Algorithmes combinatoires et listes

Objectifs

— Fonctions combinatoires sur les listes.

Réalisations attendues

— A rendre : Exercice 2
— Indispensable : Exercices 1 à 4
— Bonus : Exercice 5

Rappel : tester à l'aide de Utop

— `dune utop` – lance utop
— `open Tp3 ;;` – charge le module `Tp3` i.e les fonctions contenues dans le fichier `tp3.ml`
— `gray_code 3;;` – par exemple pour tester la fonction `gray_code`
— `exit 0;;` – pour quitter

Attention, pour ce TP les fonctions auxilaires avec accumulateurs sont interdites et l'utilisation des itérateurs obligatoire quand possible.

Les algorithmes liés aux problèmes combinatoires, comme ceux utilisés dans la génération des permutations, sont très utiles et largement répandus en informatique, et constituent un sujet d'étude incontournable.

1 Codes binaires de Gray

Un code binaire de Gray à n bits est une énumération des 2^n séquences de n bits telle que deux séquences consécutives diffèrent seulement de 1 bit. Par exemple, voici un code de Gray pour 3 bits : `000, 001, 011, 010, 110, 111, 101, 100`

Un algorithme simple pour engendrer un code de Gray à n bits consiste à engendrer un code à $n-1$ bits $S_0, \ldots, S_{2^{n-1}}$, puis à effectuer la concaténation suivante : $0S_0, \ldots, 0S_{2^{n-1}}, 1S_{2^{n-1}}, \ldots, 1S_0$

▷ **Exercice 1** *Ecrire une fonction qui prend en argument un entier positif n, et qui renvoie un code de Gray à n bits, selon l'algorithme ci-dessus, de type* `int -> int list list` .

2 Combinaisons d'une liste

On cherche à engendrer toutes les combinaisons de k éléments choisis dans une liste l. Une combinaison sera représentée par une liste de ses éléments. Chaque combinaison devra respecter l'ordre des éléments tels qu'ils apparaissent dans l.

▷ **Exercice 2** *Ecrire une fonction qui prend en argument une liste l et un entier positif k, et qui renvoie la liste de toutes les combinaisons possibles de k éléments dans l. Par exemple :*
`combinaisons [1;2;3;4] 3 = [[1;2;3]; [1;2;4]; [1;3;4]; [2;3;4]]`

3 Permutations d'une liste

On représentera les permutations par des listes. Pour engendrer les permutations d'un ensemble, une décomposition récursive possible est de choisir un élément quelconque, d'engendrer les permutations sur les éléments restants, puis d'insérer l'élément manquant à toutes les positions possibles.

▷ **Exercice 3** *Ecrire la fonction qui prend en argument un élément e et une liste l, et qui renvoie la liste des insertions à toutes les positions possibles de e dans l. Par exemple :*
`insertions_partout 0 [1; 2] = [[0;1;2]; [1;0;2]; [1;2;0]]`

▷ **Exercice 4** *A l'aide de la fonction précédente, écrire une fonction qui prend en argument une liste l, et qui renvoie la liste des permutations de l.*

4 Partitions d'un entier

On s'intéresse ici aux différentes façons de décomposer un entier strictement positif en sommes de termes positifs rangés par ordre croissant pour éviter les redondances. Ainsi voici les différentes décompositions de 4 :

$$4 = 1+1+1+1 = 1+1+2 = 1+3 = 2+2$$

On représentera une partition donnée par une liste d'entiers croissants. *A priori*, tous les entiers inférieurs à un entier n à décomposer peuvent intervenir dans une partition. Une idée est donc de considérer tous ces termes possibles par ordre croissant, de 1 jusqu'à n.

Ainsi, les partitions de n pour un terme t donné peuvent se décomposer ainsi :

- t est pris en compte, et on exprime n en fonction de t et des partitions de $n-t$;
- t est ignoré, on exprime alors n en fonction du terme suivant $t+1$.

▷ **Exercice 5** *Ecrire une fonction qui prend en argument un entier strictement positif n et qui renvoie la liste de toutes les partitions possibles de n. Par exemple :*
`partitions 4 = [[1;1;1;1]; [1;1;2]; [1;3]; [2;2]; [4]]`