

DS 2 : Génération d'objets combinatoires

Génération des parties d'un ensemble

Question 1.

1.a L'application $\left(\begin{array}{ccc} \mathcal{F}(\llbracket 1, n \rrbracket, \{0, 1\}) & \longrightarrow & \mathcal{P}_n \\ f & \longmapsto & \{k \in \llbracket 1, n \rrbracket / f(k) = 1\} \end{array} \right)$ est clairement l'application réciproque de $A \mapsto \chi_A$.

1.b Par unicité de la décomposition d'un entier en base 2, l'application $\left(\begin{array}{ccc} \{0, 1\}^n & \longrightarrow & \llbracket 0, 2^n - 1 \rrbracket \\ (b_1, \dots, b_n) & \longmapsto & \sum_{k=1}^n b_k 2^{k-1} \end{array} \right)$ est une bijection. Par ailleurs, il est clair que l'application $\left(\begin{array}{ccc} \mathcal{F}(\llbracket 1, n \rrbracket, \{0, 1\}) & \longrightarrow & \{0, 1\}^n \\ f & \longmapsto & (f(1), \dots, f(n)) \end{array} \right)$ est elle aussi bijective. On en déduit, car la composée de plusieurs bijections est encore une bijection, que l'application $\left(\begin{array}{ccc} \mathcal{P}_n & \longrightarrow & \llbracket 0, 2^n - 1 \rrbracket \\ A & \longmapsto & \sum_{k=1}^n \chi_A(k) 2^{k-1} \end{array} \right)$ est une bijection.

Question 2.

2.a La fonction suivante calcule $k \oplus \mathcal{B}$ lorsque k est un entier et \mathcal{B} une liste de liste.

```
let rec ajoute k = function
  [] -> []
  | t::q -> (k::t)::(ajoute k q) ;;
```

2.b Clairement, $\mathcal{P}_n = \mathcal{P}_{n-1} \cup (n \oplus \mathcal{P}_{n-1})$ (\mathcal{P}_{n-1} est l'ensemble des parties de $\llbracket 1, n \rrbracket$ qui ne contiennent pas n , et $n \oplus \mathcal{P}_{n-1}$ celles qui le contiennent).

Par ailleurs, $\mathcal{P}_0 = \{\emptyset\}$; la rédaction est donc naturellement récursive :

```
let rec parties1 = function
  0 -> [[]]
  | n -> let q = parties1 (n-1) in q @ (ajoute n q) ;;
```

Montrons par récurrence sur n que `parties1 n` retourne la liste des éléments de \mathcal{P}_n rangés par ordre croissant.

– C'est clair si $n = 0$.

– Si $n \geq 1$, supposons le résultat acquis au rang $n - 1$. Par hypothèse de récurrence, `q` est rangé par ordre croissant. Il est clair que `ajoute n q` reste rangé par ordre croissant. Or pour tout $(A, B) \in \mathcal{P}_{n-1} \times (n \oplus \mathcal{P}_{n-1})$,

$\varphi(A) \leq \sum_{k=1}^{n-1} 2^{k-1} = 2^n - 1 < 2^n \leq \varphi(B)$, donc `q @ (ajoute n q)` est encore rangé par ordre croissant.

Question 3.

3.a On obtient successivement :

$$A_1 = \{4\}, A_2 = \{3, 4\}, A_3 = \{2, 3, 4\}, A_4 = \{1, 2, 3, 4\}, A_5 = \{1, 3, 4\}, A_6 = \{2, 4\}, A_7 = \{1, 2, 4\},$$

$$A_8 = \{1, 4\}, A_9 = \{3\}, A_{10} = \{2, 3\}, A_{11} = \{1, 2, 3\}, A_{12} = \{1, 3\}, A_{13} = \{2\}, A_{14} = \{1, 2\},$$

$$A_{15} = \{1\}, A_{16} = \emptyset.$$

3.b Montrons par récurrence sur $n \in \mathbb{N}^*$ que A_1, \dots, A_{2^n} sont définis, que $A_{2^n} = \emptyset$, et que $\mathcal{P}_n = \{A_p / p \in \llbracket 1, 2^n \rrbracket\}$.

– C'est clair si $n = 1$ car alors $A_1 = \{1\}$, $A_2 = \emptyset$.

– Si $n \geq 2$, supposons le résultat acquis au rang $n - 1$. On a $A_1 = \{n\}$ et $A_2 = \{n - 1, n\} = A'_1 \cup \{n\}$, en notant $A'_1 = \{n - 1\}$. Par hypothèse de récurrence, $A'_1, \dots, A'_{2^{n-1}-1}$ sont définis, constituent les éléments de $\mathcal{P}_{n-1} \setminus \{\emptyset\}$, et $A'_{2^{n-1}-1} = \{1\}$. On en déduit que $A_1, \dots, A_{2^{n-1}}$ sont définis, constituent les éléments de $n \oplus \mathcal{P}_{n-1}$, et $A_{2^{n-1}} = \{1, n\}$.

Mais alors $A_{2^{n-1}+1} = \{n - 1\}$, et en appliquant de nouveau l'hypothèse de récurrence, on en déduit que $A_{2^{n-1}+1}, \dots, A_{2^{n-1}+2^{n-1}}$ sont définis, constituent les éléments de \mathcal{P}_{n-1} , et $A_{2^n} = \emptyset$.

Sachant que $\mathcal{P}_{n-1} \cup (n \oplus \mathcal{P}_{n-1}) = \mathcal{P}_n$, on peut conclure quant au résultat au rang n .

Question 4.

4.a Il est naturel d'opérer par filtrage :

```
let suivant = fonction
    [] -> failwith "suivant"
  | [1] -> []
  | [n] -> [n-1;n]
  | 1::t::q -> (t-1)::q
  | t::q as l -> (t-1)::l ;;
```

4.b Une solution possible est :

```
let partie2 n =
  let rec aux l = fonction
    0 -> 1
    | k -> let t = hd l in aux ((suivant t)::l) (k-1)
  in aux [[n]] (deux_puissance n - 1) ;;
```

Génération des permutations

Question 5.

5.a $c_i(\sigma)$ est le nombre d'entiers j strictement supérieurs à i qui apparaissent avant i dans la liste $(\sigma(1), \dots, \sigma(n))$. Ainsi :

$$\sigma = (2, 1, 4, 5, 3) \Rightarrow c(\sigma) = (1, 0, 2, 0, 0).$$

5.b La somme $\sum_{i=1}^n c_i(\sigma)$ représente le nombre de couples $(i, j) \in \llbracket 1, n \rrbracket^2$ tel que $i < j$ et $\sigma^{-1}(j) < \sigma^{-1}(i)$. Mais σ est une bijection, donc l'application $(i, j) \mapsto (\sigma(i), \sigma(j))$ est une bijection de $\llbracket 1, n \rrbracket^2$ dans lui-même, et en posant $(i', j') = (\sigma(i), \sigma(j))$, on a :

$$(i < j \text{ et } \sigma^{-1}(j) < \sigma^{-1}(i)) \iff (j' < i' \text{ et } \sigma(i') < \sigma(j')).$$

Ainsi, $\sum_{i=1}^n c_i(\sigma)$ est le nombre d'inversions de σ .

Question 6.

6.a On obtient successivement : $\ell_5 = (5)$, $\ell_4 = (5, 4)$, $\ell_3 = (5, 3, 4)$, $\ell_2 = (5, 3, 2, 4)$, $\ell_1 = (5, 3, 2, 4, 1)$.

6.b Il est clair que ℓ_1 est une liste de longueur n formée des n premiers entiers naturels non nuls, donc représente une permutation. Montrons pour tout $k \in \llbracket 1, n \rrbracket$ que $c_k(\ell_1) = \gamma_k$.

– Si $k = 1$, on sait que ℓ_1 est de la forme $(a_1, \dots, a_{\gamma_1}, 1, a_{\gamma_1+1}, \dots, a_{n-1})$, chaque a_i étant supérieur ou égal à 2 (1 vient d'être inséré après l'élément d'ordre γ_1). On a donc bien $c_1(\ell_1) = \gamma_1$.

– Si $k > 1$, la définition de c_k montre que l'on peut supprimer de la liste ℓ_1 les entiers inférieurs strictement à k sans modifier la valeur de $c_k(\ell_1)$. Autrement dit, $c_k(\ell_1) = c_k(\ell_k)$, et cette dernière valeur, comme pour le cas $k = 1$, vaut clairement γ_k .

Ainsi, $c(\ell_1) = (\gamma_1, \dots, \gamma_n)$.

Nous venons de montrer que l'application $d : \gamma \mapsto \ell_1$ vérifie : $c \circ d = \text{Id}_{K_n}$; autrement dit, elle est injective. Sachant que $\text{card } \mathfrak{S}_n = n! = \text{card } K_n$, il s'agit en fait d'une bijection, et $c = d^{-1}$.

Question 7.

7.a On écrit :

```
let rec insère = fun
  | a 0 -> a::l
  | [] a _ -> failwith "insère"
  | (t::q) a j -> t::(insère q a (j-1)) ;;
```

7.b Nous allons utiliser une fonction auxiliaire aux k qui calcule la liste ℓ_{n-k} lorsque $k \in \llbracket 0, n-1 \rrbracket$.

```
let génère c =
  let n = vect_length c in
  let rec aux = fonction
    0 -> [n]
    | k -> let l = aux (k-1) in insère l (n-k) c.(n-k-1)
  in aux (n-1) ;;
```

(c_{n-k} est stocké dans la case $c.(n-k-1)$ car les vecteurs sont indexés à partir de 0).

Question 8.

8.a Posons $S_j = \sum_{k=1}^j kk!$. Alors $\sum_{k=1}^j (k+1)! = \sum_{k=1}^j (k+1)k! = S_j + \sum_{k=1}^j k! \iff S_j = \sum_{k=2}^{j+1} k! - \sum_{k=1}^j k! = (j+1)! - 1$.

8.b Si $p \in \llbracket 0, n-1 \rrbracket$, il existe un unique entier $j \in \llbracket 1, n-1 \rrbracket$ tel que $j! \leq p \leq (j+1)! - 1$. Une condition nécessaire d'existence de la décomposition est que $d_{j+1} = \dots = d_n = 0$ (car $p < (j+1)!$). Par ailleurs, si une telle décomposition existe, on a : $p = d_j j! + \sum_{k=1}^{j-1} d_k k!$, et la question précédente montre que

$$0 \leq \sum_{k=1}^{j-1} d_k k! \leq j! - 1. \text{ Ceci montre que } d_j \text{ est le quotient de la division euclidienne de } p \text{ par } j!, \text{ soit } d_j = \left\lfloor \frac{p}{j!} \right\rfloor.$$

Montrons alors l'existence et l'unicité de la décomposition en raisonnant par récurrence sur p .

– Si $p = 0$, on a clairement $d_1 = \dots = d_{n-1} = 0$.

– Si $p > 0$, supposons le résultat acquis jusqu'au rang $p-1$. Par hypothèse de récurrence, $p - \left\lfloor \frac{p}{j!} \right\rfloor j!$

admet une unique décomposition $\sum_{k=0}^{j-1} d_k k!$, et en posant $d_j = \left\lfloor \frac{p}{j!} \right\rfloor$, on obtient bien $p = \sum_{k=1}^j d_k k!$, avec

$$d_j < \frac{(j+1)!}{j!} = j+1.$$

Question 9.

Les questions précédentes montrent que l'application $f : \left(\begin{array}{l} \llbracket 0, n-1 \rrbracket \longrightarrow \mathfrak{S}_n \\ p \longmapsto c^{-1}(d_{n-1}, \dots, d_1, 0) \end{array} \right)$ réalise une bijection.

Nous allons utiliser une fonction auxiliaire aux k qui calcule la liste $(f(k-1), \dots, f(1), f(0))$, ce qui se fait naturellement de manière récursive.

```

let permutations n =
  let rec aux = function
    0 -> []
  | k -> let q = aux (k-1)          (* q est la liste (f(k-2),...f(0)) *)
          and t = g n re (lehmer n (k-1))  (* t est  gal   f(k-1) *)
          in t::q
  in aux (factorielle n) ;;

```

Remarque. Voici une d finition possible de la fonction de Lehmer (issue de ce qui pr c de) :

```

let rec factorielle = function
  0 -> 1
| n -> n * factorielle (n-1) ;;

let rec calcul_de_j = function
  1 -> 1
| p -> let k = calcul_de_j (p-1) in
        if p < factorielle (k+1) then k else k+1 ;;

let rec lehmer n = function
  0 -> make_vect n 0
| p -> let j = calcul_de_j p in
        let d = p / (factorielle j) in
        let c = lehmer n (p - d * (factorielle j)) in
        c.(n-1-j) <- d ;
        c ;;

```

