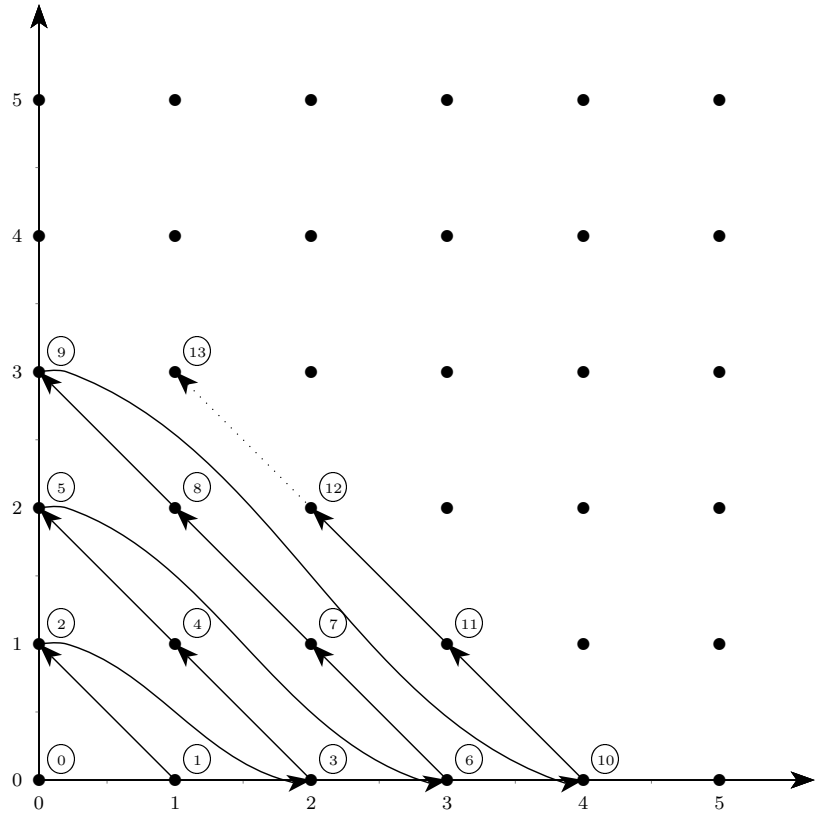
TD 3 - Récursivité

Dans tous les exercices, on essaiera de réaliser des fonctions récursives curryfiées et d'utiliser des filtres plutôt que des `if` lorsque cela est possible.

1 Les incontournables

Exercice 1 Il y a autant de nombres dans $\mathbb{N} \times \mathbb{N}$ que dans $\mathbb{N}$... L'argument de la diagonale fut découvert par le mathématicien allemand Georg Cantor et publié en 1891. Il permet la construction d'une bijection de $\mathbb{N}^2$ dans $\mathbb{N}$ proposant ainsi une numérotation des couples (x, y) de $\mathbb{N}^2$ comme l'illustre la figure ci-dessous.



Comme pour le TD précédent, on privilégie les fonctions récursives avec filtrage.

1. Écrire une fonction `next : int * int -> int * int` qui reçoit un couple d'entiers et retourne les coordonnées du « point suivant » sur la diagonale de Cantor. Par exemple :

```
# next (2,1);;
- : int * int = (1, 2)
# next (0,4);;
- : int * int = (5, 0)
```

2. Écrire une fonction `couple : int -> int * int` qui reçoit un entier naturel n et renvoie les coordonnées du point d'indice n dans le déplacement de Cantor :

```
# couple 7;;
- : int * int = (2, 1)
```

3. Écrire une fonction `indice : int * int -> int` qui réalise l'opération inverse. Par exemple :

```
# indice (2,1);;
- : int = 7
```

Exercice 2 Soit f, une fonction d'un ensemble I dans lui-même et a un élément de I. On définit alors une suite $(u_n)_{n\in\mathbb{N}}$ à valeurs dans I par : $u_0 = a$ et $\forall n \in \mathbb{N},\ u_{n+1} = f(u_n)$. Écrire une fonction récursive `terme` qui reçoit 3 arguments : f, a et n et retourne u_n.

Exercice 3 Écrire une fonction `binaire : int -> unit` qui reçoit un entier naturel non nul et affiche la décomposition binaire ce celui-ci. par exemple :

```
binaire 10;;
1010- : unit = ()
```

Exercice 4 Écrire une fonction `decompose` de type `int -> unit` qui affiche la décomposition d'un entier en produit de nombres premiers. Par exemple, `decompose 36` doit afficher : `2 2 3 3`.

Exercice 5 Deux nombres entiers n et m sont dits amicaux si la somme des diviseurs de l'un est égale à la somme des diviseurs de l'autre et si ces deux sommes valent la somme des deux nombres. Si l'on appelle σ la fonction qui, à un entier associe la somme de ses diviseurs : $\sigma(n) = \sigma(m) = n + m$.

1. Justifier que 220 et 284 sont amicaux.
2. Écrire une fonction `somme` qui renvoie la somme des diviseurs d'un entier naturel non nul.
3. Écrire une fonction `affiche(n)` qui reçoit un entier `n` non nul et affiche la liste des couples de nombres amicaux de l'intervalle [1 ; `n`].

2 Pour s'entrainer

Exercice 6 Écrire une fonction récursive `dichotomie` qui, étant donnés :

- Une fonction continue $f : I \longrightarrow \mathbb{R}$
- Deux réels a et b de I tels que $f(a)f(b) < 0$
- Un réel $\varepsilon > 0$

renvoie une solution sur I de l'équation $f(x) = 0$ approchée à ε près en utilisant le principe de dichotomie. Par exemple avec la fonction $x \longmapsto x^2 - 2$ sur l'intervalle $[0; 10]$, on obtient une valeur approchée de $\sqrt{2}$ à 10^{-4} ainsi :

```
dichotomie (function x -> x**2. -. 2.) 0. 10. 0.0001;;
- : float = 1.4142227172851563
```

Exercice 7 Écrire une fonction `bezout` de signature `int -> int -> int * int` qui calcule des coefficients de Bezout de deux entiers.

Exercice 8 Écrire une fonction `nb` qui à tout entier $n > 1$ associe le nombre de couples d'entiers $(a; b) \in [1, n]^2$ tels que a et b soient premiers entre eux. On rappelle que si $(a, b) \in \mathbb{N} \times \mathbb{N}^*$, $PGCD(a, b) = PGCD(b, r)$ où r est le reste de la division de a par b. Par exemple `nb 4` renvoie `11` car il y a 11 couples de nombres premiers entre-eux dans l'intervalle $[1; 4]$:

$$\big\{(1,1),(1,2),(1,3),(1,4),(2,1),(2,3),(3,1),(3,2),(3,4),(4,1),(4,3)\big\}$$

Exercice 9 Écrire une fonction `premier : int -> bool` qui reçoit un entier et renvoie un booléen indiquant si cet entier est premier ou non.

Correction 1

1. Il faut traiter à part les cas où le point est sur l'axe des ordonnées :

```
let next = function
        | (0,y) -> (y+1,0)
        | (x,y) -> (x-1,y+1)
;;
```

2. Cette fonction est récursive :

```
let rec couple = function
        | 0 -> (0,0)
        | n -> next (couple (n-1))
;;
```

3. 2 idées :
   — idée 1 : on reprogramme une fonction `before` qui fait l'inverse de next et on l'utilise comme dans la fonction couple, ou on fait tout d'un coup :

```
let rec indice = function
        | (0,0) -> 0
        | (x,0) -> 1 + indice (0,x-1)
        | (x,y) -> 1 + indice (x+1,y-1)
;;
```

   — idée 2 : on programme une distance entre 2 points (au sens du schéma) et on calcule la distance à (0,0) :

```
let indice c =
        let rec distance c1 c2 = (* avec c2 avant c1 *)
                if c1 = c2
                then 0
                else 1 + distance c1 (next c2)
        in distance c (0,0)
;;
```

Correction 2 Pas de difficulté pour cet exercice, deux proposition : la première n'est pas terminale, s'appuie sur le fait que $u_n = f\big(u_{n-1}\big)$ et s'écrit naturellement sur :

```
let rec terme f a = function
        | 0 -> a
        | n -> f (terme f a (n-1))
;;
```

La seconde est récursive terminale, mais plus compliquée à comprendre (d'ailleurs, ce n'est pas un objectif du programme d'option info) : si on pose pour $n \in \mathbb{N},\ v_n = u_{n+1}$ alors $\begin{cases} v_0 = u_1 = f(a) \\ \forall n \in \mathbb{N},\ v_{n+1} = f(v_n) \end{cases}$ et $\forall n \in \mathbb{N}^*,\ u_n = v_{n-1}$, ainsi :

```
let rec terme f a = function
        | 0 -> a
        | n -> terme f (f a) (n-1)
;;
```

Correction 3 Il faut simplement faire attention à l'ordre d'affichage :

```
let rec binaire = function
        0 -> ();
        | n when n mod 2 = 0 -> binaire (n/2);print_int 0;
        | n -> binaire (n/2); print_int 1
;;
```

Correction 4 Une solution qui utilise une fonction auxiliaire `decompose(a,b)` qui permet avec le filtrage
- de savoir s'il faut arrêter (si `a` = 1).
- d'afficher `b` s'il divise `a` et de continuer à tester si on peut encore diviser par `b`.
- de continuer à tester les diviseurs suivant si `b` ne divise pas `a`.

```
let decompose n =
    let rec decomposeR = function
        | (1, _) -> ()
        | (n, b) when n mod b = 0 -> print_int b;decomposeR(n/b,b)
        | (n, b) -> decomposeR(n, b+1)
    in decomposeR (n,2)
;;
```

A noter que comme on commence par tester le diviseur 2 puis les suivants, on ne peut trouver que des diviseurs premiers, il est donc inutile de tester si le diviseur affiché est premier. Par exemple :

appel	affichage
`decompose 60`	-
`decomposeR(60,2)`	2
`decomposeR(30,2)`	2
`decomposeR(15,2)`	-
`decomposeR(15,3)`	3
`decomposeR(5,3)`	-
`decomposeR(5,4)`	-
`decomposeR(5,5)`	5
`decomposeR(1,5)`	-FIN-

Correction 5

1. $220 = 2^2 \times 5 \times 11$. Les diviseurs de 220 sont donc : $\{1\ ;\ 2\ ;\ 5\ ;\ 11\ ;\ 4\ ;\ 10\ ;\ 22\ ;\ 55\ ;\ 20\ ;\ 44\ ;\ 110\ ;\ 220\}$
   En détail : $\{1\ ;\ 2\ ;\ 5\ ;\ 11\ ;\ 2\times2\ ;\ 2\times5\ ;\ 2\times11\ ;\ 5\times11\ ;\ 2\times2\times5\ ;\ 2\times2\times11\ ;\ 2\times5\times11\ ;\ 2\times2\times5\times11\}$
   De même : $284 = 2^2 \times 71$. Ses diviseurs sont sont $\{1\ ;\ 2\ ;\ 4\ ;\ 71\ ;\ 142\ ;\ 284\}$.
   Ainsi : $220 + 284 = 504$, $\sigma(220) = 1+2+5+11+4+10+22+55+20+44+110+220 = 504$ et enfin $\sigma(284) = 1+2+4+71+142+284 = 504$. 220 et 284 sont donc amicaux.
2.
```
let somme n =
    let rec sommeR n = function
        | d when d = n -> n
        | d when n mod d = 0 -> d + sommeR n (d+1)
        | d -> sommeR n (d+1)
    in sommeR n 1
;;
```

3. Proposition :

```
let affiche n =
    let rec afficheR = function
        | (a,1) when a = n+1 -> ()
        | (a,b) when b = n+1 -> afficheR(a+1,1)
        | (a,b) when somme a = a + b && somme b = a + b ->
            print_int a;
            print_string "␣et␣";
            print_int b;
            print_newline();
            afficheR(a,b+1);
        | (a,b) -> afficheR(a,b+1)
    in afficheR(1,1)
;;
```

Correction 6

```
let rec dichotomie f a b eps =
    let c = (a +. b) /. 2.0 in
        if b -. a < eps
        then c
        else if f(a) *. f(c) <= 0.0
            then dichotomie f a c eps
            else dichotomie f c b eps
;;
```

ou avec un filtrage (même si tout se passe dans le `when`) :

```
let rec dichotomie f a b eps =
    match (a +. b) /. 2.0 with
        | c when b -. a < eps -> c
        | c when f(a) *. f(c) <= 0.0 -> dichotomie f a c eps
        | c -> dichotomie f c b eps
;;
```

Correction 7 Voici une solution (détails sur demande) :

```
let rec bezout a b =
  let rec euclideR(u,v,r,u',v',r') =
    if r' = 0 then (u,v,r)
    else
      let q = r/r' in
      euclideR(u',v',r',u - q * u', v - q * v', r - q * r')
  in euclideR(1,0,a,0,1,b)
;;
```

Correction 8 APRÈS LE DM ☺

Correction 9

```
let premier n =
        let rec premierR n = function
                  d when d > n-2 -> true
                | d when n mod d = 0 -> false
                | d -> premierR n (d+1)
        in n > 1 && premierR n 2
;;
```