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 Tas binaires

a Définition

■ **Définition 1 — 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.

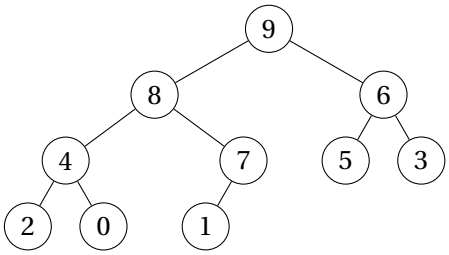


FIGURE 1 – Tas max

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 3).

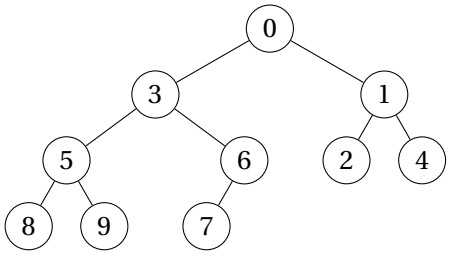
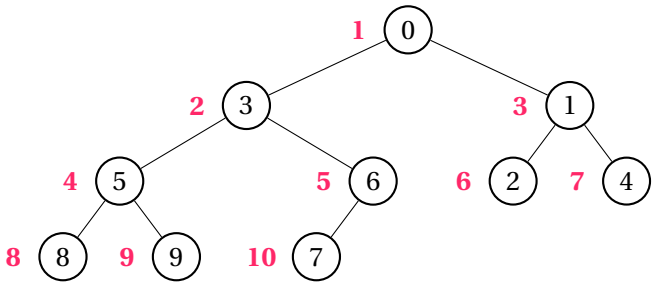


FIGURE 2 – Tas min

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 2 — 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.



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

FIGURE 3 – 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 $2k+1$

R Comme, en informatique, les indices commencent à 0, on choisit souvent la convention de positionner la racine du tas dans la case d'indice 0 et décaler ensuite tous les indices de 1. Les fils d'un nœud d'indice k se situe alors en $2k+1$ et $2k+2$.

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

Faire monter un élément dans le tas

Cette opération est expliquée sur la figure 4.

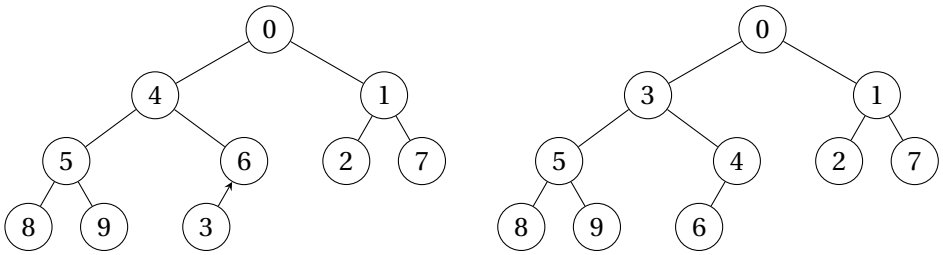


FIGURE 4 – Tas min : à gauche, l'élément 3 doit monter dans le tas min. À droite, on a échangé les places de 3 avec les pères (6 puis 4) jusqu'à ce que la structure soit conforme à un tas min

Faire descendre un élément dans le tas

Faire descendre dans le tas se dit aussi tamiser le tas et est expliqué sur la figure 5.

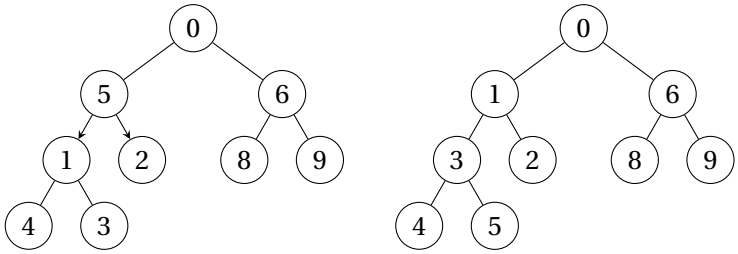


FIGURE 5 – Tas min : à gauche, l'élément 5 doit descendre dans le tas min. À droite, on a échangé la place de 5 avec le fils le plus petit (d'abord 1 puis 3) pour que la structure soit conforme à un tas min

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 .

1. 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 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 $O(n \log n)$.
2. 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 de hauteur i descend dans le pire des cas $h-1-i$ pour trouver sa place. $C(h) = \sum_{i=0}^{h-1} 2^i (h-1-i)$ car à la hauteur i il y a au plus 2^i éléments. Donc [1] $C(h) = 2^h - h - 1 = O(2^h)$ Or, dans un arbre binaire parfait, on a $2^h \leqslant n \leqslant 2^{h+1}$ si n est la taille de l'arbre. Finalement, $C(h) = O(n)$. Il est donc plus efficace de faire descendre les éléments pour construire un tas.

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

B Tri par tas binaire

■ **Définition 3 — 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
```

1. à vous de calculer !

C File de priorités implémentée par un tas

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

■ **Définition 4 — 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.