Compositions et partitions d'un entier

Partie I. Compositions

Question 1.

a) On obtient : $(5) \succ (4,1) \succ (3,2) \succ (3,1,1) \succ (2,3) \succ (2,2,1) \succ (2,1,2) \succ (2,1,1,1) \succ (1,4) \succ (1,3,1) \succ (1,2,2) \succ (1,2,1,1) \succ (1,1,3) \succ (1,1,2,1) \succ (1,1,1,2) \succ (1,1,1,1,1)$.

b) Considérons l'égalité : $n = 1 + 1 + \cdots + 1$, composée de n chiffres 1 et de $n-1$ symboles $+$. Définir une composition de n de longueur k revient à choisir $k-1$ symboles $+$ parmi les $n-1$ disponibles et à calculer les k sommes entre chacun des symboles sélectionnés. Par exemple, dans la somme $5 = 1+1+1+1+1$, la composition $(2,1,2)$ revient à sélectionner les deuxième et troisième symboles $+$: $5 = (1+1)+(1)+(1+1)$.
Ainsi, il apparaît que le nombre de compositions de longueur k est égal à $\binom{n-1}{k-1}$.

c) Puisque $k \in [\![1,n]\!]$, on en déduit que le nombre de compositions de n vaut $\sum_{k=1}^{n-1}\binom{n-1}{k-1} = 2^{n-1}$.

Question 2.

a) Soit $a = (p_1, \ldots, p_k)$ une composition de n qui n'est pas la dernière, autrement dit qui contient au moins une valeur de p_i différente de 1. Considérons l'entier i maximal vérifiant cette condition : on a $p_i > 1$ et $p_j = 1$ pour $i < j \leqslant k$. La composition suivante pour l'ordre $\succ$ est alors $b = (p_1, \ldots, p_{i-1}, p_i - 1, k - i + 1)$.

b) On définit :

```
let scinde l =
  let rec aux n = function
    | 1::q -> aux (n+1) q
    | q    -> (n, q)
  in aux 0 l ;;
```

La fonction auxiliaire **aux** utilise un accumulateur **n** qui dénombre le nombre de 1 lus en tête de la liste **l** jusqu'à rencontrer un élement différent de 1.

c) On définit :

```
let composition_suivante l =
  let (n, q) = scinde l in
  match q with
    | []   -> []
    | h::t -> (n+1)::(h-1)::t ;;
```

d) On définit :

```
let compositions n =
  let rec aux l =
    let c = composition_suivante (hd l) in
    match c with
      | [] -> l
      | _  -> aux (c::l)
  in aux [[n]] ;;
```

La fonction **aux** prend en argument la liste des compositions déjà générées et lui ajoute la composition qui suit celle située en tête jusqu'à obtenir la dernière.

Partie II. Partitions

Question 3. On obtient : $(7) \succ (6,1) \succ (5,2) \succ (5,1,1) \succ (4,3) \succ (4,2,1) \succ (4,1,1,1) \succ (3,3,1) \succ (3,2,2) \succ (3,2,1,1) \succ (3,1,1,1,1) \succ (2,2,2,1) \succ (2,2,1,1,1) \succ (2,1,1,1,1,1) \succ (1,1,1,1,1,1,1)$.

Question 4.

a) Soit $a = (p_1, \ldots, p_k)$ une partition de n qui n'est pas la dernière, autrement dit qui contient au moins une valeur de p_i différente de 1. Considérons l'entier i maximal vérifiant cette condition : on a $p_i > 1$ et $p_j = 1$ pour $i < j \leqslant k$. Effectuons la division euclidienne de $p_i + k - i$ par $p_i - 1$: $p_i + k - i = q(p_i - 1) + r$. La partition suivante pour l'ordre $\succ$ est alors $b = (p_1, \ldots, p_{i-1}, \underbrace{p_i - 1, p_i - 1, \ldots, p_i - 1}_{q \text{ fois}})$ si $r = 0$, et $b = (p_1, \ldots, p_{i-1}, \underbrace{p_i - 1, p_i - 1, \ldots, p_i - 1}_{q \text{ fois}}, r)$ sinon.

b) On définit :

```
let rec ajoute n x l = match n with
  | 0 -> l
  | _ -> x::(ajoute (n-1) x l) ;;
```

c) On définit :

```
let partition_suivante l =
  let (n, u) = scinde l in
  match u with
    | []   -> []
    | t::v -> let q = (n + t) / (t - 1)
              and r = (n + t) mod (t - 1) in
              match r with
                | 0 -> ajoute q (t-1) v
                | _ -> r::(ajoute q (t-1) v) ;;
```

d) Cette fonction est semblable à celle de la question 2d :

```
let partitions n =
  let rec aux l =
    let c = partition_suivante (hd l) in
    match c with
      | [] -> l
      | _  -> aux (c::l)
  in aux [[n]] ;;
```

Partie III. Tableaux de Young

Question 5.

a) Il suffit de faire la somme des tailles des listes qui composent la représentation du tableau de Young :

```
let taille t = it_list (fun a b -> a + (list_length b)) 0 t ;;
```

b) C'est presque la même chose avec la fonction suivante :

```
let forme t = it_list (fun a b -> (list_length b)::a) [] t ;;
```

Question 6.

a) Soit $a = (p_1, p_2, \ldots, p_k)$ la partition associée au tableau Young, et $i \in [\![1, k+1]\!]$ la ligne où est ajoutée une case. Pour vérifier qu'on obtient toujours un diagramme associé à une partition, il faut montrer que si $i \geqslant 2$ alors $p_i + 1 \leqslant p_{i-1}$, autrement dit que $p_i < p_{i-1}$.

Raisonnons par l'absurde en supposant $p_i = p_{i-1}$, notons x_i et x_{i-1} les deux derniers éléments de chacune de ces deux lignes, et x l'entier ajouté à la i^{e} ligne. On a $x < x_{i-1}$ et $x > x_i$ donc $x_i < x_{i-1}$, ce qui contredit la définition des tableaux de Young puisque nous avons fait l'hypothèse que x_i et x_{i-1} sont situés dans la même colonne.

Le nouveau diagramme est donc toujours associé à une partition de $n+1$.

Il faut maintenant vérifier que le remplacement de y par x (ou l'ajout de x) dans une ligne ne modifie pas l'ordonnancement relatif de la ligne et de la colonne ou s'effectue ce remplacement. Considérons les quatre cases situées autour de y (notons que certaines peuvent ne pas exister, mais dans ce cas le raisonnement reste le même) :

	c	
a	y	b
	d	

On a $a < y < b$ et par hypothèse $a < x < y$ donc $a < x < b$; l'ordonnancement horizontal n'est pas modifié.
On a $c < y < d$ et $x < y$ donc $x < d$. Il reste à observer que si on avait $x < c$, x aurait été inséré à la ligne précédente, ce qui montre que l'ordonnancement vertical n'est pas modifié.

b) Commençons par écrire une fonction qui prend pour arguments un élément x et une ligne ℓ d'un tableau de Young et qui renvoie un couple (y, ℓ') avec :

- si x est supérieur à tous les éléments de ℓ alors $y = 0$ et ℓ' est la ligne ℓ à qui on a ajouté x en queue ;
- sinon, ℓ' est la liste ℓ dans laquelle y a été remplacé par x.

```
let rec insere_ligne x = function
  | []              -> (0, [x])
  | y::q when y < x -> let z, l = insere_ligne x q in z, y::l
  | y::q            -> y, x::q ;;
```

Il reste ensuite à appliquer l'algorithme d'insertion décrit dans l'énoncé :

```
let rec insere x = function
  | [] -> [[x]]
  | l::q -> let y, m = insere_ligne x l in
            match y with
              | 0 -> m::q
              | _ -> m::(insere y q) ;;
```

c) Pour supprimer l'élément x d'un tableau de Young on procède comme suit :

- si x est situé sur la dernière ligne, il est simplement supprimé de cette ligne ;
- si x est inférieur à tous les éléments de la ligne suivante, il est là encore simplement supprimé ;
- sinon, soit y le plus grand élément de la ligne suivante qui soit inférieur à x ; on remplace x par y et on poursuit l'algorithme en supprimant y de la ligne suivante.