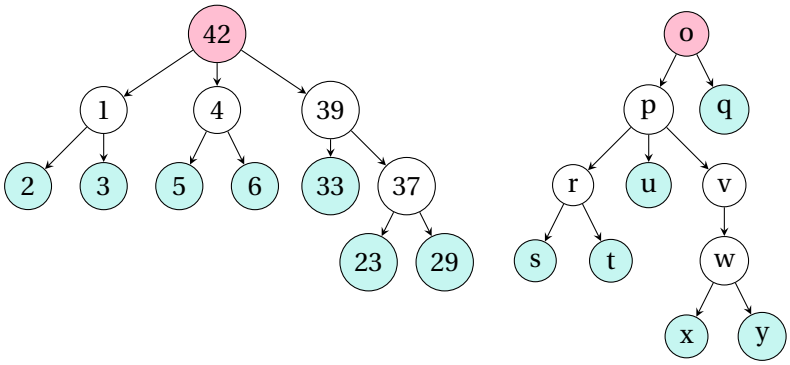


FIGURE 1 – Exemples d'arbres enracinés. Les racines des arbres sont en rouge, les feuilles en turquoise. Le tout forme une forêt.

B Arbres binaires

■ **Définition 11 — Arbre binaire.** Un arbre binaire est un arbre tels que tous les nœuds ont une arité inférieure ou égale à deux : chaque nœud possède au plus deux fils.

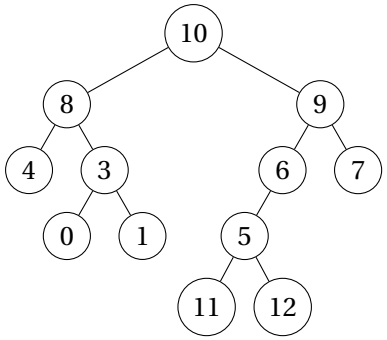


FIGURE 2 – Arbre binaire

■ **Définition 12 — Arbre binaire strict.** Un arbre binaire strict est un arbre dont tous les nœuds possèdent zéro ou deux fils.

■ **Définition 13 — 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.

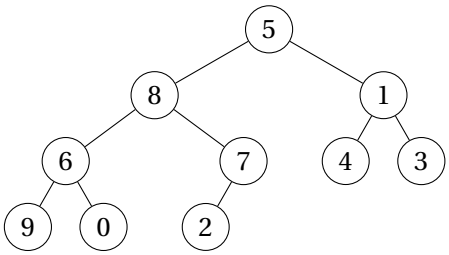


FIGURE 3 – Arbre binaire parfait

■ **Définition 14 — 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 Induction et arbre binaire

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.

■ **Définition 15 — Étiquette d'un nœud.** Une étiquette d'un nœud est une information portée au niveau d'un nœud.

■ **Définition 16 — 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. NIL est un arbre binaire appelé arbre vide (parfois noté $\circ$),
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.

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

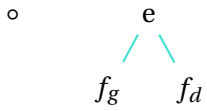


FIGURE 4 – Arbre binaire et définition inductive : arbre vide à gauche, arbre induit par e, f_g et f_d à droite

■ **Définition 17 — 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 $\mathcal{P}(\text{NIL})$ est vraie, c'est-à-dire que la propriété est vraie pour l'arbre vide,
2. (**PAS D'INDUCTION**) 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 $a \in \mathcal{A}$, $\mathcal{P}(a)$ est vraie.

■ **Définition 18 — 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 $\phi(\text{NIL})$,
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. $h(\text{NIL}) = 0$,
2. $h\big((f_g, e, f_d)\big) = 1 + \max\big(h(f_g), h(f_d)\big)$.

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

D Tas binaires

a Définition

■ **Définition 19 — Tas max et tas min.** On appelle tas max (resp. tas min) un arbre binaire parfait étiqueté par un ensemble ordonné E tel que l'étiquette de chaque nœud soit inférieure (resp. supérieure) ou égale à l'étiquette de son père. La racine est ainsi la valeur maximale (resp. minimale) du tas.

b Implémentation

On peut naturellement implémenter un tas par un type `a' arbre` mais également par un tableau `Array` en numérotant les nœuds selon la numérotation Sosa-Stradonitz (cf. figure 7).

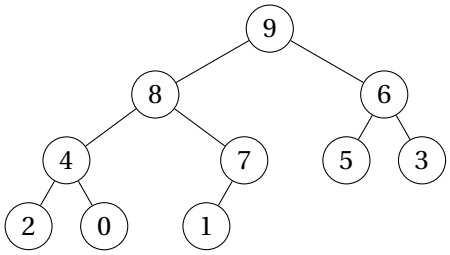


FIGURE 5 – Tas max

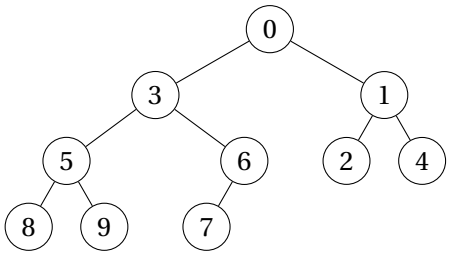


FIGURE 6 – Tas min

R Un tas implémenté par un tableau est une structure de taille donnée, fixée dès la construction du tas : on ne pourra donc pas représenter tous les tas, uniquement ceux qui pourront s'inscrire dans le tableau. Par ailleurs, pour construire cette structure et la préserver lors de l'exécution d'algorithmes, il est important que l'on puisse faire évoluer les éléments à l'intérieur du tas. C'est pourquoi cette structure de donnée doit être muable.

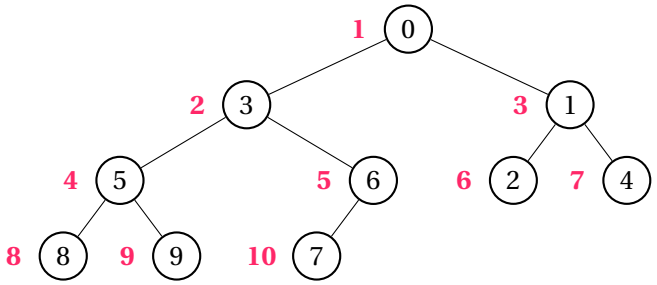
■ **Définition 20 — Numérotation Sosa-Stradonitz d'un arbre binaire.** Cette numérotation utilise les puissances de deux pour identifier les nœuds d'un arbre binaire. La racine se voit attribuer la puissance 0. Le premier élément de chaque niveau k de la hiérarchie possède l'indice 2^k. Ainsi, sur le troisième niveau d'un arbre binaire, on trouvera les numéros $8, 9, 10, 11, 12, 13, 14$ et 15. Cette numérotation est utilisée dans le domaine de la généalogie.

R Comme les langages de programmation comptent à partir de 0, on choisit souvent la convention de positionner la raciner dans la case d'indice 0 et décaler ensuite tous les indices de 1.

c Opérations

On s'intéresse à la construction et à l'évolution d'un tas au cours du temps : comment préserver la structure de tas lorsqu'on ajoute ou retire un élément ?

On définit des opérations descendre et faire monter un élément dans un tas qui préservent la structure du tas. Ce sont des opérations dans un tas sont des opérations dont la complexité



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

FIGURE 7 – Implémentation d'un tas min par un tableau selon les indices de la numérotation Sosa-Stradonitz. On vérifie que les fils du nœud à l'indice k se trouvent à l'indice $2k$ et $2k+1$

est $O(h(a)) = O(\log n)$.

Faire monter un élément dans le tas

Cette opération est expliquée sur la figure 8.

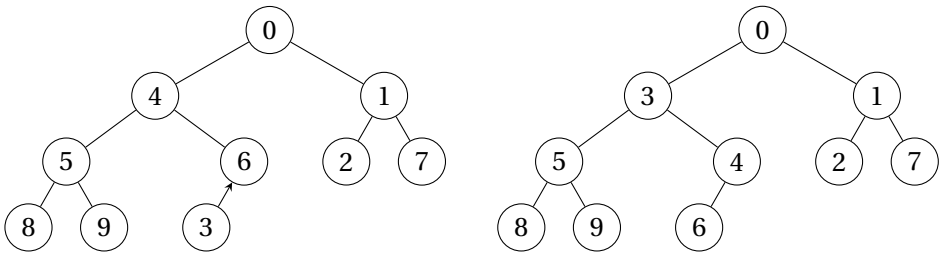


FIGURE 8 – Tas min : à gauche, l'élément 3 doit monter dans le tas min. À droite, on a échangé les places de 3 avec les pères jusqu'à ce que la structure soit conforme à un tas min

Faire descendre un élément dans le tas

Faire descendre dans le tas se dit aussi tamiser le tas et est expliqué sur la figure 9.

Construire un tas

On peut imaginer construire un tas en faisant monter les éléments ou en faisant descendre les éléments au fur et à mesure. Ces deux méthodes ne présentent pas la même complexité .

Si l'on procède en faisant monter les éléments, on considère que la racine est un tas à un élément et on intègre les éléments restant du tableau dans ce tas en les faisant monter. Dans

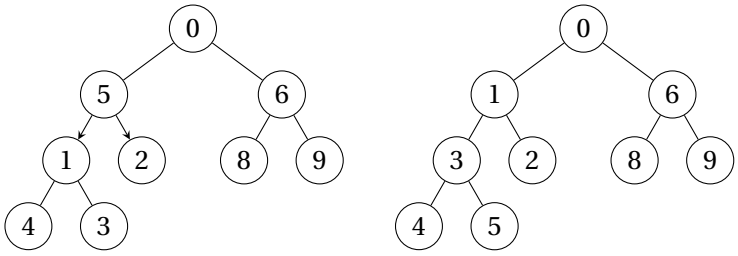


FIGURE 9 – Tas min : à gauche, l'élément 3 doit descendre dans le tas min. À droite, on a échangé la place de 3 avec le fils le plus petit pour que la structure soit conforme à un tas min

le pire des cas, on doit faire monter les $n-1$ nœuds depuis le niveau de profondeur maximale (hauteur de l'arbre) jusqu'à la racine et donc répéter l'opération monter (de complexité $\log n$). C'est pourquoi cette méthode présente une complexité en $O(n \log n)$.

La seconde méthode considère que chaque feuille est un tas à un élément. On fait descendre les $\lfloor n/2 \rfloor$ premiers éléments à partir du $\lfloor n/2 \rfloor$ [e] dans les tas. Au fur et à mesure, on réunit ces tas en un plus gros tas. Dans le pire des cas, on peut montrer que cette méthode est linéaire en $O(n)$. En effet, un élément i est à la hauteur $\lfloor \log_2 i \rfloor$ et il descend au maximum de $h - \lfloor \log_2 i \rfloor$ pour trouver sa place. Il est donc plus efficace de faire descendre les éléments pour créer un tas.

R Dans un tas de taille n, la première feuille se situe à l'indice $n/2$.

E Tri par tas binaire

■ **Définition 21 — Tri par tas.** Le tri par tas procède en formant d'un tas à partir du tableau à trier. Pour un tri ascendant, on utilise un tas-max et pour un tri descendant un tas-min.

On peut considérer que cette méthode est une amélioration du tri par sélection : le structure de tas permet d'éviter la recherche de l'élément à sélectionner.

Le tri par tas est un tri comparatif en place et non stable. Sa complexité dans le pire des cas est en $O(n \log n)$ car il nécessite au pire de descendre les n éléments au niveaux des feuilles dans le tas.

Vocabulary 1 — Heap sort ⇜⇝ Tri par tas

L'algorithme 1 fait appel à un tas-max et son implémentation est radicalement simple, tout le cœur du mécanisme de tri reposant sur la structure de tas.

Algorithme 1 Tri par tas, ascendant

```
1 : Fonction TRI_PAR_TAS(t)
2 :     n ← nombre d'éléments de t
3 :     Faire un tas-max de t
4 :     pour k de n − 1 à 0 répéter
5 :         Échanger t[0] et t[k]                    ▷ Le plus grand va à la fin
6 :         Faire descendre t[0] dans le tas t[: k]   ▷ Le nouvel élément descend
7 :     renvoyer t
```

F File de priorités implémentée par un tas

Une autre application des tas binaires est l'implémentation d'une file à priorités.

■ **Définition 22 — TAD File de priorités.** Une file de priorités est une extension du TAD file dont les éléments sont à valeur dans un ensemble $E = (V, P)$ où P est un ensemble totalement ordonné. Les éléments de la file sont donc des couples valeur - priorité.

Les opérations sur une file de priorités sont :

1. créer une file vide,
2. insérer dans la file une valeur associée à une priorité (ENFILER),
3. sortir de la file la valeur associée la priorité maximale (ou minimale) (DÉFILER).

L'utilisation d'un tas permet d'obtenir une complexité en $O(\log n)$ pour les opérations DÉFILER et ENFILER d'une file de priorités. Pour des algorithmes comme celui du plus court chemin de Dijkstra, c'est une solution intéressante pour améliorer les performances de l'algorithme.