

## Algo

- Numero: 4
- Prof: Alexandre Duret-Lutz
- Date: 24 Octobre 2017

## Objectif

On va voir un theoreme qui fait le cafe. Notament pour avoir la complexite d'une fonction recursif.

## Theoreme general

Ce theoreme permet de resoudre les équations de complexites de la forme:

**Tips:** Il sera donne en partiel. Il faut quand meme savoir l'utiliser.

$$T(n) = aT\left(\frac{n}{b} \pm \Theta(1)\right) + f(n) \implies \Theta(n \log(n))$$

ou  $a \geq 1, b \geq 1$ , et  $f(n)$  fonction positive

Il y a 4 cas:

1. si  $f(n) = \mathcal{O}(n^{\log_b(a)-\varepsilon})$  pour un  $\varepsilon \geq 0$  alors  $T(n) = \Theta(n^{\log_b a})$
2. si  $f(n) = \Theta(n^{\log_b(a)})$  pour un  $\varepsilon \geq 0$  alors  $T(n) = \Theta(n^{\log_b a} \times \log n) = \Theta(f(n) \log n)$
3. si  $f(n) = \Omega(n^{\log_b(a)+\varepsilon})$  pour un  $\varepsilon \geq 0$  et de plus il existe  $c \in [0, 1[$  tq  $af(n/b) \leq cf(n)$  alors  $T(n) = \Theta(f(n))$
4. sinon le theoreme ne s'applique pas.

## Exemples

### 1<sup>er</sup> exemple

$$T(n) = 3T\left(\frac{n}{4}\right) + \sqrt{n}$$

Donc  $a = 3$  et  $b = 4$ . De plus  $\log_4(3) \simeq 0.792$

$\sqrt{n} = n^{0.5}$  on est donc dans le cas 1.

$\mathcal{O}(n^{\log_4 3 - \varepsilon})$  avec  $\varepsilon = 0.1$  par exemple.

**2<sup>eme</sup> exemple**

$$T(n) = 3T\left(\frac{n}{4}\right) + n$$

Donc  $a = 3$  et  $b = 4$ .

On est dans le cas 3.

$\Omega(n^{\log_4 3 + \varepsilon})$  avec  $\varepsilon = 1$  et de plus il existe  $c = 3/16 \leq 1$  tel que  $af(n/b) \leq cf(n)$

$$3\left(\frac{n}{4}\right)^2 \leq cn^2$$

$$\frac{3}{16} \leq c$$

$$\text{Donc } T(n) = \Theta(n^2)$$

**La substitution**

$$T(n) = 2T\left(\frac{n}{2}\right) + cn \log_2 n$$

1<sup>er</sup> substitution:

$$\begin{aligned} T(n) &= 2T\left(2T\left(\frac{n}{4}\right) + c\frac{n}{2} \log_2 \frac{n}{2}\right) + cn \log_2 n \\ &= 4T\left(\frac{n}{4}\right) + cn(\log_2(\frac{n}{2}) + \log_2 n) \end{aligned}$$

2<sup>eme</sup> substitution:

$$\begin{aligned} T(n) &= 4T\left(2T\left(\frac{n}{8}\right) + c\frac{n}{4} \log_2 \frac{n}{4}\right) + cn(\log_2(\frac{n}{2}) + \log_2 n) \\ &= 8T\left(\frac{n}{8}\right) + cn(\log_2(\frac{n}{4}) + \log_2(\frac{n}{2}) + \log_2 n) \end{aligned}$$

On voit un pattern apparaitre.

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n \log n)$$

$$a = 2b = 2f(n) = \Theta(n \log n)$$

On est dans le 4<sup>eme</sup> cas du theoreme general.

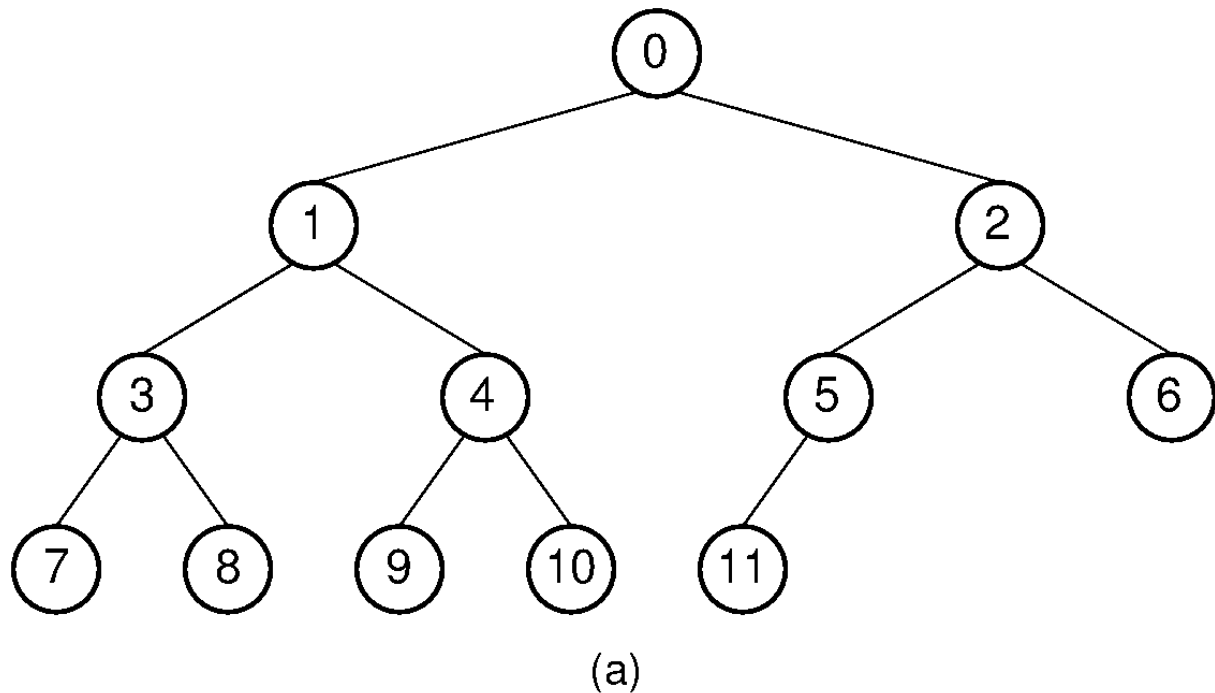
$$T(n) = nT(n) + cn \left( \sum_{k=0}^{\log_2(n)-1} \log_2 n - \sum_{k=0}^{\log_2(n)-1} k \right)$$

$$T(n) = \Theta(n) + \Theta(n \log(n) \log(n))$$

$$= \Theta(n \log(n) \log(n))$$

### Arbre parfait

Arbre binaire (chaque sommet a au plus deux fils), avec tous les niveaux pleins, sauf éventuellement le derniers ou les feuilles sont toutes a gauche.



**Figure 1:** Arbre parfait

Chaque numero de l'arbre va correspondre sa place dans une liste. Cela est une representation simple:

0	1	2	3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	---	---	---	----	----

- $\text{leftchild}(i) = 2 * i + 1$  (s'il existe)
- $\text{rightchild}(i) = 2 * i + 2$  (s'il existe)

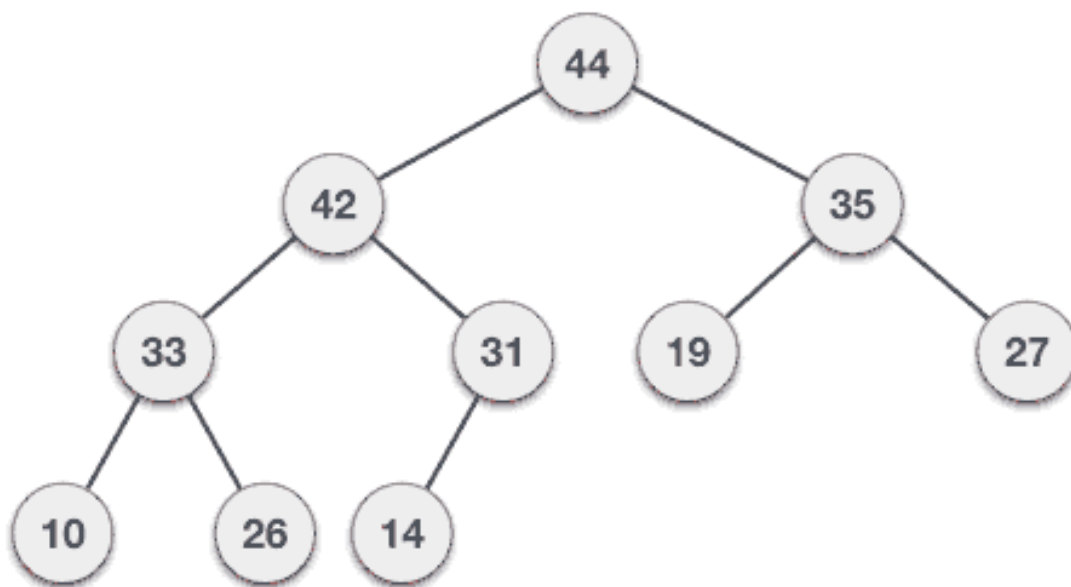
Interet:

Un arbre parfait se represente avec un tableau en lisant les valeurs ligne par ligne.

## Tas

### Structures

Arbre parfait avec la contrainte de tas (chaque noeud est plus grand que ses fils). Pour un tas "max".



**Figure 2:** Tas max

Pour voir l'animation des algos d'ajouts et suppression d'elements: [Cliquez ICI](#)

**Algos**

```
1 def Heapify(A, m, i): //m la taille du tas
2     // On suppose fg(i) et fd(i) sont des tas
3     // mais i pas forcément.
4     l = leftChild(i)
5     r = rightChild(i)
6     if l < m and A[l] > A[i]:
7         largest = l
8     else:
9         largest = i
10    if r < m and A[r] > A[largest]:
11        largest = r
12    if largest != i:
13        swap(A, largest, i)
14        Heapify(A, m, largest)
```

```
1 def BuildHeap(A, m):
2     for i in range(n // 2 - 1, -1, -1):
3         Heapify(A, m, i)
```

```
1 def HeapSort(A, m):
2     BuildHeap(A, m)
3     for i in range(m - 1, 0, -1): // Le cas ou i = 0 est inutile a
        traiter
4         swap(A, 0, i)
5         Heapify(A, i, 0)
```

**Complexite**

$$T_H(n) \leq T_H\left(\frac{2}{3}n\right) + \Theta(1)$$

On est dans le cas 2 du theoreme general. Donc:

$$T(n) \leq \Theta(\log n)$$