6 – Méthode diviser pour régner

Méthode général pour résoudre des problèmes, qui s'applique particulièrement bien en informatique.
Elle consiste à diviser un problème en plusieurs sous-problèmes indépendants, puis à les résoudre récursivement.

1 – Présentation de la méthode

On veut résoudre le problème A.

Si on sait :

1. Transformer le problème A en un problème B.
2. Résoudre le problème B.
3. Transformer la solution du problème B en une solution du problème A

Alors on sait résoudre le problème A.

Ex :

Supposons que l'on veuille construire une arche en pierres. Cela ce révèle difficile car, tant que toutes les pierres ne sont pas mise, l'arche ne tient pas. C'est le problème difficile que l'on cherche à résoudre. Et nous avons un autre problème, plus simple à résoudre, c'est d'utiliser un gabarit en bois.
En partant de la, il est plus simple de construire l'arche en pierre en posant les pierres sur le gabarit en bois.
La question qui va donc ce poser est :
Peut on utiliser ce problème facile (faire une arche en pierre, avec un gabarit en bois), pour résoudre le problème plus compliqué de faire une arche en pierre sans utiliser le gabarit en bois ?

La réponse est oui, et l'on va procéder par étapes :

1. Construire un gabarit en bois et le positionner au bon endroit
2. Résoudre le problème plus facile, poser les pierres sur le gabarit
3. Pour revenir au problème initial, on retire le gabarit

C'est comme cela que l'on va pouvoir résoudre un problème difficile en se basant sur un problème plus facile.

La méthode « diviser pour régner » va utiliser cette idée, mais sur des problèmes ou la notion de taille va apparaître. On va résoudre des problèmes de grande taille en se ramenant à des problèmes de plus petite taille.

Si l'on prend l'exemple du tri, arrivera t-on a trier une grande liste de nombre, si l'on sait trier une petite liste de nombres.

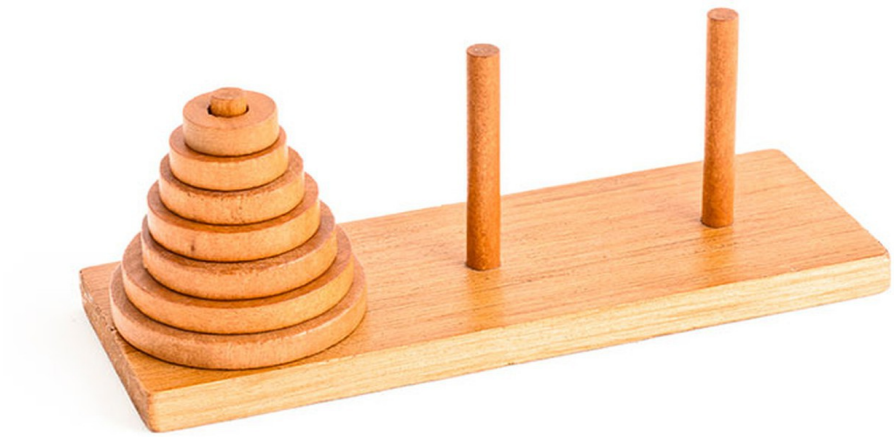
Comment passer de l'un à l'autre, et comment utiliser la solution du petit problème pour résoudre le gros problème ?

Résolution en 3 étapes :

1. **Diviser** pour faire apparaître les sous-problèmes à résoudre
2. **Régner** pour résoudre effectivement les sous-problèmes
3. **Combiner** pour obtenir une solution du problème initial

S'applique très bien dans le cadre de l'informatique.

2 – Exemple 1 : Les tours de Hanoï



Le but est de transférer les cercles du pilier A vers le pilier C en suivant 2 règles :

- On ne peut déplacer qu'un disque à la fois (le plus en haut sur les piliers)
- On ne peut poser un disque que sur un disque plus grand ou un pilier vide

Avec peu de disque, le problème est assez simple à résoudre

Ex avec 2 disques :

On prend le disque en haut sur le pilier A, on le met sur le pilier B, puis on prend le 2° disque du pilier A, on le place sur le pilier C, puis l'on prend le premier disque sur le pilier B et on le met sur le 2° disques sur le pilier C.

Avec 3 disques, cela devient plus compliqué, mais ça va nous permettre de mieux comprendre comme se passe la résolution du problème.

Pour résoudre ce problème, l'étape cruciale va être de déplacer le grand disque du pilier A vers le pilier C.
Après les règles vu précédemment, plusieurs conditions doivent être remplis :

- D'abord, le pilier C doit être libre, il ne doit pas y avoir de disque dessus, sinon, ce grand disque, nous serions obligé de poser ce disque sur un disque plus petit.
- Le grand disque, sur le pilier A, doit être accessible, ce qui signifie que les 2 plus petits disques doivent être ailleurs, et ne peuvent pas se trouver sur le pilier C, et sont donc nécessairement sur le pilier B.

Pour résoudre se problème avec 3 disques, on va donc le décomposer de la façon suivante :

- On commence par amener les 2 plus petits disques sur le pilier B
- En suite on peut transférer le grand disque du pilier A vers le pilier C
- Puis on termine en prenant les 2 plus petits disques et en les déplaçant vers le pilier C

Pour ce faire, on va déplacer :
- le 1° disque du pilier A vers le pilier C
- le 2° disque du pilier A vers le pilier B
- le 1° disque du pilier C vers le pilier B
- le 3° disque du pilier A vers le pilier C
- le 1° disque du pilier B vers le pilier A
- le 2° disque du pilier B vers le pilier C
- le 1° disque du pilier A vers le pilier C

Les questions qu'il faut se poser :

- Diviser → Quels problèmes « plus petits » considérer ?
  (Pour résoudre le problème avec n disques, on s'est ramené à un problème avec n-1 disques)
- Cas de base → Comment résoudre les problèmes les plus petits ?
  (Problème que l'on peut résoudre directement sans faire de raisonnement trop compliqué)
- Combiner → A partir d'une solution du problème plus petit, comment obtenir une solution du problème plus grand ?
  (Si je sais résoudre le problème avec n-1 disques, comment en déduire une solution avec n disques)

Le cas de base :

> Avec un seul disque, comment amener le disque du pilier A vers le pilier C ?
> La solution est simple, on prend le disque du pilier A et on le déplace vers le pilier C.

Diviser pour régner :

> Plus compliqué, si on à n disques sur le pilier A, comment les amener sur le pilier C ?
> On va donc diviser, c'est à dire utiliser la solution du problème à n-1 disques.
> On prend les n-1 premiers disques, on les mets sur le pilier B (on suppose que l'on est capable de faire cela), puis on place le grand disque sur le pilier C, puis on finalise en ramenant tous les n-1 disques du pilier B vers le pilier C.

On a donc une méthode général pour résoudre le problème des tours de Hanoï en utilisant la méthode « diviser pour régner ».

On peut traduire ça assez simplement en un programme Python :

```
def hanoi(n, depart, milieu, arrivee):
    # n est le nombre de disques
    # depart est le pilier de départ
    # milieu, celui du milieu
    # arrivee, celui de fin
    if n == 1: #Cas de base
        # Si l'on a qu'un seul disque, il suffit de le transférer
        # Du pilier de départ, au pilier d'arrivé
        print("Disque ", n, " : ", depart, " -> ", arrivee)
    elif n >= 2: #Diviser pour régner
        # Le pilier de départ contient le dernier disque
        # Le pilier d'arrivée est libre
        # Le pilier du milieu contient les n-1 autres disques
        hanoi(n - 1, depart, arrivee, milieu)
        # On peut donc prendre le plus grand disque
        # Et le transférer directement sur le pilier de départ
        print("Disque ", n, " : ", depart, " -> ", arrivee)
        # On a plus qu'a transférer les n-1 autres disques
        hanoi(n - 1, milieu, depart, arrivee)
```

On a donc résolut le problème des tours de Hanoï indépendamment du nombre de disques.

Si l'on appel notre fonction Hanoï, on va donc avoir l'affichage suivant :

```
Disque  1  :  A  ->  C
Disque  2  :  A  ->  B
Disque  1  :  C  ->  B
Disque  3  :  A  ->  C
Disque  1  :  B  ->  A
Disque  2  :  B  ->  C
Disque  1  :  A  ->  C
```

On retrouve bien le résultat que l'on à eu précédemment.

Grâce à la méthode « Diviser pour régner », on a pu avoir une solution simple pour résoudre le problème des Tours de Hanoï.

3 – Exemple 2 : Le tri-fusion

On va appliquer la méthode « Diviser pour régner » à l'élaboration d'un algorithme de tri.

Certaines méthodes de tri peuvent se comprendre facilement grâce à la méthode « Diviser pour régner ».

Ex :

[12, 15, 13, 2, 3, 5, 10, 9, 7, 17, 11, 18, 19]

Diviser : diviser le tableau en 2

[12, 15, 13, 2, 3, 5, 10] [9, 7, 17, 11, 18, 19]

Régner : suppose que l'on est capable de trier ses 2 tableaux

[2, 3, 5, 10, 12, 13, 15] [7, 9, 11, 17, 18, 19]

En résultat, on à les 2 sous-tableaux, triés.

Combiner : avoir la version triée du tableau initial

[2, 3, 5, 7, 9, 10, 11, 12, 13, 15, 17, 18, 19]

L'étape importante va être de combiner les 2 tableaux triés, ce qu'on appel la fusion (qui donne son nom a l'algorithme de tri).
Cette partie est relativement simple.

On suppose que l'on a les 2 sous-tableaux suivant, trié après l'étape de régner.

On va avoir besoin d'utiliser les nombres présent dans les 2 sous-tableaux, afin de remplir le tableau de départ, qui contient les données triées.

Pour cela on va avoir 2 marqueurs de position qui vont indiquer les numéros que l'on va considérer.

[2, 3, 5, 10, 12, 13, 15] [7, 9, 11, 17, 18, 19]

Ces marqueurs vont se déplacer de gauche à droite au fur et à mesure que l'on va comparer les différentes valeurs.

On va donc récupérer la plus petites des 2 valeurs, et la mettre dans notre tableau

Puis on va continuer d'avancer dans le parcours des tableaux, et continuer de récupérer les valeurs dans l'ordre croissant.

Le 2° tableau contient encore 3 valeurs que nous ne pouvons plus comparer. Comme ils sont supposé triés, on va pouvoir les ajouter directement à la fin du tableau.

[2, 3, 5, 7, 9, 10, 11, 12, 13, 15, 17, 18, 19]

Complexité du tri-fusion :

nombre d'étapes de recombinaison * longueur du tableau

nombre d'étapes = $\log_2$ (log en base 2)

complexité du tri-fusion = n $\log_2$ n

```
def fusion(tab_1, tab_2): #tableaux ordonnées
    res = [] # tableau final
    pos_1, pos_2 = 0, 0 #position de parcour de nos tableaux
    # on va vérifier que l'on se situe toujours dans les tableaux
    # que l'on souhaite parcourir
    # cette boucle permet de vérifier que l'on est pas sortit du tableau
    # et que l'on pointe bien vers une valeur de ce tableau
    while pos_1 < len(tab_1) and pos_2 < len(tab_2):
        # on va vérifier que la valeur la plus à gauche du tableau 1
        # soit bien plus petite que celle du tableau 2
        if tab_1[pos_1] < tab_2[pos_2]:
            # si elle l'est, on l'ajoute en fin de talbeau (.append)
            res.append(tab_1[pos_1])
            # et on décale le marqueur d'un rang vers la droite
            pos_1 += 1
        # sinon, cela signifie que la plus petite valeur se trouve dans le 2° tableau
        else:
            res.append(tab_2[pos_2])
            pos_2 += 1
    # si pos_1 est strictement plus petit que la longueur du tableau 1
    # c'est qu'il reste encore des nombres dedans
    if pos_1 < len(tab_1):
        # on ajoute ces valeurs à la fin du tableau
        res += tab_1[pos_1:]
        # tab_1[pos_1:] dans tab_1, on prend les valeurs
        # de la position 1 jusqu'a la fin
    # sinon, c'est que les valeurs restantes sont dans le tableau 2
    else:
        res += tab_2[pos_2:]
    return res
```

Parlons un peu de la complexité, c'est à dire le nombre d'opération nécessaire pour fusionner 2 tableaux.

La fusion de 2 tableaux de longueurs n1 et n2 est n1+n2
Ici l'on va parcourir les 2 tableaux de la gauche vers la droite, et à chaque fois que l'on change un marqueur de position, on va faire quelque chose, donc le nombre d'étape que l'on va avoir sera égale à la somme des longueurs des 2 tableaux.

Maintenant que tout est en place, nous pouvons écrire l'algorithme de tri-fusion.

```
def tri_fusion(tab):
    n = len(tab)
    if n <= 1:
        # cas de base
        return tab
    # sinon, il va falloir faire un tri
    # on va donc utiliser la méthode "diviser pour régner"
    else:
        # diviser
        # n//2 = quotient de la division euclidienne
        # on va donc diviser le tableau par 2
        t1 = tab[:n//2] # on prend le début du tableau (0) jusqu'a la moitié
        t2 = tab[n//2:] # on prend la fin (n) du tableau a partir de la moitié
        # Si n n'est pas un nombre pair
        # ex : 13 / 2 = 6
        # le premier tableau contiendra 7 valeurs
        # le deuxième tableau contiendra 6 valeurs
        # régner
        t1_trie = tri_fusion(t1)
        t2_trie = tri_fusion(t2)
        # combiner
        res = fusion(t1_trie, t2_trie)
        return res
```

https://www.lumni.fr/video/la-methode-laquo-diviser-pour-regner-raquo#containerType=serie&containerSlug=la-maison-lumni-lycee