La récursivité et le paradigme « diviser pour régner »

- **Récursivité**

  De l'art d'écrire des programmes qui résolvent des problèmes que l'on ne sait pas résoudre soi-même !

  - **Définition 4 (Définition récursive, algorithme récursif).** *Une définition récursive est une définition dans laquelle intervient ce que l'on veut définir. Un algorithme est dit récursif lorsqu'il est défini en fonction de lui-même.*

- **Récursivité simple**

  Revenons à la fonction puissance $x \rightarrow x^n$. Cette fonction peut être définie récursivement :

$$x^n = \begin{cases} 1 & \text{si } n = 0; \\ x \times x^{n-1} & \text{si } n \geq 1. \end{cases}$$

La récursivité et le paradigme « diviser pour régner »

- **Récursivité multiple**

  Une définition récursive peut contenir plus d'un appel récursif. Nous voulons calculer ici les combinaisons C^n_p en se servant de la relation de Pascal :

$$C^p_n = \begin{cases} 1 & \text{si } p = 0 \text{ ou } p = n; \\ C^p_{n-1} + C^{p-1}_{n-1} & \text{sinon.} \end{cases}$$

- **Récursivité mutuelle**

  Des définitions sont dites *mutuellement récursives* si elles dépendent les unes des autres. Ça peut être le cas pour la définition de la parité :

$$\text{pair}(n) = \begin{cases} \textit{vrai} & \text{si } n = 0; \\ \text{impair}(n-1) & \text{sinon;} \end{cases} \quad \text{et} \quad \text{impair}(n) = \begin{cases} \textit{faux} & \text{si } n = 0; \\ \text{pair}(n-1) & \text{sinon.} \end{cases}$$

La récursivité et le paradigme « diviser pour régner »

- **Récursivité imbriquée**

  La fonction d'Ackermann est définie comme suit :

$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ et } n = 0 \\ A(m-1, A(m,n-1)) & \text{sinon} \end{cases}$$

La récursivité et le paradigme « diviser pour régner »

- **Principe et dangers de la récursivité**
  - **Principe et intérêt :** ce sont les mêmes que ceux de la démonstration par récurrence en mathématiques. On doit avoir :
    - un certain nombre de cas dont la résolution est connue, ces « cas simples » formeront les cas d'arrêt de la récursion ;
    - un moyen de se ramener d'un cas « compliqué » à un cas « plus simple ».
  - La récursivité permet d'écrire des algorithmes concis et élégants.

La récursivité et le paradigme « diviser pour régner »

- **Difficultés :**
  - la définition peut être dénuée de sens :

    Algorithme A(n)

    renvoyer A(n)
  - il faut être sûrs que l'on retombera toujours sur un cas connu, c'est-à-dire sur un cas d'arrêt ; il nous faut nous assurer que la fonction est complètement définie, c'est-à-dire, qu'elle est définie sur tout son domaine d'applications.

La récursivité et le paradigme « diviser pour régner »

- **Non décidabilité de la terminaison**

  Question : peut-on écrire un programme qui vérifie automatiquement si un programme donné *P* termine quand il est exécuté sur un jeu de données *D*?
  - **Entrée:** Un programme P et un jeu de données D.
  - **Sortie:** *vrai* si le programme *P* termine sur le jeu de données *D*, et *faux* sinon.
- **Démonstration de la non décidabilité**
  - Supposons qu'il existe un tel programme, nommé ***termine***, de vérification de la terminaison. À partir de ce programme on conçoit le programme Q suivant :

La récursivité et le paradigme « diviser pour régner »

programme Q
résultat = termine(Q)
tant que résultat = *vrai* **faire** attendre une seconde **fin tant que**
renvoyer résultat

- Supposons que le programme Q —qui ne prend pas d'arguments— termine. Donc termine(Q) renvoie *vrai*, la deuxième instruction de Q boucle indéfiniment et Q ne termine pas. Il y a donc contradiction et le programme Q ne termine pas. Donc, termine(Q) renvoie *faux*, la deuxième instruction de Q ne boucle pas, et le programme Q termine normalement. Il y a une nouvelle fois contradiction : par conséquent, il n'existe pas de programme tel que termine.

- **Le problème de la terminaison est indécidable!!**

La récursivité et le paradigme « diviser pour régner »

- **Diviser pour régner**
  - **Principe**

Nombres d'algorithmes ont une structure récursive : pour résoudre un problème donné, ils s'appellent eux-mêmes récursivement une ou plusieurs fois sur des problèmes très similaires, mais de tailles moindres, résolvent les sous problèmes de manière récursive puis combinent les résultats pour trouver une solution au problème initial.

Le paradigme « diviser pour régner » donne lieu à trois étapes à chaque niveau de récursivité :

- **Diviser :** le problème en un certain nombre de sous-problèmes ;
- **Régner :** sur les sous-problèmes en les résolvant récursivement ou, si la taille d'un sous-problème est assez réduite, le résoudre directement ;
- **Combiner :** les solutions des sous-problèmes en une solution complète du problème initial.

La récursivité et le paradigme « diviser pour régner »

- **Premier exemple : multiplication naïve de matrices**
  - Nous nous intéressons ici à la multiplication de matrices carrés de taille n.
  - **Algorithme naïf**

```
MULTIPLIER-MATRICES(A, B)
Soit n la taille des matrices carrés A et B
Soit C une matrice carré de taille n
Pour i ← 1 à n faire
  Pour j ← 1 à n faire
    cij ← 0
    Pour k ← 1 à n faire
        cij ← cij +aik * bkj
renvoyer C
```

  - Cet algorithme effectue $\Theta(n^3)$ multiplications et autant d'additions.

La récursivité et le paradigme « diviser pour régner »

- **Algorithme « diviser pour régner » naïf**

Dans la suite nous supposerons que n est une puissance exacte de 2. Décomposons les matrices A, B et C en sous-matrices de taille $n/2$ *$n/2$. L'équation $C = A * B$ peut alors se récrire :

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix}$$

- En développant cette équation, nous obtenons :

$$r = ae + bf, \quad s = ag + bh, \quad t = ce + df \quad \text{et} \quad u = cg + dh.$$

Chacune de ces **quatre** opérations correspond à **deux** multiplications de matrices carrés de taille $n/2$ et une addition de telles matrices. À partir de ces équations on peut aisément dériver un algorithme « diviser pour régner » dont la complexité est donnée par la récurrence :

$$T(n) = 8T(n/2) + \Theta(n^2),$$

- l'addition des matrices carrés de taille $n/2$ étant en $\Theta(n^2)$.

La récursivité et le paradigme « diviser pour régner »

- **Analyse des algorithmes « diviser pour régner »**
  - Lorsqu'un algorithme contient un appel récursif à lui-même, son temps d'exécution peut souvent être décrit par une équation de récurrence qui décrit le temps d'exécution global pour un problème de taille n en fonction du temps d'exécution pour des entrées de taille moindre.

La récursivité et le paradigme « diviser pour régner »

- La récurrence définissant le temps d'exécution d'un algorithme « diviser pour régner » se décompose suivant les trois étapes du paradigme de base :

1. Si la taille du problème est suffisamment réduite, $n \leq c$ pour une certaine constante c, la résolution est directe et consomme un temps constant $\Theta(1)$.
2. Sinon, on divise le problème en a sous-problèmes chacun de taille $1/b$ de la taille du problème initial. Le temps d'exécution total se décompose alors en trois parties :
   (a) $D(n)$: le temps nécessaire à la division du problème en sous-problèmes.
   (b) $aT(n/b)$: le temps de résolution des a sous-problèmes.
   (c) $C(n)$: le temps nécessaire pour construire la solution finale à partir des solutions aux sous-problèmes.

La récursivité et le paradigme « diviser pour régner »

- La relation de récurrence prend alors la forme :

$$T(n) = \begin{cases} \Theta(1) & \text{si } n \le c, \\ aT(n/b) + D(n) + C(n) & \text{sinon,} \end{cases}$$

- où l'on interprète n/b soit comme $\lfloor n/b \rfloor$ soit comme $\lceil n/b \rceil$.

La récursivité et le paradigme « diviser pour régner »

- **Résolution des récurrences** :

Théorème 1 (Résolution des récurrences « diviser pour régner »).

Soient $a \ge 1$ et $b > 1$ deux constantes, soit $f(n)$ une fonction et soit $T(n)$ une fonction définie pour les entiers positifs par la récurrence :

$$T(n) = aT(n/b) + f(n),$$

où l'on interprète n/b soit comme $\lfloor n/b \rfloor$, soit comme $\lceil n/b \rceil$.

$T(n)$ peut alors être bornée asymptotiquement comme suit :

1. *Si $f(n) = O(n^{\log_b a - \varepsilon})$ pour une certaine constante $\varepsilon > 0$, alors $T(n) = \Theta(n^{\log_b a})$.*
2. *Si $f(n) = \Theta(n^{\log_b a})$, alors $T(n) = \Theta(n^{\log_b a} \log n)$.*
3. *Si $f(n) = \Omega(n^{\log_b a + \varepsilon})$ pour une certaine constante $\varepsilon > 0$, et si $af(n/b) \le cf(n)$ pour une constante $c < 1$ et n suffisamment grand, alors $T(n) = \Theta(f(n))$.*

Pour une démonstration de ce théorème voir Rivest-Carmen-etal

Il existe d 'autres méthodes de résolution des récurrences : par substitution, changement de variables etc.

La récursivité et le paradigme « diviser pour régner »

- **Algorithmes de tri**
  - **Tri par fusion**

Principe

L'algorithme de tri par fusion est construit suivant le paradigme « diviser pour régner » :

1. Il divise la séquence de n nombres à trier en deux sous-séquences de taille $n/2$.
2. Il trie récursivement les deux sous-séquences.
3. Il fusionne les deux sous-séquences triées pour produire la séquence complète triée.

La récursion termine quand la sous-séquence à trier est de longueur 1 car une telle séquence est toujours triée.

La récursivité et le paradigme « diviser pour régner »

```
TRI-FUSION(A, p, r)
  si p < r alors q ← ⌊(p+r)/2⌋
                 TRI-FUSION(A, p, q)
                 TRI-FUSION(A, q + 1, r)
                 FUSIONNER(A, p, q, r)
```

- **Complexité**
  - Pour déterminer la formule de récurrence qui nous donnera la complexité de l'algorithme TRI-FUSION, nous étudions les trois phases de cet algorithme « diviser pour régner » :
    - **Diviser :** cette étape se réduit au calcul du milieu de l'intervalle [p..r]
    - **Régner :** l'algorithme résout récursivement deux sous-problèmes de tailles respectives n/2
    - **Combiner :** la complexité de cette étape est celle de l'algorithme de fusion qui est de **$\Theta(n)$** pour la construction d'un tableau solution de taille n.

La récursivité et le paradigme « diviser pour régner »

- Par conséquent, la complexité du tri par fusion est donnée par la récurrence :

$$T(n) = \begin{cases} \Theta(1) & \text{si } n = 1, \\ 2T(\frac{n}{2}) + \Theta(n) & \text{sinon.} \end{cases}$$

- Pour déterminer la complexité du tri par fusion, nous utilisons de nouveau le théorème.
  - Ici a=2 et b=2 donc $\log_b a = 1$, et nous nous trouvons dans le deuxième cas du théorème $f(n) = \Theta(n^{\log_b a}) = \Theta(n)$
  - par conséquent : $T(n) = \Theta(n \log n)$.
  - Pour des valeurs de n suffisamment grandes, le tri par fusion avec son temps d'exécution en $\Theta(n \log n)$ est nettement plus efficace que le tri par insertion dont le temps d'exécution est en $\Theta(n^2)$