

Chapitre 21

Structures de données

Sommaire.

1	Arbres.	1
1.1	Arbres binaires.	1
1.1.1	Définitions.	1
1.1.2	Parcours.	1
1.2	Tas.	1
1.2.1	Définitions.	1
1.2.2	Implémentation.	1
1.3	Arbre binaire de recherche.	1
2	Arbres bicolores.	1
3	Dictionnaires.	1
4	Tables de hachage.	1
4.1	Résolution des collisions par sondage.	1
4.2	Résolution des collisions par chaînage.	1

Les propositions marquées de ★ sont au programme de colles.

1 Arbres.

1.1 Arbres binaires.

1.1.1 Définitions.

Définition 1: Arbre binaire.

- Un **arbre binaire** est une structure de données hiérarchique où chaque élément est un **noeud**.
Il est défini de manière inductive :
- Soit c'est une arbre vide.
 - Soit il est constitué d'un noeud et de deux arbre binaires disjoints, appelés **fils gauche** et **fils droit**

Définition 2: Racine

La **racine** d'un arbre est le seul noeud qui n'a pas de père.

Définition 3: Feuilles, noeuds internes.

Les **feuilles** d'un arbre sont les noeuds qui n'ont pas de fils.
Une **branche** est un chemin de la racine à une feuille.
Les **noeuds internes** sont les noeuds qui ont au moins un fils.
On a donc $| \text{feuilles} | + | \text{noeuds internes} | = | \text{noeuds} |$

Définition 4: Arbre binaire strict.

Un arbre binaire strict est un arbre où tous les noeuds internes ont deux fils.

Proposition 5: Dénombrement. ★

Dans un arbre binaire strict non vide, le nombre de feuilles est égal au nombre de noeuds internes plus un.

Preuve :

On a d'abord : $| \text{feuilles} | = | \text{noeuds} | - | \text{noeuds internes} |$.
On dénombre les liaisons père-fils : $N = 2 \cdot | \text{noeuds internes} |$ (les noeuds internes ont deux fils).
On dénombre les liaisons fils-père : $N = | \text{noeuds} | - 1$ (la racine n'a pas de père).
En combinant les deux : $| \text{noeuds} | - 1 = 2 | \text{noeuds internes} |$.
Donc $| \text{feuilles} | = | \text{noeuds internes} | + 1$.

Définition 6: Vocabulaire.

La **taille** d'un arbre est son nombre de noeuds.
La **profondeur** d'un noeud est sa distance à la racine.
Un **niveau** d'un arbre est l'ensemble des noeuds de même profondeur.
La **hauteur** d'un arbre est la profondeur maximale de ses noeuds.
Convention: La hauteur d'un arbre vide est -1, la hauteur d'un arbre réduit à sa racine est 0.

Définition 7: Arbre binaire parfait.

Un arbre binaire est **parfait** si toutes ses feuilles sont de même profondeur et que tous ses noeuds internes possèdent deux fils.

Proposition 8: Nombre de noeuds d'un arbre parfait. ★

Un arbre binaire parfait de hauteur h a $2^{h+1} - 1$ noeuds.
De plus, pour tout $k \in [0, h]$, il a 2^k noeuds de profondeur k .

Preuve :

Montrons le par récurrence sur la hauteur.
Initialisation. Un arbre de hauteur 0 est réduit à sa racine et de taille $1 = 2^{0+1} - 1$.
Hérédité. Supposons la propriété vraie sur les arbres de hauteur h .
Soit un arbre binaire parfait de hauteur $h + 1$.
Un noeud de profondeur $k + 1$ est le fils d'un noeud de profondeur k , qui a deux fils.
Ainsi, il y a deux fois plus de noeuds de profondeur $k + 1$ que de noeuds de profondeur k .
Alors le nombre de noeuds total vaut $\sum_{k=0}^h 2^k = \frac{2^{h+1}-1}{2-1} = 2^{h+1} - 1$.
La propriété est vraie pour $h + 1$.
Par récurrence, elle est vraie pour tout $h \in \mathbb{N}$.

Définition 9: Arbre binaire complet.

Un arbre binaire de hauteur h est **complet** si tous ses niveaux sont remplis, sauf peut-être le dernier, qui est rempli de gauche à droite.

Corrolaire 10: Nombre de noeuds d'un arbre complet.

La taille d'un arbre binaire complet de hauteur h est comprise entre 2^h et $2^{h+1} - 1$.
La hauteur d'un arbre complet de taille n est $\lfloor \log n \rfloor$.

Définition 11: Étiquette.

Une **étiquette** est une valeur associée à un noeud d'un arbre.

1.1.2 Parcours.

Définition 12: Parcours en profondeur. ★

Un **parcours en profondeur** est un parcours où on termine d'explorer une branche avant d'en visiter une autre.
Il existe trois types de parcours en profondeur :

- **Préfixe** : on visite le noeud, puis le fils gauche, puis le fils droit.
- **Infixe** : on visite le fils gauche, puis le noeud, puis le fils droit.
- **Postfixe** : on visite le fils gauche, puis le fils droit, puis le noeud.

Définition 13: Parcours en largeur. ★

Un **parcours en largeur** est un parcours où on visite les noeuds de même profondeur avant de passer à la profondeur suivante.

On utilise une file pour implémenter ce parcours.

1.2 Tas.

1.2.1 Définitions.

Définition 14: File de priorité.

Une **file de priorité** est une structure de données qui permet de stocker des éléments d'un ensemble totalement ordonné.
Interface :

- Insérer un nouvel élément.
- Extraire l'élément le plus grand.
- Modifier la valeur d'un élément.
- Tester si la file est vide.

Définition 15: Tas.

Un **tas** est un arbre binaire complet tel que tout noeud porte une étiquette supérieure à celle de ses fils.

1.2.2 Implémentation.

Définition 16: Percoler vers le bas. ★

Algorithme 1 : Percoler vers le bas

Entrées : Un tas A , un indice i et une valeur v
Sorties : Un tas similaire à A , tel que $A[i] = v$

```
1  $A[i] \leftarrow v$ .
2  $\text{max} \leftarrow$  indice du noeud d'étiquette maximale entre  $A[i]$  et ses fils.
3 si  $\text{max} \neq i$  alors
4    $A[i] \leftarrow A[\text{max}]$ .
5   percoler_vers_le_bas( $A$ ,  $\text{max}$ ,  $v$ ).
6 fin
```

Terminaison.
Les lignes 1, 2, 3, 4 se terminent. Le variant d'appel est la hauteur du noeud d'indice i .
Complexité.
Notons $T(h)$ le nombre d'opérations élémentaires pour une certaine hauteur h .
On a $T(0) = \alpha$ et $T(h) = \alpha + T(h - 1)$ donc $T(h) = \alpha h = O(h)$.
Or $2^h \leq n \leq 2^{h+1} - 1$ avec n le nombre de noeuds de l'arbre.
Donc $h \leq \log_2(n) \leq h + 1$: complexité dans le pire des cas en $O(\log n)$.
Correction.
Soit h la hauteur d'un noeud d'indice i .
Si $h = 0$, alors c'est une feuille et $\text{max} = i$: il n'y a pas d'appel.
Supposons que l'appel est correct pour une certaine hauteur $h - 1$. Montrons que l'appel sur h fonctionne.
On prend un indice i d'un noeud de hauteur h .
Si $\text{max} = i$ alors par hypothèse de récurrence, les sous-arbres de i sont des tas et l'appel est correct.
Si $\text{max} \neq i$ alors on remplace $A[i]$ par le max et la condition est donc vérifiée entre i et ses fils.
Par hypothèse de récurrence, on sait que l'appel récursif est correct.

Définition 17: Percoler vers le haut.

Algorithme 2 : Percoler vers le haut

Entrées : Un tas A , un indice i et une valeur v
Sorties : Un tas similaire à A , tel que $A[i] = v$

```
1  $A[i] \leftarrow v$ 
2  $\text{min} \leftarrow$  indice du noeud d'étiquette minimale entre  $A[i]$  et son père.
3 si  $\text{max} \neq i$  alors
4    $A[i] \leftarrow A[\text{max}]$ 
5   percoler_vers_le_haut( $A$ ,  $\text{min}$ ,  $v$ )
6 fin
```

Mêmes propriétés que pour **percoler_vers_le_bas**.

Définition 18: Modification.

Algorithme 3 : Modification

Entrées : Un tas A , un indice i et une valeur v
Sorties : Un tas similaire à A , tel que $A[i] = v$

```
1 si  $v > A[i]$  alors
2   percoler_vers_le_bas( $A$ ,  $i$ ,  $v$ )
3 fin
4 sinon
5   percoler_vers_le_haut( $A$ ,  $i$ ,  $v$ )
6 fin
```

Sa correction / complexité découle de celles de **percoler_vers_le_bas** et **percoler_vers_le_haut**.

Définition 19: Insertion.

Algorithme 4 : Insertion

Entrées : Un tas A et une valeur v
Sorties : La valeur v a été insérée dans le tas A .

```
1 Ajouter  $v$  à la fin du tas.
2 percoler_vers_le_haut( $A$ ,  $n$ ,  $v$ ).
```

Définition 20: Extraction.

Algorithme 5 : Extraction

Entrées : Un tas A
Sorties : La valeur maximale a été extraite du tas A .

```
1 si le tas est de taille 1 alors
2   supprimer son élément et le renvoyer.
3 fin
4 Garder en mémoire le plus grand élément du tas (le premier).
5 Remplacer le premier élément par le dernier.
6 Le percoler vers le bas.
7 retourner le plus grand élément.
```

Définition 21: Construire un tas. ★

Algorithme 6 : Construire un tas

Entrées : Un tableau T
Sorties : T tel qu'il vérifie la condition de tas.

```
1 pour  $i$  de  $\lfloor n/2 \rfloor$  à 1 faire
2   percoler_vers_le_bas( $T$ ,  $i$ ,  $T[i]$ ).
3 fin
```

Complexité.
Soit H la hauteur du tas. On sait que **percoler_vers_le_bas** est en $O(H)$.
À une certaine profondeur p , il y a au plus 2^p noeuds.
De plus, ces noeuds sont à hauteur soit $h = H - p$, soit $h = H - p - 1$, donc $p = H - h$ ou $p = H - h - 1$.
La complexité de **percoler_vers_le_bas** sur un noeud de hauteur h est donc de αh , $\alpha \in \mathbb{R}$.
Ainsi, il y a au plus 2^{H-h} noeuds de hauteur h .
Alors, dans le pire des cas :

$$\sum_{h=0}^H \alpha h \cdot 2^{H-h} = \alpha 2^H \sum_{h=0}^H \frac{h}{2^h}$$

On pose $f : x \mapsto \sum_{h=0}^x x^h$, alors $f(x) = \sum_{h=0}^x x^h = \frac{x^{x+1}-1}{x-1}$.
De plus, on pose $g : x \mapsto xf'(x)$, et pour $x < 1$: $f'(x) = \frac{Hx^{H+1}-Hx^H+x^{H+1}}{(x-1)^2} + 1 = \frac{1}{(x-1)^2} + Hx^H \frac{x-1-\frac{1}{x}}{(x-1)^2} \rightarrow \frac{1}{(x-1)^2}$.
Alors $g(x) \leq \frac{x}{(x-1)^2} + \beta$ en particulier pour $\frac{1}{2}$.

$$\sum_{h=0}^H \alpha h 2^{H-h} \leq \alpha 2^H (2 + \beta)$$

Alors $\alpha 2^H \sum_{h=0}^H \frac{h}{2^h} = O(2^H = n)$.

1.3 Arbre binaire de recherche.

Définition 22: Arbre binaire de recherche.

Un **arbre binaire de recherche (ABR)** a des noeuds étiquetés dans un ensemble totalement ordonné et tel que l'étiquette de chaque noeud interne est supérieur aux étiquettes de son sous-arbre gauche, et inférieure aux étiquettes de son sous-arbre droit.
Ce type d'arbres n'est pas nécessairement strict.

Proposition 23: Élément minimal.

Le minimum se trouve à l'extrémité gauche de l'arbre.

Preuve :

Par récurrence sur la hauteur h de l'arbre.
Initialisation: Pour $h = 0$, un seul élément, c'est l'extrémité gauche.
Hérédité: Supposons que la propriété est vraie pour les arbres de hauteur $h - 1$.
Soit un arbre binaire de recherche de hauteur h .
1er cas: Pas de sous-arbre gauche, la racine est le minimum.
2ème cas: Il y a un sous-arbre gauche, le minimum y appartient, or ce sous-arbre est de hauteur inférieure.
Par hypothèse, le minimum est à l'extrémité gauche de ce sous-arbre.
Donc le minimum est à l'extrémité gauche de l'arbre.
Par récurrence, on a bien la propriété.

Proposition 24: Parcours infixe. ★

Soit \mathcal{A} un ABR de hauteur h et P_h : «Le parcours infixe de \mathcal{A} donne une liste triée».

Preuve :

Initialisation. Pour $h = 0$ c'est trivial car l'arbre est réduit à sa racine.
Hérédité. Supposons P_h vrai pour toute hauteur **strictement** inférieure à h . Montrons P_h .
On note \mathcal{A}_g le fils gauche de \mathcal{A} , \mathcal{A}_d son fils droit et r sa racine.
Par supposition, le parcours est correct sur \mathcal{A}_g et \mathcal{A}_d car ils sont de hauteurs strictement inférieures à h .
Le parcours infixe parcourt d'abord \mathcal{A}_g , puis r , puis \mathcal{A}_d .
On a que tout élément de \mathcal{A}_g est inférieur à r et que tout élément de \mathcal{A}_d est supérieur à r par propriété des ABR.
Donc le parcours de \mathcal{A}_g puis r puis \mathcal{A}_d est dans l'ordre croissant.

2 Arbres bicolores.

Définition 25: Arbre bicolore.

Un **arbre bicolore** est un arbre binaire de recherche où chaque noeud est coloré en rouge ou noir.
Les feuilles sont noires.
Pour tout noeud rouge, ses fils sont noirs.
Toutes les branches d'un noeud donnent le même nombre de noeuds noirs.

3 Dictionnaires.

Définition 26

Les **dictionnaires** permettent de manipuler des ensembles d'associations.
On appelle **clé** un élément de l'ensemble de départ et **valeur** son élément associé.
Interface:

- **Insertion** (clé, valeur).
- **Recherche** de clé.
- **Suppression** de clé.

Contraintes:

- Les clés sont toutes de mêmes types.
- Les valeurs sont toutes de mêmes types.
- Les clés sont uniques.

Si les clés sont des entiers, on peut utiliser un tableau.
On parle de table à adressage direct, on appelle chaque case du tableau une **alvéole**.

4 Tables de hachage.

Définition 27: Fonction de hachage.

Une **fonction de hachage** h est une fonction qui associe à chaque clé un entier, appelé **haché**.

Définition 28: Table de hachage.

Une **table de hachage** est un tableau associatif, où chaque case est une alvéole.
On utilise une fonction de hachage pour associer une clé à une alvéole.

4.1 Résolution des collisions par sondage.

Définition 29

Principe : S'il n'y a pas de place, on en cherche ailleurs.
Comment choisir le «ailleurs» ?
On définit un cycle σ , qu'on applique successivement au hachage d'une clé, jusqu'à trouver une place libre.

4.2 Résolution des collisions par chaînage.

Définition 30

Principe : On chaîne les clés avec le même hachage.
Dans le pire des cas : insertion en $O(1)$, recherche et suppression en $O(n)$.

Définition 31: Facteur de remplissage.

Le **facteur de remplissage** α est le rapport entre le nombre d'éléments stockés et le nombre d'alvéoles.

Théorème 32: Complexité. ★

Dans une table de hachage où les collisions sont résolues par chaînage, la recherche d'un élément a une complexité temporelle moyenne en $O(1 + \alpha)$.

Preuve :

On se place dans le cas où les hachés sont uniformément distribués.
1er cas: La recherche échoue.
Le calcul de la valeur hachée est en $O(1)$, le parcours moyen d'une liste chaînée est en $O(\alpha)$.
Ainsi, pour toute la recherche, on obtient une complexité moyenne en $O(1 + \alpha)$.
2ème cas: La recherche réussit.
Comme on est susceptible de s'arrêter avant la fin du parcours, la complexité est meilleure que dans le premier cas, donc en $O(1 + \alpha)$.