Nom : ..... **Prénom** : ..... **Groupe de TD** : ....

Examen Informatique – 1A – S1

Décembre 2022

Durée : 2h00

Aucun document n'est autorisé, en dehors de l'aide-mémoire annexé en fin de ce sujet. Les calculatrices sont également interdites.

Vous écrivez les solutions de la partie I directement sur le sujet.

1 Partie I (10 points)

1. Qu'affiche le programme suivant ?

```
print(7//4, 7/2, 8%3)
```

Réponse : 1 3.5 2

2. Qu'affiche le programme suivant ?

```
for i in range(5, 0, -1):
    print(i, end=' ')
print()
```

Réponse : 5 4 3 2 1

3. Qu'affiche le programme suivant ?

```
for i in range(0, 2):
    print(i, end = ' ')
    for j in range(0, 3):
        print(j, end=' ')
print()
```

Réponse : 0 0 1 2 1 0 1 2

4. Qu'affiche le programme suivant ?

```
x = 0
def f(x):
    x = x + 1
    return x
print(f(3), x)
```

Réponse : 4 0

5. Qu'affiche le programme suivant ?

```
x = 0
def f(y):
    global x
    x = x + 1
    return y
print(f(3), x)
```

Réponse : 3 1

6. Quelle est la valeur, en décimal, du nombre binaire 1001 1110 ?

Réponse : 158

7. Quelle est l'écriture binaire de 205 ?

Réponse : 1100 1101

8. Quelle est, **en binaire**, la somme des deux nombres binaires 110 1011 et 101 1001 ? Posez l'addition en binaire.

```
      1 1 0 1 0 1 1
 +      1 0 1 1 0 0 1
-----------------------
 =  1 1 0 0 0 1 0 0
```

9. Quelle est, **en binaire**, le produit des deux nombres binaires $\underline{11001}$ et $\underline{1011}$? Posez la multiplication en binaire.

```
              1 1 0 0 1
 ×              1 0 1 1
-------------------------
              1 1 0 0 1
            1 1 0 0 1 0
 +    1 1 0 0 1 0 0 0
-------------------------
 =  1 0 0 0 1 0 0 1 1 1
```

10. On considère le nombre binaire $\underline{a_4\, a_3\, a_2\, a_1\, a_0}$, où $\forall i \in \{0, 1, 2, 3, 4\},\ a_i \in \{0, 1\}$. On suppose que la valeur en décimal de ce nombre est n.
Quelle est l'écriture en binaire de $2 \times n$?

Réponse : $\underline{a_4\, a_3\, a_2\, a_1\, a_0\, 0}$

11. On considère le nombre binaire $\underline{a_4\, a_3\, a_2\, a_1\, 0}$, où $\forall i \in \{1, 2, 3, 4\},\ a_i \in \{0, 1\}$. On suppose que la valeur en décimal de ce nombre est n.
Quelle est l'écriture en binaire de $\dfrac{n}{2}$?

Réponse : $\underline{a_4\, a_3\, a_2\, a_1}$

12. Soit `L` une liste d'entiers. Écrire un programme qui calcule et affiche la liste `M` des entiers de `L` qui sont divisibles par 3. Les nombres de `M` doivent apparaître dans le même ordre que dans `L`. Par exemple, si `L = [1, 6, 4, 3]`, alors `M = [6, 3]`.

```
L = [1, 6, 4, 3]
M = []
for e in L:
    if e % 3 == 0:
        M.append(e)
print(M)
```

13. Soit `L` une liste d'entiers. Écrire un programme qui calcule et affiche la liste de listes `M`, telle que pour `i` $\in \{0, 1, 2\}$, `M[i]` est la liste des entiers n de `L` tels que le reste de la division par 3 de n est `i`. Par exemple, si `L = [1, 6, 4, 3]`, alors `M = [[6, 3], [1, 4], []]`.

```
L = [1, 6, 4, 3]
M = [[], [], []]
for e in L:
    M[e % 3].append(e)
print(M)
```

14. Écrire en Python une fonction f telle que

$$\forall n \in \mathbb{N},\ f(n) = \sum_{i=0}^{n} \left(\prod_{j=0}^{i} (i+j) \right)$$

et afficher la valeur $f(6)$.

```
def f(n):
     s = 0
     for i in range(0, n + 1):
         p = 1
         for j in range(0, i + 1):
             p = p * (i + j)
         s = s + p
     return s
print(f(6))
```

15. On considère la suite $(U_n)_{n\in\mathbb{N}}$ définie par :

$$U_0 = 0\,;\quad U_1 = 1\,;\quad U_{n+2} = U_n + 2\sqrt{U_{n+1}}.$$

Écrire une fonction récursive `U` qui calcule U_n en fonction de n.

```
def U(n):
     if n <= 1:
         return n
     else:
         return U(n - 2) + 2 * U(n - 1) ** 0.5
```

16. On considère la suite $(U_n)_{n\in\mathbb{N}}$ définie ci-dessus. Écrire une fonction itérative (c'est-à-dire non récursive), `V` qui calcule U_n en fonction de n.

```
def V(n):
    if n <= 1:
        return n
    else:
        a = 0
        b = 1
        for i in range(2, n + 1):
            c = a + 2 * b ** 0.5
            a = b
            b = c
        return c
```

2 Partie II (10 points)

2.1 Matrices

On considère des matrices de taille $m \times n$, qui comportent m lignes et n colonnes. Par exemple, la matrice M ci-dessous est de taille 2×3, elle a donc 2 lignes et 3 colonnes.

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

On choisit de numéroter les lignes et les colonnes à partir de 0. En ajoutant les numéros de lignes et de colonnes, on peut écrire la matrice M sous la forme

$$M = \begin{array}{c} \\ {\scriptstyle 0} \\ {\scriptstyle 1} \end{array} \begin{array}{c} \begin{array}{ccc} {\scriptstyle 0} & {\scriptstyle 1} & {\scriptstyle 2} \end{array} \\ \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \end{array}$$

Étant donné deux entiers i et j tels que $0 \leq i \leq m-1$ et $0 \leq j \leq n-1$, $M_{i,j}$ est le nombre qui se trouve en ligne i et en colonne j. Par exemple, $M_{1,2} = 6$.

Représentation de matrices à l'aide de listes de listes en Python

On choisit de représenter des matrices en Python par des listes de listes. Plus précisément, une matrice de taille $m \times n$, avec m lignes et n colonnes, est codée par une liste comportant m listes de n nombres. Par exemple, la matrice M définie ci-dessus, de taille 2×3, est représentée par la liste

`M = [[1, 2, 3], [4, 5, 6]]`.

Question 1 (1 point) Étant donné une matrice A représentée par une liste de listes `A` et deux entiers i et j, quelle est l'expression Python qui correspond à $A_{i,j}$? Quelle est la complexité du calcul de $A_{i,j}$?

`A[i][j]`
Complexité : constante $O(1)$

Question 2 (1 point) Écrire une fonction `zeros(m, n)`, qui prend en paramètre deux entiers strictement positifs m et n, et renvoie une liste de listes `Z` représantant une matrice Z telle que

$$\forall i \in \mathbb{N},\ 0 \leq i \leq m-1,\ \forall j \in \mathbb{N},\ 0 \leq j \leq n-1,\ Z_{i,j} = 0.$$

Par exemple, `zero(2, 3) = Z = [[0, 0, 0], [0, 0, 0]]` et représente la matrice

$$Z = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Quelle est la complexité de la fonction `zeros` ?

```
def zeros(m, n):
    Z = []
    for i in range(m):
        L = []
        for j in range(n):
            L.append(0)
        Z.append(L)
    return Z
```

Complexité : $O(m \times n)$

Matrices identités

La matrice identité Id_n est une matrice de taille $n \times n$ telle que :

$$(Id_n)_{i,j} = 0 \text{ si } i \neq j, \text{ et } (Id_n)_{i,j} = 1 \text{ si } i = j.$$

Question 3 (1 point) Définir une fonction `identite(n)` qui prend en paramètre un entier positif `n` et une liste de listes qui représente la matrice Id_n. Par exemple, pour $n = 3$,

$$Id_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

et `identite(3) = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]`.
Quelle est la complexité de la fonction `identite` ?

```
def identite(n):
    Id = []
    for i in range(n):
        L = []
        for j in range(n):
            if i == j:
                L.append(1)
            else:
                L.append(0)
        Id.append(L)
```

```
    return Id
```

Complexité : $O(n^2)$

Autre solution : on réutilise `zeros`

```
def identite(n):
    Z = zeros(n) # Complexité : n * n
    for i in range(n):
        Z[i][i] = 1
    return Z
```

Complexité : $O(n^2)$

Produits de matrices

On peut multiplier une matrice X de taille $m \times n$ par une matrice Y de taille $n \times p$. On obtient une matrice Z de taille $m \times p$ telle que $Z_{i,j}$ est la somme des produits des éléments de la ligne i de X par ceux de la colonne j de Y. Formellement, $\forall i \in \mathbb{N},\ 0 \leq i \leq m-1,\ \forall j \in \mathbb{N},\ 0 \leq j \leq p-1$,

$$Z_{i,j} = \sum_{k=0}^{n-1} X_{i,k} \times Y_{k,j}.$$

Par exemple,

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} = \begin{pmatrix} 1\times 1 + 2\times 3 + 3\times 5 & 1\times 2 + 2\times 4 + 3\times 6 \\ 4\times 1 + 5\times 3 + 6\times 5 & 4\times 2 + 5\times 4 + 6\times 6 \end{pmatrix}$$

Question 4 (1 point)
Écrire une fonction `terme(X, Y, i, j)` qui, étant donné deux listes de listes `X` et `Y` représentant respectivement les matrices X et Y et deux entiers i et j, calcule le terme $Z_{i,j}$.
Quelle est la complexité de la fonction `terme(X, Y, i, j)` ?

```
def terme(X, Y, i, j):
    n = len(X[0])
    s = 0
    for k in range(n):
        s += X[i][k] * Y[k][j]
    return s
```

Complexité : $O(n)$, où X est une matrice de m lignes et n colonnes et Y est une matrice de n lignes et p colonnes.

Question 5 (1 point) Écrire une fonction `mult(X, Y)` qui calcule le produit des matrices X et Y. S'il n'est pas possible d'effectuer ce produit, on lèvera l'exception `AssertionError` accompagnée d'un message adéquat avec l'instruction `assert` :

`assert` *condition*`, "`*message d'erreur*`"`

Donner la complexité de la fonction `mult`.

```
def mult(X, Y):
    m = len(X)
    n1 = len(X[0])
    n2 = len(Y)
    p = len(Y[0])
    assert n1 == n2, "Le nombre de colonnes de X doit être égal
       au nombre de lignes de Y"
    Z = []
    for i in range(m):
        L = []
        for j in range(p):
            s = terme(X, Y, i, j)
            L.append(s)
        Z.append(L)
    return Z
```

Complexité : $O(m \times n \times p)$, où X est une matrice de m lignes et n colonnes et Y est une matrice de n lignes et p colonnes.

2.2 Graphes représentés par des matrices d'adjacence

Définitions

Soit un entier naturel n. On considère l'ensemble $S_n = \{0, 1, 2, \cdots, n-1\}$.

Un *graphe orienté* $\mathcal{G}$ comportant n sommets est un couple (S_n, E), où E est un sous-ensemble de $S_n \times S_n$. Un élément de S_n est un *sommet* de $\mathcal{G}$. Un élément $(i, j) \in E$ est un *arc* du sommet i vers le sommet j.

Par exemple, pour $n = 4$, on peut définir le graphe $\mathcal{G}_1 = (S_4, E_1)$ avec

$$E_1 = \{(0,1), (1,0), (1,2), (1,3), (2,3), (3,0)\}.$$

La figure 1 montre une représentation graphique du graphe $\mathcal{G}_1$.

La *matrice d'adjacence* d'un graphe $\mathcal{G} = (S_n, E)$ est une matrice G de taille $n \times n$ telle que

$$G_{i,j} = 1 \text{ si } (i,j) \in E, \text{ et } G_{i,j} = 0 \text{ si } (i,j) \notin E.$$

Par exemple, la matrice d'adjacence du graphe $\mathcal{G}_1$ est la matrice représentée figure 2.

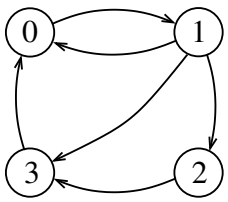

FIGURE 1 – Graphe $\mathcal{G}_1$

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

FIGURE 2 – Matrice d'adjacence G_1 du graphe $\mathcal{G}_1$

Nombre de sommets d'un graphe

Question 6 (0,5 point) Quel est le nombre de sommets du graphe $\mathcal{G}_1$?
Écrire une fonction `nombre_sommets(G)` qui calcule le nombre de sommets du graphe $\mathcal{G}$ représenté par sa matrice d'adjacence G. Quelle est la complexité de cette fonction ?

Le graphe $\mathcal{G}_1$ a 4 sommets.

```
def nombre_sommets(G):
    assert len(G) == len(G[0]), "G doit être une matrice carrée"
    return len(G)
```

Complexité : $O(1)$

Nombre d'arcs d'un graphe

Question 7 (0,5 point) Quel est le nombre d'arcs du graphe $\mathcal{G}_1$?
Écrire une fonction `nombre_arcs(G)` qui calcule le nombre d'arcs du graphe $\mathcal{G}$ représenté par sa matrice d'adjacence G. Quelle est la complexité de cette fonction ?

Le graphe $\mathcal{G}_1$ a 6 arcs.

```
def nombre_arcs(G):
    m = len(G)
    n = len(G[0])
    assert m == n, "G doit être une matrice carrée"
    a = 0
    for i in range(m):
        for j in range(n):
            a = a + G[i][j]
    return a
```

Complexité : $O(n^2)$, où n est le nombre de sommets du graphe `G`.

2.3 Chemins entre deux sommets

Un chemin de longueur k du sommet i_0 vers le sommet i_k est une suite d'arcs

$$(i_0, i_1), (i_1, i_2), (i_2, i_3), \cdots (i_{k-2}, i_{k-1}), (i_{k-1}, i_k).$$

Par convention, pour tout sommet i, on a un chemin de longueur 0 de i vers i.

Par exemple, dans le graphe $\mathcal{G}_1$, on a deux chemins de 0 vers 3 :
- un chemin de longueur 3 : $(0, 1), (1, 2), (2, 3)$;
- un chemin de longueur 2 : $(0, 1), (1, 3)$.

Étant donné la matrice d'adjacence G d'un graphe comportant n sommets, on définit pour tout $k \in \mathbb{N}$ la matrice G^k de la façon suivante :

$$\begin{array}{lcl} G^0 & = & Id_n \\ G^1 & = & G \\ G^{k+1} & = & G^k \times G \end{array}$$

On admet les deux résultats suivants :

1. Étant donné deux sommets i et j, $G^k_{i,j}$ a pour valeur le nombre de chemins de longueur k de i vers j.
2. La longueur du plus court chemin de i vers j est le plus petit k tel que $G^k_{i,j} \neq 0$.

Il peut arriver qu'il n'existe aucun chemin entre deux sommets i et j. Dans ce cas, $\forall k \in \mathbb{N},\ G^k_{i,j} = 0$.

Question 8 (0,5 point) Expliquer pourquoi, s'il existe au moins un chemin entre deux sommets i et j, le plus court de ces chemins a forcément une longueur strictement inférieure au nombre de sommets du graphe.

Si le plus court chemin entre i et j est de longueur supérieure ou égale au nombre de sommets, il passe forcément deux fois par le même sommet. Ce chemin comporte donc une boucle qu'on peut supprimer et ainsi obtenir un chemin plus court entre i et j.

Question 9 (1 point) Écrire une fonction `longueur_plus_court_chemin(G, i, j)` qui calcule la longueur du plus court chemin entre les sommets i et j. S'il n'existe pas de plus court chemin entre i et j, la fonction devra renvoyer la valeur $+\infty$, représentée en Python par **`float("+Inf")`**. Quelle est la complexité de cette fonction ?

```
def longueur_plus_court_chemin(G, i, j):
    m = len(G)
    n = len(G[0])
    k = 0
    nb_sommets = nombre_sommets(G)
    Pk = identite(n)
    while k < nb_sommets and Pk[i][j] == 0:
        k += 1
        Pk = mult(Pk, G) # Complexité : O(n * n * n)
    if k >= nb_sommets:
        return float("+Inf")
    else:
        return k
```

Complexité : pire des cas : $O(n^4)$

Matrice des longueurs des plus courts chemins d'un graphe

On cherche à calculer la matrice M de taille $n \times n$ des longueurs des plus courts chemins d'un graphe $\mathcal{G}$, telle que $M_{i,j}$ est la longueur du plus court chemin entre les sommets i et j de $\mathcal{G}$.

On peut coder cette fonction de façon naïve, en appelant n^2 fois la fonction `longueur_plus_court_chemin`.

Question 10 (0,5 points) Quelle serait la complexité de cette fonction ? (On ne demande pas d'écrire le code).

Complexité : $O(n^2 \times n^4) = O(n^6)$

Pour obtenir une fonction de complexité inférieure, on va utiliser une variable globale `liste_puiss` qui va stocker les puissances successives $[G^0, G^1, G^2, \cdots G^p]$. Le but est de ne stocker que les valeurs nécessaires au calcul de M. L'entier p sera donc soit la longueur maximale de l'ensemble des plus courts chemins entre deux sommets quelconques i et j, soit `nb_sommets(G)` -1, s'il existe un couple de sommets (i, j) du graphe $\mathcal{G}$ tel qu'il n'existe pas de chemin entre i et j.

On définit la variable globale `liste_puiss` et une fonction `init_liste_puiss` qui initialise cette variable à $[G^0]$.

```
liste_puiss = []

def init_liste_puiss(G):
    global liste_puiss
    n = nombre_sommets(G)
    liste_puiss = [Identite(n)]
```

Question 11 (1 point) Définir une fonction `element_liste_puiss(G, i, j, k)`, qui renvoie la valeur $G^k_{i,j}$, en utilisant la variable globale `liste_puiss`. Si cette variable ne contient pas G^k, on la met à jour avec les puissances de G manquantes.

```
def element_liste_puiss(G, i, j, k):
    global liste_puiss
    for p in range(len(liste_puiss), k + 1):
        Gp = mult(liste_puiss[-1], G)
        liste_puiss.append(Gp)
    return liste_puiss[k][i][j]
```

Question 12 (1 points)
En utilisant la variable globale `liste_puiss`, les fonctions `init_liste_puiss` et `element_liste_puiss`, définir la fonction `matrice_plus_courts_chemins(G)` qui permet de calculer la matrice des plus cours chemins du graphe $\mathcal{G}$. Quelle est la complexité de cette fonction ?

```
def matrice_plus_courts_chemins(G):
    global liste_puiss
    init_liste_puiss(G) # en n * n
    m = len(G)
    n = len(G[0])
    M = zeros(n, n) # en n * n
    nb_sommets = nombre_sommets(G) # en O(1)
    for i in range(n): # n * n * n
        for j in range(n):
            k = 0
            fini = False
            while k < nb_sommets and not fini: # au max en n
                e = element_liste_puiss(G, i, j, k)
                if e != 0:
                    fini = True
```

```
                else:
                    k += 1
            if fini:
                M[i][j] = k
            else:
                M[i][j] = float("+Inf")
    return M
```

Complexité : La plupart du temps, `element_liste_puiss` va être calculé en temps constant. On va globalement calculer $[G^0, G^1, G^2, \cdots G^p]$ en $(p+1) \times n^3$, avec $p < n$. La complexité est donc en $O(n^4)$.