# Algorithmique des structures de données arborescentes
# Feuille 6 - Chemins dans les arbres binaires

## 1 Rappel sur les arbres quasi-parfaits

Nous avons vu que le nombre de nœuds $n$ d'un arbre binaire parfait de hauteur $h$ est $n = 2^{h+1} - 1$. Autrement dit, $n + 1 = 2^{h+1}$, ou encore, en appliquant la fonction $\log_2$ :

$$h = \log_2(n + 1) - 1.$$

La hauteur d'un arbre parfait est donc **très petite** par rapport à son nombre de nœuds. Lorsque on utilise ces arbres comme structure de données, cette propriété nous permet d'implémenter les opérations usuelles avec une complexité en $O(h)$ (nous l'avons déjà vus avec les ABRs).
Mais le nombre de nœuds d'un arbre parfait est contraint : c'est une « puissance de 2 moins 1 ». On s'intéressera donc a des arbres se rapprochant le plus possible des arbres parfaits : les quasi-parfaits. Par la suite nous utiliserons des arbres quasi-parfaits pour définir les tas binaires.

**Exercice 1 : Arbres quasi-parfaits**

Montrer qu'un arbre quasi-parfait de hauteur $h$ peut avoir entre $2^h$ et $2^{h+1} - 1$ nœuds. En déduire que la hauteur d'un arbre quasi-parfait à $n$ nœuds est, asymptotiquement, $O(\log_2 n)$.

## 2 Chemins dans les arbres binaires

Etant donné un arbre binaire, une suite de directions (gauche ou droite) peut définir un chemin depuis la racine de l'arbre jusqu'à un nœud. Ce nœud est obtenu en partant de la racine de l'arbre, et en suivant les directions données dans la liste, dans l'ordre.

Pour coder une liste de directions menant à à un nœud nous utiliserons un nouveau type :

```
type direction = L | R.
```

Ainsi un chemin sera du type `direction list`.

**Exercice 2 : Suivre un chemin dans un arbre binaire quelconque**

Écrire une fonction `follow_path` qui prend en entrée un arbre binaire `t` (quelconque) et une liste de directions `l` et renvoie l'élément obtenu en suivant les directions de `l` dans `t`.

**Note.** Dans cette fonction et les suivantes, vous pouvez utiliser l'instruction `invalid_arg "message d'erreur"` pour les cas où la fonction est indéfinie.

**Exercice 3 : Trouver le chemin vers une clé dans un ABR**

Écrire une fonction `path_to_key` qui prend en entrée un arbre binaire de recherche et une clé et renvoie une liste de directions indiquant le chemin jusqu'au nœud de l'arbre contenant cette clé.

## 3 Chemins vers les feuilles dans un arbre binaire quasi-parfait

Dans la suite de l'étude des structures de données arborescentes, nous nous interesserons à l'implémentation des files de priorités au moyen d'arbres quasi-parfait. Nous aurons besoin pour cela de trouver le chemin jusqu'à la dernière feuille (dans un parcours en largeur) d'un arbre quasi parfait, ou jusqu'à la première place « libre » pour ajouter une nouvelle feuille.

**Exercice 4 : Chemin vers la dernière feuille d'un arbre quasi-parfait**

1. Soit $t = (r, tg, td)$ un arbre binaire de hauteur $h$. Rappeler les conditions sur $tg$ et $th$ pour que $t$ soit quasi-parfait.
2. En déduire un algorithme linéaire pour vérifier qu'un arbre binaire est quasi-parfait.
3. La dernière feuille (dans un parcours en largeur) d'un arbre binaire quasi-parfait non vide est sa feuille de profondeur maximale la plus à droite. Montrer que pour tout arbre binaire quasi-parfait $t$ de taille $n$, si $b_1 b_2 \cdots b_k$ est la représentation binaire de $n$ alors, en interprétant 0 comme « gauche » et 1 comme « droite », $b_2 \cdots b_k$ est le chemin jusqu'à la dernière feuille de $t$.

   **Conseil.** Commencez par dessiner plusieurs petits exemples d'arbres quasi-parfaits et vérifiez cette propriété dessus.

La chemin vers la dernière feuille peut se calculer en fonction de la taille de l'arbre.

**Exercice 5 : Chemin menant à la dernière feuille**

1. Écrire une fonction `pow2` qui calcule $2^n$ en exponentiation rapide, en tenant compte de la parité de $n$.
2. En utilisant la fonction `pow2` et l'exercice précédent, écrire une fonction `get_dirlist : int -> direction list` prenant en paramètre un entier $n$ et qui renvoie la liste de directions menant à la dernière feuille d'un arbre quasi-parfait à $n$ nœuds (on obtiendra cette liste uniquement par le calcul).
3. En déduire une fonction qui prend en paramètre un arbre $t$ quasi-parfait et une valeur $c$ et ajoute une nouvelle feuille à $t$ contenant la valeur $c$ de sorte que l'arbre obtenu reste un arbre quasi-parfait..