Devoir maison Toussaint

Bartolomeo Ryan

Exercice 1

On peut implémenter de façon naive la suite de Fibonacci en suivant sa définition mathématique :

```
let rec fib n =
  match n with
  0 | 1 -> n
  | n -> fib (n-1) + fib (n-2)
```

Mais cet algorithme devient lent pour de grandes valeurs par sa complexcité en $O(n^2)$.
Une solution serait de faire un appel récursif simple, c'est à dire que chaque appel ne rajoute qu'un seul autre appel sur la pile au lieu de deux :

```
let fib2 n =
  let rec fib_aux n =
    match n with
    | 0 -> (0,0)
    | 1 -> (0,1)
    | n -> let a,b = fib_aux (n-1) in (a+b,a)
  in
    let a,b = fib_aux n in a
```

ici, on a fait une récursivité terminale afin de miniser le coup en mémoire et en temps : cet algorithme a une complexicité en $O(n)$

Exercice 2

```
let rec permute all =
  (*Liste des permutation des éléments*)
  if all = [] then [[]] else (*cas de base*)

  let join elem l = elem::l
  in

  let remove elem l = List.filter ((<>) elem) l
  in

  List.concat( (*assemblage*)
    List.map (*combinaisons*)
      (fun t -> List.map
        (join t)
        (permute (remove t all)))
      all
  )
```

Exercice 3

```
let partition (n:int) (q:int) =
    (*
        On fera la liste l de membres de la somme et on
        compte le nombre de fois où cette somme tombe juste.
        On suppose n>=q
    *)
    if n=q || q=1 then 1 else (*cas de base*)
    let is_good (l:int list) = if (List.fold_left ( + ) 0 l) = n then 1 else 0 in

    (*Modification des facteurs*)
    let (++|) a b = if a=n then b else a+1 in
    let rec increment (l:int list) (p:int): (int list) =
        match l with
        | [] -> []
        | [x] -> [x ++| p]
        | t::q -> let t' = if List.hd q >= n then t ++| p else t in
            t'::increment q t'
    in

    (*Essais*)
    let rec partion (l:int list) : int =
        if List.hd l >= n then 0 (*si le premier membre a atteint n, on a fini*)
        else is_good l + partion (increment l 0) (*recursion des cas*)
    in
    partion (List.init q (fun _->1))
```

On obtient alors le tableau :

n\q	1	2	3	4	5	6	7	8	9	10
1	1	0	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0
3	1	1	1	0	0	0	0	0	0	0
4	1	2	1	1	0	0	0	0	0	0
5	1	2	2	1	1	0	0	0	0	0
6	1	3	3	2	1	1	0	0	0	0
7	1	3	4	3	2	1	1	0	0	0
8	1	4	5	5	3	2	1	1	0	0
9	1	4	7	6	5	3	2	1	1	0
10	1	5	8	9	7	5	3	2	1	1

qui ressemble au triangle de Pascal. On en déduit qu'il y a $\binom{n-1}{q}$ façons différentes d'écrire n comme la somme de q entiers si $n \neq q$ et 1 sinon.