Recursivité, Programmation récursive

1.Base théorique

La récursivité est basée sur une construction de l'ensemble N des entiers naturel:
un ensemble qui contient 0 et qui contient le successeur de chacun de ses élément contient l'ensemble des entiers naturels.
Démontrer une propriété par récurence c'est montrer qu'elle est vraie pour un ensemble qui contient N.
Si la propriété $\mathcal{P}(0)$ est vraie et si, pour tout n
on peut déduire $\mathcal{P}(n)$ à partir de $\mathcal{P}(n-1)$, alors
la propriété $\mathcal{P}$ est vraie pour l'ensemble des entiers naturels.

On l'utilise souvent en algorithmique sous la forme:
si on sait déterminer une fonction f pour 0 et qu'on peut déterminer $f(n), n > 0$ à partir de $f(n-1)$ alors on peut d'éterminer f sur l'ensemble des entiers naturels.
L'analyse récursive d'un problème consiste à:

- définir les valeurs pour lesquelles la solution est connue
- exprimer le problème à l'aide du même problème pour des données plus simples
- montrer comment composer les solutions des problèmes plus simples pour obtenir la solution du problème (souvent regroupé avec la partie précédente

Exemples

1. **Somme des entiers de 1 à n, $n \geq 1$**
   Si n vaut 1 la somme est 1
   Si $n > 1$ la somme est égale à n + la somme des entiers de 1 à $n - 1$.

```
ButSomme des entiers de 1 à n
Donnée en entrée un entier n ¿0
Donnée en sortieentier la somme
entier SommeEntier(entier n)
Si(n==1)
   retour 1
sinon
   retour n+ SommeEntier(n-1)
fin de SommeEntier
```

   on aurait pu renvoyer $\frac{n(n+1)}{2}$

2. **Somme des cubes entiers de 1 à n, $n \geq 1$**
   Si n vaut 1 la somme est 1
   Si $n > 1$ la somme est égale à n^3 + la somme des cubes des entiers de 1 à $n - 1$.

```
ButSomme des cubes entiers de 1 à n
Donnée en entrée un entier n > 0
Donnée en sortie entier la somme 1^3 + 2^3 + ... + n^3
entier SommeCubes(entier n)
Si(n==1)
   retour 1
sinon
   retour n*n*n+ SommeCubes(n-1)
fin de SommeCubes
```

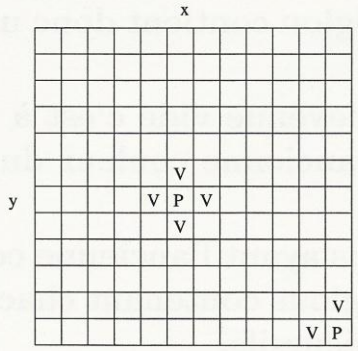
3. Recherche dichotomique dans un tableau T trié de *taille* éléments
   On recherche un élément x entre deux indices *debut* et fin.
   Au premier appel $debut = 0$ et $fin = taille - 1$.
   Si la partie est vide x n'est pas dans le tableau.
   on détermine l'indice *milieu* du milieu du tableau.
   Si l'élément à l'indice *mileu* est x la recherche est terminée;
   si l'élément à l'indice *mileu* est inférieur à x le résultat est celui de la recherche dans la partie restante d'indice supérieurs à *milieu*;
   si l'élément à l'indice *mileu* est supérieur à x le résultat est celui de la recherche dans la partie restante d'indices inférieurs à *milieu*.

But Déterminer si un élément est présent dans un tableau.
Données en entrée: un tableau T trié en ordre croissant,
un élément x à chercher dans T,
l'indice de début et l'indice de fin de la partie de T où effectuer la recherche.
Données en sortie: VRAI si x est présent dans T, FAUX sinon

```
Booleen RechercheDicho (TypeElement T[], Element
x,entier debut,entier fin)
entier milieu Si (debut¿fin)
   retour FAUX
sinon
   milieu=(debut+fin)/2
   Si (T[milieu]==x)
        retour VRAI
    sinon
        Si (T[milieu]<x)
                retour Recherche (T,x,milieu+1,fin)
        sinon
                retourRecherche (T,x,debut,milieu-1)
fin de RechercheDicho
```


4. Changement de couleur d'une région d'une image
   On représente une image carrée par un tableau $N \times N$ d'entiers.
   Chaque point de l'image correspond à une case du tableau et chaque entier correspond à une couleur.
   Un point de coordonnées (x, y) à au plus quatre voisins:
   les points de coordonnées $(x, y-1)$,$(x, y+1)$,$(x-1, y)$ et $(x+1, y)$.

Une région est un ensemble de cases voisines les unes des autres de même couleur.



LES VOISINS

Elle est donc limitée par les bords du tableau ou par des points de couleurs différentes. Le but est d'écrire une fonction permettant de changer la couleur de tous les points d'une région à partir d'un de ses points.

Méthode:
on mémorise l'ancienne couleur du point et on change sa couleur. La région contient donc un point de moins.

Si la région est devenue vide c'est à dire si les voisins ne sont pas de l'ancienne couleur du point, on a fini.

S'il y a des voisins ayant l'ancienne couleur, on change la couleur des régions contenant chacun de ces voisins, par des appels récursifs.

But Changer la couleur des points d'une région.
Données en entrée: un tableau d'entiers (N× N) T
les coordonnées entières x et y d'un point de la région
la nouvelle couleur
Données en sortie : aucune

```
void ChangeZone(entierT[][N],entier x,entier y,
entier nouvelleCouleur)
entier ancienneCouleur
entier vx,vy
ancienneCouleur=T[x][y]
T[x][y]=nouvelleCouleur
pour chaque voisin (vx,vy) de (x,y)
    Si T[vx][vy]==ancienneCouleur
        ChangeZone(T,vx,vy,nouvelleCouleur)
fin de ChangeZone
```

La partie `pour chaque voisin` (vx,vy) de (x,y) ... peut être séparées en quatre traitements:

```
vx=x;vy=y-1
Si (vy>=0 et T[vx][vy]==ancienneCouleur)
        ChangeZone(T,vx,vy,nouvelleCouleur)
vy=y+1
Si (vy<N et T[vx][vy]==ancienneCouleur)
        ChangeZone(T,vx,vy,nouvelleCouleur)
vx=x-1;vy=y
Si (vx>=0 et T[vx][vy]==ancienneCouleur)
        ChangeZone(T,vx,vy,nouvelleCouleur)
vx=x+1
Si (vx<N et T[vx][vy]==ancienneCouleur)
        ChangeZone(T,vx,vy,nouvelleCouleur)
```