TP n° 05 bis – Boucles et complexité

I Vérification d'un carré magique

Un carré magique est un arrangement de nombres différents, généralement entiers, dans un tableau dont la somme des éléments sur les lignes, les colonnes et les diagonales sont identiques. On note n la taille des lignes et colonnes, et on parle de carré magique d'ordre n. Le carré magique est dit « normal » s'il est constitué exactement de tous les nombres entiers de 1 à n^2.

(a) Carré magique normal d'ordre 3.

2	7	6
9	5	1
4	3	8

(b) Carré magique d'ordre 3.

23	28	21
22	24	26
27	20	25

(c) Carré magique normal d'ordre 4.

4	14	15	1
9	7	6	12
5	11	10	8
16	2	3	13

Table 1 – Exemples de carrés magiques.

1. Vérifier à la main les carrés magiques de la table 1.

Les carrés magiques seront stockés dans des listes en python. Chaque élément de la liste est lui-même une liste contenant une ligne du carré magique de telle sorte que la commande `T[i][j]` permettra d'accéder à l'élément de la ligne i et de colonne j du tableau T. Pour le premier carré magique on a donc : `T = [ [2,7,6] , [9,5,1] , [4,3,8] ]`.
On se propose d'écrire les fonctions permettant de vérifier qu'un tableau passé en argument est bien un carré magique. Si nécessaire, des indications sont proposées en Annexes.

2. Dans un premier temps, la fonction de vérification doit calculer la somme de chaque ligne, de chaque colonne et de chaque diagonale en stockant le résultat dans une liste, puis on effectue la vérification du carré magique. Écrire une fonction `verif_carre_1(T)` qui vérifie si le tableau `T` correspond à un carré magique ou non. La fonction renverra `True` si la carré est magique et `False` sinon.
3. Déterminer la complexité de votre fonction en ne comptant que le nombre d'additions et de comparaisons réalisées dans le meilleur et dans le pire des cas en fonction de n.
4. Cette fonction est-elle optimale ? Proposer une nouvelle fonction `verif_carre_2(T)` pour améliorer les performances et donner la complexité dans le meilleur et dans le pire des cas en fonction de n.

II Calendrier grégorien

La manipulation des dates dans les logiciels de gestion, ou encore sur les sites web de réservation, doit s'effectuer conformément au calendrier grégorien (entré en vigueur à la fin du XVIe siècle en France) en précisant pour chaque date le jour de la semaine. Or le calendrier grégorien est un format assez éloigné des formats habituels de stockage des nombres dans un ordinateur.
On cherche à déterminer, pour une date donnée, sa position dans l'énumération des jours depuis le 1er janvier 1600. et le jour de la semaine correspondant.

1. Proposer une fonction `nombre_de_jours(jours,mois,annee)` qui prend en entrée une date de la forme `jours,mois,annee` postérieure au 1,1,1600 et renvoie le nombre de jours écoulés entre le 1er janvier 1600 et la date considérée, sans tenir compte des années bissextiles (c'est-à-dire en supposant que chaque année comporte 365 jours). On pourra utiliser une liste des nombres de jours de chaque mois pour une année non bissextile : `m = [31,28,31,30,31,30,31,31,30,31,30,31]`.
2. Les années bissextiles sont déterminées par la règle suivante [1] :
   — Si l'année est divisible par 4 et non-divisible par 100, c'est une année bissextile (elle a 366 jours).
   — Si l'année est divisible par 400, c'est une année bissextile (elle a 366 jours).
   — Sinon, l'année n'est pas bissextile (elle a 365 jours).
   Proposer une fonction `bissextile(annee)` renvoyant `True` si l'année est bissextile et `False` sinon.
3. Modifier le programme de la fonction `nombre_de_jours(jours,mois,annee)` pour tenir compte des années bissextiles. Par exemple, `nombre_de_jours(1,2,1600)` doit retourner la valeur 31, `nombre_de_jours(1,1,1604)` la valeur 1461 (366+365+365+365) puisque l'année 1600 est bissextile.
4. Le 1er janvier 2001 était un lundi. Déterminer, en utilisant la fonction de la question précédente, quel jour de la semaine tombera le 1er mai 2040. Quel jour de la semaine est tombé le 14 juillet 1789 ?

1. Wikipédia en français, « Année bissextile », consulté le 23 novembre 2015.