Lycée Fénelon Sainte-Marie		MP
Année 2017-2018		Informatique Pour Tous
Mercredi 10 janvier 2018	Durée : 2 heures	Concours blanc

Toute calculatrice autorisée.
Le sujet comporte un total de 3 exercices
que vous pouvez traiter dans l'ordre de votre choix.

Les codes demandés devront être **clairement** commentés !

EXERCICE 1. Des carrés… magiques. 15 points

Introduction

Un carré magique d'ordre *n* est un tableau carré comportant n^2 cellules que l'on numérote à l'aide des entiers de 1 à n^2 de telle sorte que les sommes des entiers de chaque ligne, chaque colonne et des deux diagonales principales soient égales. La somme commune est appelée *somme magique*.

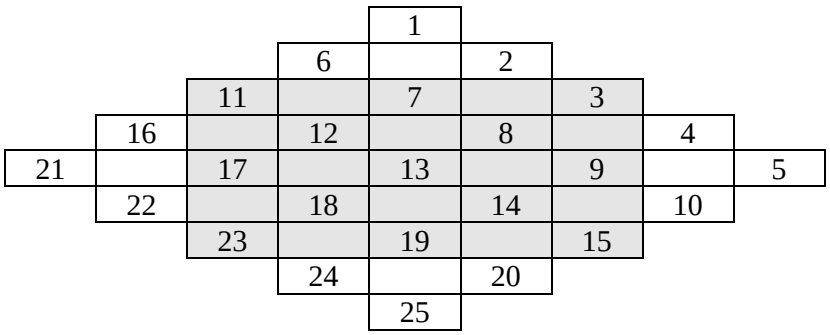
Voici un exemple de carré magique d'ordre 5 :

11	24	7	20	3
4	12	25	8	16
17	5	13	21	9
10	18	1	14	22
23	6	19	2	15

Vous n'aurez pas de difficulté à vérifier que la somme de chaque ligne, chaque colonne et des deux diagonales vaut ici : 65.

Le mathématicien et grammairien français Bachet de Méziriac (1581-1638) fut le premier à proposer en français une méthode générale, dite *méthode de Bachet*, de construction d'un carré magique d'ordre impair (mais cette méthode était connue depuis longtemps…). En voici une illustration dans le style « histoire sans paroles » :

1ère étape :



2ème étape :

			1					
			6		**2**			
		11		7		3		
	16		12		8		4	
21		17		13		9		5
	22		18	**1**	14		10	
		23	**6**	19	**2**	15		
			24		20			
				25				

				1				
			6		2			
		11	**24**	7	**20**	3		
	16	**4**	12	**25**	8	**16**	4	
21		17	**15**	13	**21**	9		5
	22	**10**	18	**1**	14	**22**	10	
		23	**6**	19	**2**	15		
			24		20			
				25				

3ème étape :

11	24	7	20	3
4	12	25	8	16
17	15	13	21	9
10	18	1	14	22
23	6	19	2	15

Dans tout ce qui suit, les carrés magiques seront représentés par des tableaux `numpy`.

Partie A : pour se mettre en jambe…

[Q.1 / 2 points] Démontrer que, pour un carré magique d'ordre *n*, la somme commune des lignes, des colonnes et de deux diagonales principales est égale à $\frac{n\left(n^2+1\right)}{2}$.

[Q.2 / 2 points] Donner <u>une</u> instruction permettant de construire la liste des entiers naturels compris entre 1 et n^2 (on supposera que la variable `n` aura été définie par ailleurs).

[Q.3 / 3 points] Ecrire une fonction Python `MagicSquareValid` qui reçoit en argument un tableau numpy `M` d'entiers et renverra un booléen valant `True` ou `False` suivant que `M` représente ou non un carré magique.

La fonction vérifiera, dans un premier temps, que le tableau comporte autant de lignes que de colonnes et que ses éléments correspondent bien aux entiers de l'intervalle $[\![1; n^2]\!]$ (on utilisera le résultat de la question précédente).

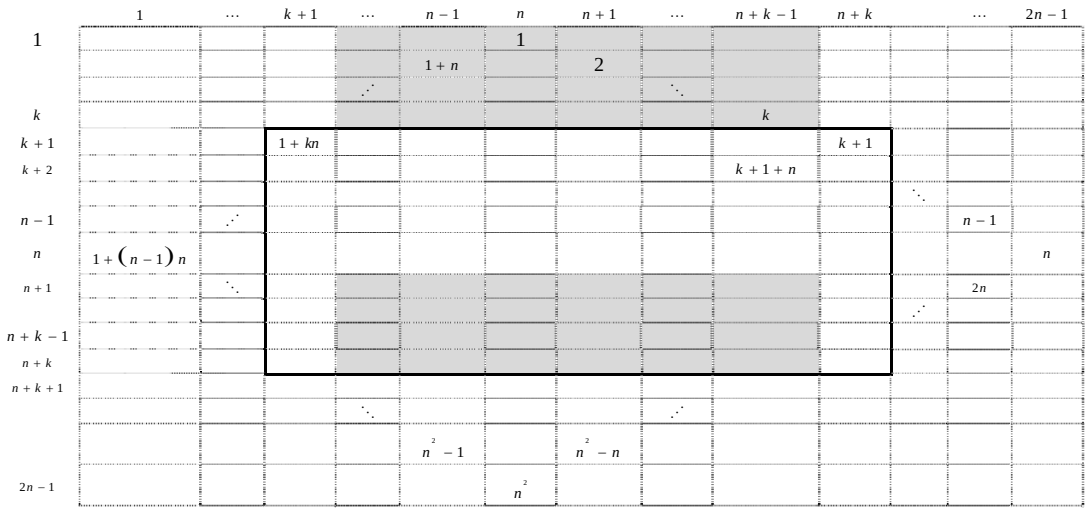
Partie B : Mise en œuvre de la méthode de Bachet

On va écrire une fonction Python `OddMagicSquares` qui recevra en argument un entier naturel impair $n = 2k+1$ où n est l'ordre du carré magique à créer.

Les questions ci-dessous visent à construire cette fonction par étape en généralisant l'histoire sans parole présentée plus haut dans l'introduction...

1ère étape : construction d'un tableau `numpy` rempli de zéros.

[Q.4 / 1 point] Construire un tableau `numpy M` rempli de zéros (on choisira le type `int`) et correspondant à une matrice carrée d'ordre $2n-1 = 4k+1$ (cf. le tableau ci-dessous).



2ème étape : remplissage des *n* sous-diagonales.

Dans le tableau précédent, on observe des sous-diagonales parallèles à la diagonale principale, comportant exactement *n* coefficients et dont les valeurs des premiers coefficients sont : 1, $1+n$, $1+2n$, …, $1+(n-1)n$. On a donc, comme valeurs des coefficients :

- Pour la 1ère sous-diagonale : 1, 2, 3, …, *n*
- Pour la 2ème sous-diagonale : $1+n$, $2+n$, $3+n$, …, $2n$
- …
- Pour la *n*ième sous-diagonale : $1+(n-1)n$, $2+(n-1)n$, $3+(n-1)n$, …, $n+(n-1)n=n^2$

[Q.5 / 3 points] Remplir les *n* sous-diagonales du tableau `M` selon le remplissage décrit ci-dessus.

3ème étape : recopie des sous-tableaux « extérieurs »

Une fois les sous-diagonales remplies, on doit recopier les quatre sous-tableaux « extérieurs ». Par exemple, le sous-tableau grisé de la figure correspondant aux indices de ligne 1 à *k* et aux indices de colonnes $k+2$ à $n+k-1=k+2+(n-3)$, est recopié dans la zone de `M` correspondant aux indices de ligne $n+1$ à $n+k$ et aux indices de colonnes d'indices $k+2$ à $n+k-1$.

On aura, par exemple, les instructions suivantes :

```
k = n // 2
# --> bloc du haut (va en bas dans le carré magique)<-- #
M[n:3 * k + 1,k + 1:3 * k] += M[:k,k + 1:3 * k]
```

[Q.6 / 3 points] Donner trois instructions permettant d'effectuer les recopies des zones du bas (vers le haut), de la gauche (vers la droite) et de la droite (vers la gauche).

Une fois ces recopies effectuées , il ne reste plus qu'à « extraire » le carré magique du tableau `M`.

[Q.7 / 1 point] Extraire puis imprimer le sous-tableau de M correspondant aux indices de ligne et de colonne compris entre $k+1$ et $n+k$.

EXERCICE 2. La fonction d'Ackermann — 16 points

La fonction d'Ackermann a été initialement introduite en 1928 comme une fonction de trois variables entières naturelles. En fixant la première variable à la valeur 2, Peter a obtenu une fonction de deux variables entières naturelles classiquement appelée « fonction d'Ackermann » et définie récursivement comme suit :

$$A:(m,n)\mapsto\begin{cases}n+1 & \text{si } m=0\\ A(m-1,1) & \text{si } m>0 \text{ et } n=0\\ A\big(m-1,A(m,n-1)\big) & \text{si } m>0 \text{ et } n>0\end{cases}$$

[Q.1 / 4 points] Ecrire une fonction Python `AP` qui reçoit en argument deux variables entières `m` et `n` et renvoie $A(m,n)$ en implémentant la définition récursive ci-dessus.
Votre fonction devra d'abord vérifier que `m` et `n` sont bien entiers et positifs. Dans le cas contraire, elle devra renvoyer un message d'erreur approprié.

Pour tout entier naturel *n*, on a évidemment : $A(0,n)=n+1$.

[Q.2 / 1 point] Vérifier que l'on a : $A(1,0)=2$.

[Q.3 / 3 point] Démontrer par récurrence que l'on a : $\forall n\in\mathbb{N},\ A(1,n)=2+(n+3)-3$.

L'expression de $A(1,n)$ ci-dessus est volontairement non simplifiée car on a, pour tout *n* entier naturel :

$$A(2,n)=2\times(n+3)-3$$
$$A(3,n)=2^{n+3}-3$$
$$A(4,n)=\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+3\text{ "2" empilés}}-3$$

Ainsi :

$$A(3,4)=2^{4+3}-3=2^{7}-3=128-3=125$$
$$A(4,1)=2^{2^{2^{2}}}-3=2^{2^{4}}-3=2^{16}-3=65536-3=65533$$

Rappelons que les exponentiations successives se calculent de la droite vers la gauche.

Ainsi, malgré la simplicité calculatoire apparente de la fonction d'Ackermann, les valeurs prises augmentent (vraiment) très vite…

On peut cependant continuer de donner des expressions « simples » pour $A(5,n)$, $A(6,n)$, … en utilisant la notation des puissances itérées de Knuth.

On note par exemple : $2^{n+3} = 2 \uparrow (n+3)$ et $\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+3 \text{ "2" empilés}} = 2 \uparrow\uparrow (n+3)$.

On montre alors que l'on a : $A(5,n) = 2 \uparrow\uparrow\uparrow (n+3) - 3 = \underbrace{2 \uparrow\uparrow \left(2 \uparrow\uparrow \left(2 \uparrow\uparrow \left(\ldots \uparrow\uparrow 2\right)\right)\right)}_{\text{"2" apparaît } n+3 \text{ fois}} - 3$

Par exemple : $A(5,0) = 2 \uparrow\uparrow\uparrow 3 - 3 = 2 \uparrow\uparrow 2 \uparrow\uparrow 2 - 3 = 2 \uparrow\uparrow \left(2 \uparrow\uparrow 2\right) - 3$.

Or : $2 \uparrow\uparrow 2 = 2^2 = 4$. Donc $A(5,0) = 2 \uparrow\uparrow \left(2 \uparrow\uparrow 2\right) - 3 = 2 \uparrow\uparrow 4 - 3 = 2^{2^{2^2}} - 3 = 65533$.

On vérifie en passant que l'on a retrouvé $A(4,1)$.

La notation des puissances itérées de Knuth est ainsi fondamentalement récursive :

$$a \underbrace{\uparrow\uparrow\uparrow \ldots \uparrow\uparrow}_{\text{la flèche } \uparrow \text{ apparaît } n \text{ fois}} b = \underbrace{a \underbrace{\uparrow\uparrow\uparrow \ldots \uparrow\uparrow}_{\text{la flèche } \uparrow \text{ apparaît } n-1 \text{ fois}} a \underbrace{\uparrow\uparrow\uparrow \ldots \uparrow\uparrow}_{\text{la flèche } \uparrow \text{ apparaît } n-1 \text{ fois}} a \underbrace{\uparrow\uparrow\uparrow \ldots \uparrow\uparrow}_{\text{la flèche } \uparrow \text{ apparaît } n-1 \text{ fois}} a \ldots \underbrace{\uparrow\uparrow\uparrow \ldots \uparrow\uparrow}_{\text{la flèche } \uparrow \text{ apparaît } n-1 \text{ fois}} a}_{\text{"}a\text{" apparaît } b \text{ fois}}$$

En utilisant enfin la notation condensée : $\underbrace{\uparrow\uparrow\uparrow \ldots \uparrow\uparrow}_{\text{la flèche } \uparrow \text{ apparaît } k \text{ fois}} = \uparrow^k$, on peut définir la puissance itérée de Knuth de *a* et *b* au rang *n* par :

$$a \uparrow^n b = \begin{cases} a^b & \text{si } n = 1 \\ 1 & \text{si } b = 0 \\ a \uparrow^{n-1} \left(a \uparrow^n (b-1)\right) & \text{sinon} \end{cases}$$

[Q.4 / 4 points] Ecrire une fonction Python `KnuthPI` qui reçoit en argument trois variables entières `a`, `b` et `n` et renvoie $a \uparrow^n b$.
Votre fonction devra d'abord vérifier que `a`, `b` et `n` sont bien entiers avec `b` positif et `n` positif non nul. Dans le cas contraire, elle devra renvoyer un message d'erreur approprié.

A l'aide des notations précédentes, on montre que l'on a finalement, pour tout entier naturel *m* supérieur ou égal à 3 et tout entier naturel *n* :

$$A(m,n) = 2 \uparrow^{m-2} (n+3) - 3$$

[Q.5 / 4 points] Ecrire une fonction Python `AP2` qui reçoit en argument deux variables entières `m` et `n` et renvoie $A(m,n)$ en implémentant le résultat ci-dessus et en utilisant la fonction `KnuthPI` lorsque $m \geq 3$ (pour les autres cas, on utilisera les formules appropriées).
Votre fonction devra d'abord vérifier que `m` et `n` sont bien entiers et positifs. Dans le cas contraire, elle devra renvoyer un message d'erreur approprié.

EXERCICE 3. Des chaînes eulériennes — 15 points

I. Eléments historiques et définitions

Historiquement parlant, la théorie des graphes, est née avec le géant des mathématiques, Leonhard EULER (1707-1783), même s'il ne l'avait pas vraiment baptisée de la sorte à l'époque…

En des termes modernes, le problème qu'Euler résolut consistait à chercher une… « chaîne eulérienne » dans un graphe simple non orienté.

Définitions

On dit qu'une chaîne d'un graphe simple non orienté est « eulérienne » si elle passe une fois et une seule par chaque arête du graphe.

On dit d'un graphe qu'il s'agit d'un « graphe eulérien » s'il admet un cycle eulérien, c'est-à-dire une chaîne eulérienne qui est un cycle.

Euler a établi les deux théorèmes suivants :

Théorème d'existence d'une chaîne eulérienne.

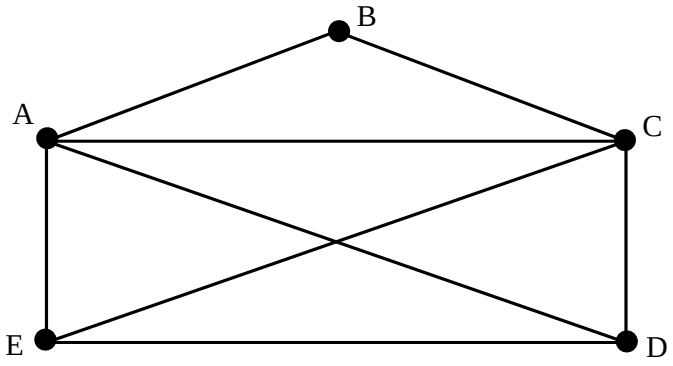
Un graphe simple non orienté admet une chaîne eulérienne si, et seulement si, le nombre de sommets de degré impair est égal 0 ou 2. Dans le deuxième cas, les sommets de degré impair sont les extrémités de toute chaîne eulérienne.

Théorème d'existence d'un cycle eulérien.

Un graphe simple non orienté admet un cycle eulérien si, et seulement si, le nombre de sommets de degré impair est égal 0.

II. L'enveloppe

On considère le graphe suivant :



[Q.1 / 2 points] Le graphe ci-dessus :

a) admet-il une chaîne eulérienne ? Si oui, en donner une.

b) admet-il un cycle eulérien ? Si oui, en donner un.

III. Existence

[Q.2 / 4 points] Ecrire une fonction Python `OddVertices` qui reçoit en argument un tableau `numpy A` correspondant à la matrice d'adjacence d'un graphe simple non orienté (on ne cherchera pas à valider qu'il s'agit bien d'une telle matrice) et renvoie la liste (éventuellement vide) des indices des sommets de degré impair.

[Q.3 / 3 points] Ecrire une fonction Python `CCEuler` qui reçoit en argument un tableau numpy `A` correspondant à la matrice d'adjacence d'un graphe simple non orienté (on ne cherchera pas à valider qu'il s'agit bien d'une telle matrice) et renvoie :

a) le booléen `False` si le graphe n'admet pas de chaîne eulérienne.

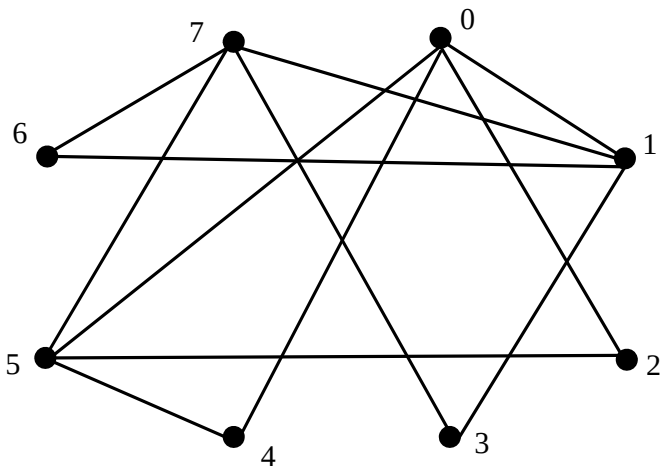
b) Le booléen `True` si le graphe admet un cycle eulérien.

c) Un tuple constitué du booléen `True` et des indices des deux sommets de degré impair si le graphe admet exactement deux sommets de degré impair.

La fonction `CCEuler` utilisera la fonction `OddVertices` de la question précédente.

IV. Recherche

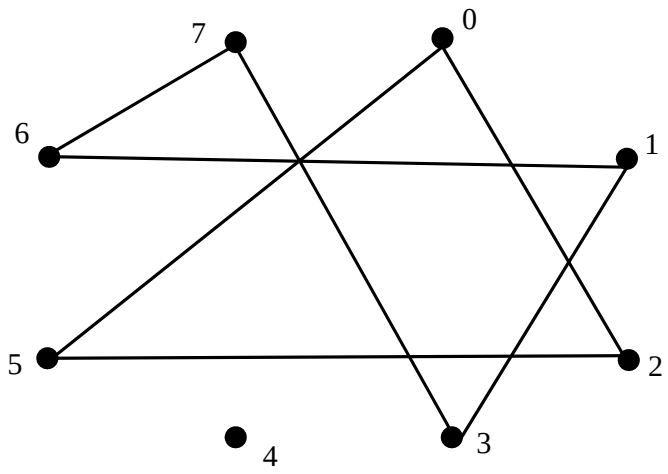
Pour décrire un algorithme permettant d'obtenir une chaîne ou un cycle eulérien, nous allons partir du graphe G d'ordre 8 suivant :



Tous les sommets étant de degré pair, le graphe G est eulérien.

1ère étape : on détermine une chaîne (un cycle ici) sans répétition d'arêtes

Par exemple : (4,5,7,1,0,4). On retire alors les arêtes correspondantes du graphe G et on obtient le graphe G' suivant :



Remarque importante : si nous avions eu affaire à un graphe admettant « seulement » une chaîne eulérienne mais pas de cycle eulérien, on aurait choisi pour extrémités de cette première chaîne, les deux sommets de degré impair.

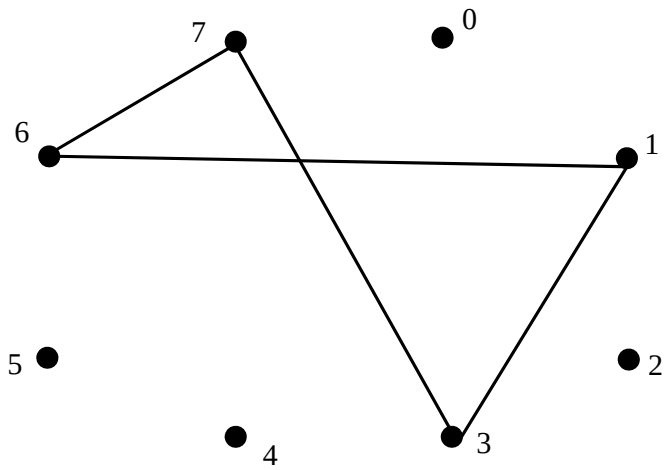
2ème étape : on choisit un sommet non isolé S de G' et on construit un cycle passant par S

On peut choisir ici le sommet 5 et on obtient le cycle (5,0,2,5).

3ème étape : on insère le cycle obtenu dans la chaîne initiale.

(4,5,7,1,0,4) devient (4,**5,0,2,5**,7,1,0,4).

4ème étape : on retire de G' les arêtes du cycle précédemment obtenu



On recommence alors les étapes 2, 3 et 4 tant que le graphe obtenu à l'étape 4 comporte encore des arêtes.

[Q.4 / 3 points] Ecrire une fonction Python `Insert` qui reçoit en argument deux listes d'entiers (représentant des listes d'indices de sommets d'un même graphe) `L` et `M`, `L` contenant le premier élément de `M`, et renvoyant une liste `N` obtenue en remplaçant dans `L` une occurrence de `M[0]` par les éléments de `M`.

Par exemple avec `L=[4,5,7,1,0,4]` et `M=[5,0,2,5]`, la fonction `Insert` devra renvoyer la liste :

`[4,5,0,2,5,7,1,0,4]`

[Q.5 / 3 points] Ecrire une fonction Python `Delete` qui reçoit en argument un tableau `numpy A` correspondant à la matrice d'adjacence d'un graphe simple non orienté et une liste `L` d'entiers représentant un cycle de ce graphe. La fonction renverra le tableau A mis à jour en mettant à zéro les coefficients correspondant aux arêtes du cycle.

La fonction devra vérifier que le premier élément et le dernier élément de `L` sont identiques.

FIN DU SUJET