TP 1

MANIPULATION DE LISTES

Exercice 1

Pour tout ensemble non vide, identifié ici à une liste d'éléments deux à deux distincts, $E = [e_0, \ldots, e_{n-1}]$ de taille n, chacune de ses parties A peut être codée sous forme d'une liste `C` à n éléments contenant des zéros et des uns :

`C[i]` = 1 si $e_i \in A$, et `C[i]` = 0 sinon.

Par exemple, les parties $\emptyset$, $\{a\}$, $\{b\}$ et $\{a,b\}$ de l'ensemble $\{a,b\}$ sont respectivement codées par les listes $[0,0]$, $[1,0]$, $[0,1]$ et $[1,1]$.

1. Écrire une fonction `cardinal` d'argument un code `C` caractérisant une partie A et renvoyant le cardinal de A.

2. Écrire une fonction `intersection` de deux arguments `C1` et `C2`, les codes caractérisant des parties A_1 et A_2 d'un même ensemble à n éléments, et renvoyant le code de $A_1 \cap A_2$.

3. Écrire de même une fonction `difference` de deux arguments `C1` et `C2` et qui renvoie le code de $A_1 \setminus A_2$, l'ensemble des éléments qui sont dans A_1 et pas dans A_2.

4. ***a.*** Écrire une fonction `decoder` de deux arguments E et C renvoyant la partie de la liste E codée par le code C.
Par exemple `decoder([2,3,5,7],[1,0,0,1])` donne `[2,7]` et `decoder([2,3,5,7],[0,0,0,0])` donne `[]`.

b. Écrire une fonction `coder` de deux arguments E et A renvoyant le code de la partie A de la liste E.
Par exemple `coder([2,3,5,7],[2,7])` donne `[1,0,0,1]`.

c. Écrire une fonction `incrementer` d'argument une liste C de zéros et de uns de taille n, représentant sur n bits l'écriture en base 2 d'un entier naturel k, et renvoyant la liste représentant sur n bits l'écriture en base 2 de l'entier $(k+1)$.
Par exemple, `[0,1,0,1,1,1]` est l'écriture en base 2 sur 6 bits de 23 et `[0,1,1,0,0,0]` est l'écriture en base 2 sur 6 bits de 24.

d. En déduire la fonction `parties` d'argument E renvoyant la liste des parties de la liste E.

e. Écrire une fonction `p_parties` de deux arguments, un ensemble E non vide de taille n et un entier naturel $p \leqslant n$, qui renvoie la liste des parties de E de taille p.

5. ***a.*** Écrire une fonction `reunion` de deux arguments `C1` et `C2`, les codes caractérisant des parties A_1 et A_2 d'un même ensemble à n éléments, et renvoyant le code de $A_1 \cup A_2$.

Quelle est la complexité de la fonction `reunion` ?

b. On dit qu'un ensemble $\{A_1, \ldots, A_m\}$ de parties d'un même ensemble E est un « *recouvrement* » de E si et seulement si $A_1 \cup A_2 \cup \cdots \cup A_m = E$.
Écrire une fonction `estRec` d'argument une liste non vide de codes, représentant des parties d'un même ensemble E, et renvoyant un booléen indiquant s'il s'agit d'un recouvrement de E, ou pas.

c. Un recouvrement $R = \{A_1, \ldots, A_m\}$ de E est dit « *minimal* » si pour tout $R' \subset R$ tel que $R' \neq R$, R' n'est pas un recouvrement de E. Écrire une fonction `estRecMin` d'argument une liste non vide de codes de parties d'un même ensemble E et renvoyant un booléen indiquant s'il s'agit d'un recouvrement minimal de E, ou pas.

d. On nomme « *partition de* E » un recouvrement de E dont tous les éléments sont non vides et deux à deux disjoints. Écrire une fonction `estPartition` d'argument une liste non vide de codes de parties d'un même ensemble E et renvoyant un booléen indiquant s'il s'agit d'une partition de E, ou pas.

e. Écrire une fonction `partition` d'argument une liste C non vide quelconque de codes de parties d'un même ensemble E et renvoyant une partition de E construite progressivement en « *rajoutant* » d'abord chaque élément de C dans l'ordre puis en complétant par la partie des éléments qui ne sont dans aucune partie de C.

Exercice 2

Dans cet exercice, un segment $[a, b]$ (avec $a \leqslant b$) est représenté par la liste : `[a, b]`.

1. Deux segments sont disjoints lorsque leur intersection est vide. Écrire une fonction `disjoints` de deux arguments `i1` et `i2` qui teste si les segments `i1` et `i2` sont disjoints en renvoyant le booléen `True` s'ils sont disjoints, `False` sinon.

2. La fusion de deux segments a comme minimum le plus petit des minimums des deux segments, et comme maximum le plus grand des maximums des deux segments. Écrire une fonction `fusion` de deux arguments `i1` et `i2` et qui renvoie le segment correspondant à la fusion de `i1` et `i2`.

3. Une « *liste bien formée* » est une liste de segments qui vérifie les propriétés suivantes :

- les segments sont deux à deux disjoints ;
- les segments de la liste sont classés par ordre croissant, en considérant qu'un segment `i1` est strictement plus petit qu'un segment `i2` si et seulement si le maximum de `i1` est strictement inférieur au minimum de `i2`.

a. Les listes suivantes sont-elles bien formées ?

L1 = [[0, 1], [2, 5], [3, 6]], L2 = [[2, 5], [0, 1], [3, 6]], L3 = [[0, 1], [2, 3], [4, 6]].

b. Écrire une fonction `verifie` qui teste si une liste de segments est bien formée (on pourra proposer une version itérative et une version récursive).

4. Écrire une fonction `appartient` de deux arguments `x` et `L` qui teste si la valeur `x` est élément d'un des segments de la liste `L` bien formée.

Exercice 3

Dans cet exercice, on manipule des listes d'entiers et on dira que la liste est «croissante», «décroissante», «monotone» si c'est le cas de la suite d'entiers sous-jacente.

1. Écrire une fonction, de complexité linéaire dans le pire des cas, `estCroissante` d'argument une liste d'entiers et qui renvoie un booléen indiquant si cette liste est croissante.

Écrire de même des fonctions `estDecroissante` et `estMonotone`.

2. Soit la liste $L = [u_0, u_1, \cdots, u_{n-1}]$ de longueur n. On appelle tranche de `L` une liste de la forme $[u_i, u_{i+1}, \cdots, u_j]$ où $0 \leqslant i \leqslant j < n$. Écrire une fonction `maxCroissante` d'argument une liste `L` qui renvoie la plus longue tranche croissante de `L`. S'il n'y a pas unicité, on renvoie la première trouvée.

3. Soit la liste $L = [u_0, u_1, \cdots, u_{n-1}]$ de longueur n. Une *monotonie* de `L` est un couple d'indices (i, j) où $0 \leqslant i < j < n$ tel que la sous-liste $[u_i, u_{i+1}, \cdots, u_j]$ soit monotone et qu'elle ne le soit plus si on l'étend, à droite ou à gauche, d'un élément supplémentaire (lorsque c'est possible). La monotonie est dite « *banale* » lorsque $j = i + 1$.

a. Proposez une liste d'entiers de longueur 5 qui ne présente que des monotonies banales. Peut-on avoir deux termes consécutifs égaux dans une liste ne présentant que des monotonies banales ?

b. Écrire une fonction `cahots`, de complexité linéaire, qui teste si une liste ne comporte que des monotonies banales.

c. Après avoir créé une liste arbitraire `L` de valeurs distinctes, on peut l'ordonner par `L.sort()`. Imaginer ensuite une méthode pour réordonner `L` de manière à ce qu'elle ne comporte que des monotonies banales.