TP 1

MANIPULATION DE LISTES

Exercice 1

On appelle « *nombre de Hamming* » tout entier naturel n de la forme $2^a 3^b 5^c$ avec $(a,b,c) \in \mathbb{N}^3$. Voici la liste des 15 premiers nombres de Hamming :

$$1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24.$$

1. Écrire une fonction `H` d'argument n qui renvoie un booléen indiquant si l'entier naturel n est un nombre de Hamming.

2. En utilisant la fonction précédente, écrire une fonction `LH1` d'argument `N` renvoyant la liste des `N` premiers nombres de Hamming.

3. On peut procéder différemment en remarquant que si `t` est la liste des j premiers nombres de Hamming alors le $(j+1)$-ème nombre est nécessairement l'un des termes de `t` multiplié par deux, trois ou cinq.

Écrire une fonction `LH2` ayant les mêmes caractéristiques que `LH1` mais utilisant cette remarque.

4. Expliquer et observer ce que fait le programme suivant :

```
def g(L2, L3, L5) :
    n = min(L2[0], L3[0], L5[0])
    if n == L2[0]:
        L2.pop(0)
    if n == L3[0]:
        L3.pop(0)
    if n == L5[0]:
        L5.pop(0)
    L2.append(2*n)
    L3.append(3*n)
    L5.append(5*n)
    return n

L2, L3, L5 = [1], [1], [1]
LH = []
for no in range(15) :
    LH.append(g(L2, L3, L5))
```

5. À l'aide de l'instruction `time` du module `time` (par exemple), comparer la durée de calcul pour obtenir le plus grand nombre de Hamming strictement inférieur à 10^6 par chacune des trois méthodes.

Exercice 2

Pour tout ensemble non vide, identifié ici à une liste d'éléments deux à deux distincts, $E = [e_0, \dots, e_{n-1}]$ 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 $\{A_1, \dots, A_m\}$ de parties d'un même ensemble E est un « *recouvrement* » de E si et seulement si $A_1 \cup A_2 \cup \dots \cup A_m = E$.
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 $R = \{A_1, \dots, A_m\}$ 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.