NOM : **Prénom :** **Classe : PSI***

Devoir Surveillé 4

Les réponses sont à écrire exclusivement sur ce document

Les différentes questions sont indépendantes les unes des autres.

1 Tris récursifs

Question 1. Rappeler les complexités dans le meilleur et dans le pire des cas des trois algorithmes de tri au programme. Aucune justification n'est demandée.

Question 2. Écrire une fonction `interlac(L1,L2)` qui prend en argument deux listes de nombres `L1` et `L2`, triées par ordre croissant et qui renvoie la liste contenant tous les éléments de `L1` et de `L2` triés par ordre croissant.

Question 3. Écrire une fonction `fusion(L)` qui prend en argument une liste de nombres `L` et qui réalise le tri par fusion de `L`.

Question 4. Adapter votre fonction d'entrelacement pour écrire une fonction `interlacT(LT1,LT2)` pour qu'elle prenne en argument des listes de tuples où la première composante est un nombre, les autres composantes quelconques, et où `LT1` et `LT2` sont triés dans l'ordre croissant selon les premières composantes de chaque tuple et qui réalise l'entrelacement de `L1` et de `L2` de sorte à ce que la liste renvoyée soit triée par ordre croissant selon la première composante de chaque tuple. Quelles modifications éventuelles faudrait-il apporter à votre fonction `fusion` pour qu'elle permette le tri d'une liste de tuples selon l'ordre croissant de leurs premières composantes ?

2 Notation polonaise inverse

Dans ce exercice nous considèrerons disposer d'un type pile et des opérations suivantes :

Opération	Effet
`pile()`	Création d'une nouvelle pile (vide)
`p.pop()`	Retire l'élément au sommet de la pile `p` et renvoie sa valeur. Une erreur survient si la pile est vide.
`p.push(x)`	Ajoute l'élément `x` au sommet de la pile `p`.
`p.is_empty()`	Renvoie `True` si la pile est vide et `False` sinon.

Nous ne disposons d'aucune autre opération sur les piles.

La notation polonaise inverse ou notation postfixée consiste à placer les opérateurs *après* les opérandes, en cela il diffère de la notion infixe.

Voici quelques exemples :

Notation infixe	Notation postfixe
$1+2$	$1\ 2\ +$
$10 \div 0.5$	$10\ 0.5\ \div$
2×3	$2\ 3\ \times$
$(1+2) \times 3$	$1\ 2\ +\ 3\ \times$
$1+(2 \times 3)$	$1\ 2\ 3\ \times\ +$
$(-2)-3$	$2\ -_1\ 3\ -_2$
$-(2-3)$	$2\ 3\ -_2\ -_1$

Il n'y a pas besoin de parenthèses en notation postfixe. Par contre, il faut faut distinguer le $-$ unaire (noté ici $-_1$) du moins binaire (noté ici $-_2$). Le premier transforme un nombre en son opposé (exemple : -2) le second calcule la différence entre deux termes (exemple : $1-3$).

Question 5. Convertir les expressions suivantes en notation polonaise inverse.

1. $1+2+3+4+5$
2. $7+(1+3)\times(2+3)$
3. $2\times 3\times 5\div 7$
4. $3+9\times(5\times(7+2)+2)$

Question 6. Convertir en notation usuelle les expressions postfixes suivantes.

1. 2 3 $-_1$ $-_2$
2. 1 2 3 + $\times$
3. 1 2 3 + $-_1$ $\times$
4. 7 5 3 2 1 + $\times$ $\div$ +

Question 7. On considère des listes de chaînes de caractères représentant une expression en notation polonaise inverse. Chaque élément de la liste représente soit un flottant, soit une des opérations suivantes : $+$, $-_1$, $-_2$, $\times$, $\div$. Les opérations seront notées dan la liste '+', 'OPP', '-', 'x', '/'.

En utilisant une pile, écrire une fonction `evalPI(L)` qui prend en argument une telle liste et renvoie la valeur de l'expression qu'elle représente. Par exemple l'expression 10 2.5 + sera représentée par `E=['10','2.5','+']` et `evalPI(E)` renverra 12.5

Question 8. On considère à présent qu'une expression est représentée non pas par une liste mais par une unique chaîne de caractères dans laquelle les nombres et les opérations sont séparés par des espaces. Par exemple, l'expression 10 2.5 + pourra être représentée par `'10 2.5 +'`.

À l'aide d'une fonction de Python et de la question précédente, écrire une fonction qui renvoie la valeur de l'expression représentée.

3 Graphes

Dans cet exercice, les graphes considérés seront non ordonnés et non valués. Les sommets seront numérotés de 0 à n-1 ; n étant l'ordre du graphe. On supposera que l'on a $n \geqslant 2$.

Un graphe sera représenté par sa matrice d'adjacence $A = (a_{i,j})_{0 \leqslant i,j \leqslant n-1}$, dont les coefficients sont définis par : $a_{i,j]} = 1$ si il existe une arête entre le sommet i et le sommet j et $a_{i,j]} = 0$ sinon.

Question 9. Écrire une fonction `testchaine(M,L)` qui prend en argument une matrice `M` représentant un graphe G et une liste `L` d'entiers compris entre 0 et n-1 représentant une séquence de sommets, et qui teste si le parcours des sommets de la liste `L` est possible sur G ou non. Cette fonction renverra un booléen.

Le parcours en profondeur d'un graphe est un algorithme d'exploration d'un graphe en partant d'un sommet, et en visitant récursivement ses voisins. Il est nommé parcours en profondeur car on explore "à fond" les sommets que l'on peut atteindre depuis un sommet donné.

Concrètement, le principe du parcours en profondeur est de partir d'un sommet s et de le marquer. Puis d'appeler récursivement la fonction de parcours en profondeur sur tous les sommets non marqués qui sont reliés à s. Nous utiliserons une liste L, initialement vide, qui correspond aux sommets marqués (déjà visités). Cet algorithme s'écrit en pseudo-code :

```
Appel initial : Graphe G représenté par une matrice M, sommet s entier et liste L vide.

fonction parcours en profondeur d'arguments G,s et L :
        ajouter s à L
        pour tout sommet adjacent à s dans G :
                si sommet n'est pas dans L :
                        appeler la fonction parcours en profondeur avec les arguments : G,sommet,L
        retourner L
```

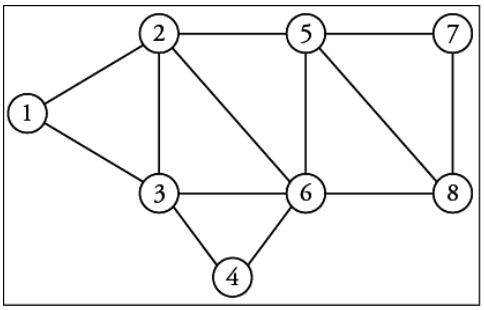
Question 10. Écrire une fonction `profondeur(M,s)` qui prend en arguments la matrice `M` représentant un graphe G et un entier s représentant le sommet de départ du graphe G et qui renvoie la liste L obtenue par parcours en profondeur du graphe G.

Un graphe est dit connexe si il existe un chemin entre tout couple de sommets.

Question 11. Écrire, en utilisant la fonction précédente, une fonction `est_connexe(M)` qui prend en argument une matrice M représentant un graphe G et qui renvoie un booléen indiquant si le graphe G est connexe.

Un graphe G est dit eulérien si il existe un chemin permettant de parcourir toutes ses arêtes une fois et une seule.

Question 12. On considère le graphe G suivant :



Donner (sans justifier) un chemin eulérien sur ce graphe.

Le degré d'un sommet est le nombre d'arêtes qui ont pour extrémité ce sommet.

Question 13. Écrire une fonction `degre(M,s)` qui prend en argument une matrice M représentant un graphe G et un entier s représentant un sommet de ce graphe et qui renvoie le degré de ce sommet.

Le théorème d'Euler donne une condition nécessaire et suffisante pour savoir si un graphe est eulérien :

Un graphe G est eulérien si et seulement si il est connexe et comporte 0 ou 2 sommets de degré impair. De plus :
- Si G ne comporte aucun sommet de degré impair, tous les chemins eulériens sont des cycles (ils ont même départ et même arrivée) et il existe un cycle eulérien partant de tout sommet ;
- Si G comporte exactement deux sommets s_1 et s_2 de degré impair, tous les chemins eulériens ont pour extrémités s_1 et s_2.

Question 14. Écrire une fonction `test_eulerien(M)` qui prend en argument une matrice M représentant un graphe G et qui renvoie un booléen indiquant si G est eulérien ou non.

Dans la suite de cet exercice, l'objectif est de déterminer un chemin eulérien sur un graphe eulérien.

Question 15. Écrire une fonction `cycle(M,s)` qui prend en argument une matrice M représentant un graphe G et un entier s représentant un sommet, et qui renvoie une liste de sommets représentant un cycle quelconque sur G partant (et arrivant sur) de s. On supposera qu'un tel cycle existe, et on s'inspirera du parcours en profondeur du graphe G.

Question 16. Dans cette question G est supposé connexe et sans sommet de degré impair. Il est donc eulérien et depuis chaque sommet il existe un chemin eulérien qui est une cycle.

L'algorithme suivant décrit comment trouver sur un tel graphe une cycle eulérien :

Algorithme CycleEulerien(G,x)

Entrée : un graphe G sans sommet de degré impair et x un sommet de G.

Sortie : une liste [x, x_2, . . . , x_k, x] de sommets de G formant un cycle eulérien.

Stocker dans A l'ensemble des sommets adjacents à x dans G.
Si A est vide :
    renvoyer [x]
Sinon :
    Stocker dans C un cycle quelconque d'origine x dans G : [x = y_1, . . . , y_l = x].
    supprimer les arêtes de C dans G.
    Créer R une liste vide
    Pour i allant de 1 à l :
        Stocker dans R la liste construite en concaténant R avec CycleEulerien(G,y_i)
    retourner R

Écrire une fonction `euler(G,s)` qui prend en arguments une matrice M représentant G et un sommet s et qui renvoie une liste d'entiers représentant une boucle eulérienne sur G partant (et arrivant) de s.

Question 17. Comment adapter l'algorithme précédent pour trouver une chaîne eulérienne dans un graphe connexe contenant exactement deux sommets de degré impair ?