Travaux pratique énoncé 3 : énumération

Similairement au sujet précédent, reprendre les documents

anneau01.py

relation.py

En effet, on revient sur les relation non pondérées.
Assurez vous que les fonctions *testSymetrie* et *testreflexive* du second sujet de TP fonctionnent correctement avec l'anneau booléen.

Méthode d'énumération

Pour l'instant, on utilise une méthode "brute force" : on génère toutes les relations sur un ensemble de taille donnée et on teste la ou les propriétés voulues sur chaque.

Quelle définition des relations semble plus facile à utiliser ?

Sur une matrice, on numérote les cases dans l'ordre de lecture, haut vers bas en ordre principal et gauche vers droite ensuite

0	1	2	3
4	5	6	7
8	9	10	11

Écrire une fonction *editeCase* qui prend en entrée une relation, un indice entier et une valeur 0 ou 1, et qui modifie la case correspondante de la relation:
Si R est de taille 3×4,
$editeCase(R, 7, 0)$ effectue $R[1][3] = 0$
$editeCase(R, 0, 1)$ effectue $R[0][0] = 1$
$editeCase(R, 79, 0)$ effectue $R[2][1] = 0$

Créer une boucle qui prend en entrée une relation de coefficients tous 0 et qui la transforme successivement en toute les relations, dans l'ordre binaire utilisant la case 0 comme bit de poids faible.
(ie la case 0 alterne entre 0 et 1 à chaque étape, la case 1 change lorsque la case 0 revient à 0, ... la case i change lorsque la case i-1 revient à 0, la boucle stoppe lorsque la relation est redevenue la relation 0)

Application

Utiliser la boucle précédente et la fonction $testreflexive$ pour écrire une fonction qui compter le nombre de relations réflexives, l'utiliser pour les relations sur un ensemble de taille 3.

Utiliser cette boucle et la fonction *testsymetrie* pour écrire une fonction qui compter le nombre de relations symétriques, et compter ce nombre sur un ensemble de taille 3.

Écrire une fonction *testTransitive* qui prend en entrée une relation et retourne un booléen indiquant si la relation est transitive.

Utiliser la boucle et la fonction précédente pour compter le nombre de relation transitive sur un ensemble de taille 3.

Utiliser la boucles et les trois fonctions pour compter le nombre de relation d'équivalences sur un ensemble à 3 éléments

Écrire une fonction *testAntiSymetrique* qui, vous l'avez deviné, prend en entrée une relation et retourne un booléen indiquant si elle est antisymétrique.

Dénombrer les relations d'ordre sur un ensemble à 3 éléments. (*Une relation d'ordre est une relation réflexive antisymétrique et transitive*).

Dénombrer les relations qui ne sont ni d'ordre, ni d'équivalence. Que remarquez vous ? Comment l'expliquez vous ?

Accélération (bonus)

Première optimisation, parallélisme Écrire une fonction qui prend en argument un entier n et retourne une liste contenant le nombre de relation d'équivalence sur un ensemble de taille n et le nombre de relations d'ordres. On ne construira chaque relation qu'une seule fois.

Gray

Combien de fois la fonction édite est elle appelée pour créer chaque nouvelle relation ? Combien cela fait il d'appel en tout ?

Il est facile de voir qu'il faut au moins un appel par relation construite, cela nous donne une borne inférieure sur le nombre d'appels.

On construit le graphe suivant :

- *Chaque nombre entre* 0 *et* $2^{n-1}-1$ *inclus, écrits en binaire est un sommet.*
- *On place une arête orienté entre deux sommets si les* $n-2$ *derniers chiffres (binaires) du premier (tous sauf le premier) sont aussi les* $n-2$ *premiers chiffres du second. On annote cette arête avec les n chiffres (1er du premier, chiffres en commun, dernier du second)*

  les arêtes sont les numéros de 0 à $2^n - 1$

Combien d'arêtes sortantes ont chaque sommet, combien d'arêtes entrantes ?

Si vous avez vu les graphes Hamiltoniens, vous reconnaissez la condition permettant d'en construire un, prouvant qu'il est possible de n'appeler edite() qu'une seule fois par relation.
La numérotation de Gray est un tel graphe.

0	=	00000	8	=	01100	16	=	11000	24	=	10100
1	=	00001	9	=	01101	17	=	11001	25	=	10101
2	=	00011	10	=	01111	18	=	11011	26	=	10111
3	=	00010	11	=	01110	19	=	11010	27	=	10110
4	=	00110	12	=	01010	20	=	11110	28	=	10010
5	=	00111	13	=	01011	21	=	11111	29	=	10011
6	=	00101	14	=	01001	22	=	11101	30	=	10001
7	=	00100	15	=	01000	23	=	11100	31	=	10000

pour effectuer l'opération $Gray(i) \rightarrow Gray(i+1)$, trouver le plus grand j tel que 2^j divise i, et inverser le bit de poids 2^i