+1/1/60+

EPFL

1

Enseignant·e·s: G. Maatouk, L. Testa
Informatique et Calcul Scientifique (ICS) - CMS
10 janvier 2024
Durée : 105 minutes

Dalton Joe

SCIPER : **987654**

Attendez le début de l'épreuve avant de tourner la page. Ce document est imprimé recto-verso, il contient 10 questions sur 16 pages, les dernières pouvant être vides. L'examen est sur 42 points dont 2 points de bonus. Ne pas dégrafer.

- Posez votre **carte d'étudiant·e** sur la table.
- L'utilisation d'une **calculatrice** et de tout **outil électronique** est **interdite** pendant l'épreuve.
- Pour les questions à **choix unique**, on comptera :
  les points indiqués si la réponse est correcte,
  0 point si il n'y a aucune ou plus d'une réponse inscrite,
  0 point si la réponse est incorrecte.
- Les algorithmes demandés sont à écrire sous forme de fonctions Python. Mettez votre code en forme en respectant les indentations: 1 carreau = 1 espace (donc 4 carreaux = 1 tab).
- Vous n'avez pas besoin de commenter votre code mais vous pouvez le faire si vous pensez que cela aide à sa compréhension.
- Utilisez un **stylo** à encre **noire ou bleu foncé** et effacez proprement avec du **correcteur blanc** si nécessaire.
- Répondez dans l'espace prévu (**aucune** feuille supplémentaire ne sera fournie).
- Les brouillons ne sont pas à rendre: ils ne seront pas corrigés.

Respectez les consignes suivantes | Observe this guidelines | Beachten Sie bitte die unten stehenden Richtlinien		
choisir une réponse | select an answer Antwort auswählen	ne PAS choisir une réponse | NOT select an answer NICHT Antwort auswählen	Corriger une réponse | Correct an answer Antwort korrigieren
ce qu'il ne faut **PAS** faire | what should **NOT** be done | was man **NICHT** tun sollte		

Première partie, questions à choix unique

Pour chaque énoncé proposé, une ou plusieurs questions sont posées. Pour chaque question, marquer la case correspondante à la réponse correcte sans faire de ratures. Il n'y a qu'**une seule** réponse correcte par question.

Question 1 (2 points)
Qu'affiche le code suivant ?

```
def my_func(L):

    n = len(L)
    bas = 0
    haut = n-1
    while haut >= bas :
        milieu = (bas+haut)//2
        if L[milieu] == milieu:
            bas = milieu + 1
        else :
            haut = milieu - 1
    return bas

L = [0, 1, 2, 3, 4, 5]
print(my_func(L))
```

- ☐ 4
- ☐ 2
- ☐ 6
- ☐ 0

Question 2 (2 points)

On considère la modification suivante de l'algorithme de recherche binaire vu en cours.

```
def recherche_binaire(L ,x):

    n = len(L)
    bas = 0
    haut = n-1
    count = 0
    while haut >= bas :
        count += 1
        milieu = (bas+haut)//2
        if L[milieu] == x:
            return count
        elif L[milieu] > x:
            haut = milieu - 1
        else :
            bas = milieu + 1
    return float('inf')
```

Soit la liste `L = [-8, -5, -4, -2, 0, 1, 2, 9, 12, 15, 18, 20, 26, 28, 32, 35]`.
Qu'affiche l'instruction `print(recherche_binaire(L,1))`?

- ☐ 'inf'
- ☐ 1
- ☐ 4
- ☐ 3
- ☐ 0
- ☐ 2

Question 3 (2 points)

```
def tri_par_fusion (L, bas, haut):

    if haut - bas > 0:
        milieu = ( bas + haut ) // 2
        tri_par_fusion (L, bas, milieu)
        tri_par_fusion (L, milieu +1, haut)
        fusion (L, bas, milieu, haut)
```

```
def fusion(L, bas, milieu, haut):
    L1 = L[bas:milieu+1]
    L2 = L[milieu+1:haut+1]

    L1.append(float('inf'))
    L2.append(float('inf'))
    L1_index = 0
    L2_index = 0
    for i in range(bas, haut+1):
        if L1[L1_index] <= L2[L2_index]:
            L[i] = L1[L1_index]
            L1_index += 1
        else:
            L[i] = L2[L2_index]
            L2_index += 1
```

On considère la liste `L = [10, 2, 25, 14, 62, 45, 84, 6]`. Quelles sont les deux premières sous-listes à être fusionnées lors de l'appel `tri_par_fusion(L, 0, len(L)-1)`?

- ☐ `[10, 2, 25, 14]` et `[62, 45, 84, 6]`
- ☐ `[62, 45]` et `[84, 6]`
- ☐ `[84]` et `[6]`
- ☐ `[62]` et `[45]`
- ☐ `[10]` et `[2]`
- ☐ `[25]` et `[14]`
- ☐ `[10, 2]` et `[25, 14]`

Question 4 (2 points)

On considère les deux fonctions suivantes définies sur les entiers strictements supérieurs à 1:

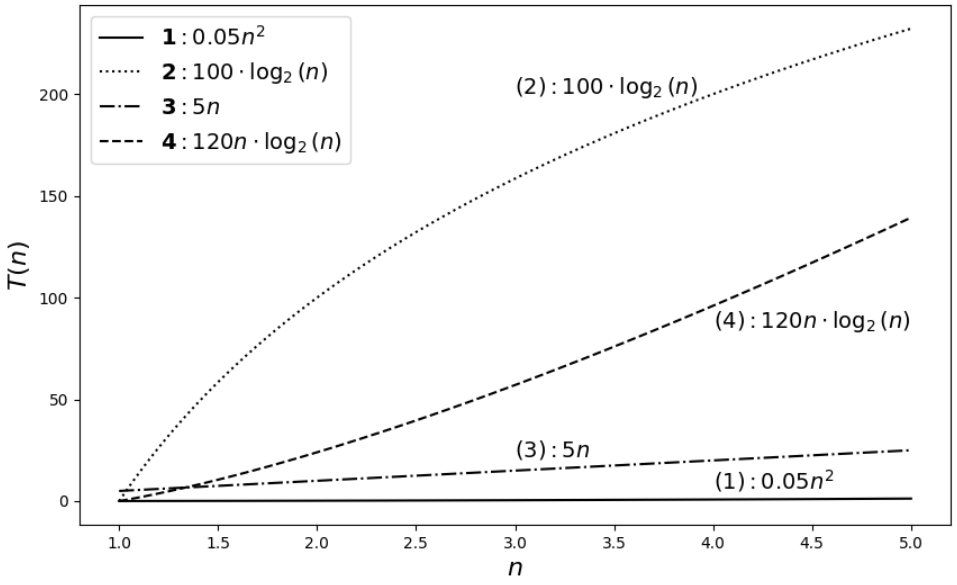
$$f(n) = \frac{n}{\log_2(n)} \text{ et } g(n) = \sqrt{n}\log_2(n).$$

Laquelle des affirmations suivantes est vraie?

- ☐ $f(n) = \Theta(g(n))$
- ☐ $f(n) = \mathcal{O}(g(n))$ mais $g(n)$ n'est pas $\mathcal{O}(f(n))$
- ☐ On ne peut pas comparer l'ordre de croissance de f et de g
- ☐ $g(n) = \mathcal{O}(f(n))$ mais $f(n)$ n'est pas $\mathcal{O}(g(n))$

Question 5 (2 points)

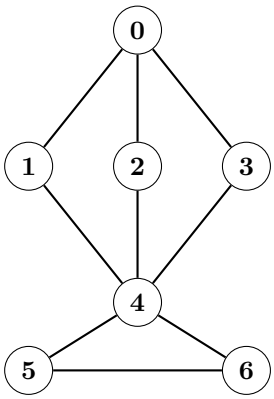
Le graphe suivant représente les temps de parcours de quatre algorithmes différents pour résoudre le même problème.



Ordonnez les quatre algorithmes du plus efficace au moins efficace.

- ☐ 2 3 1 4
- ☐ 1 3 2 4
- ☐ 2 4 3 1
- ☐ 4 2 3 1
- ☐ 1 3 4 2
- ☐ 2 3 4 1

On donne un graphe `G` ci-dessous, et on reproduit les algorithmes **`BFS`** et **`DFS_arbre`** vus en cours.




```
from collections import deque

def BFS(G,s):
    a_parcourir = deque([s])
    vu = [0 for u in G]
    vu[s] = 1

    while a_parcourir:
        sommet = a_parcourir.popleft()
        for u in G[sommet]:
            if not vu[u]:
                a_parcourir.append(u)
                vu[u] = 1
        print(sommet, end = " ")
```

```
def DFS_arbre(G, s):
    vu[s] = 1
    for u in G[s]:
        if not vu[u]:
            T[s].append(u)
            T[u].append(s)
            DFS_arbre(G,u)

G = #dict de listes d'adjacence
vu = [0 for u in G]
T = {u:[] for u in G}
```

Question 6 (2 points)

L'appel **`BFS(G, 2)`** produit l'affichage suivant:

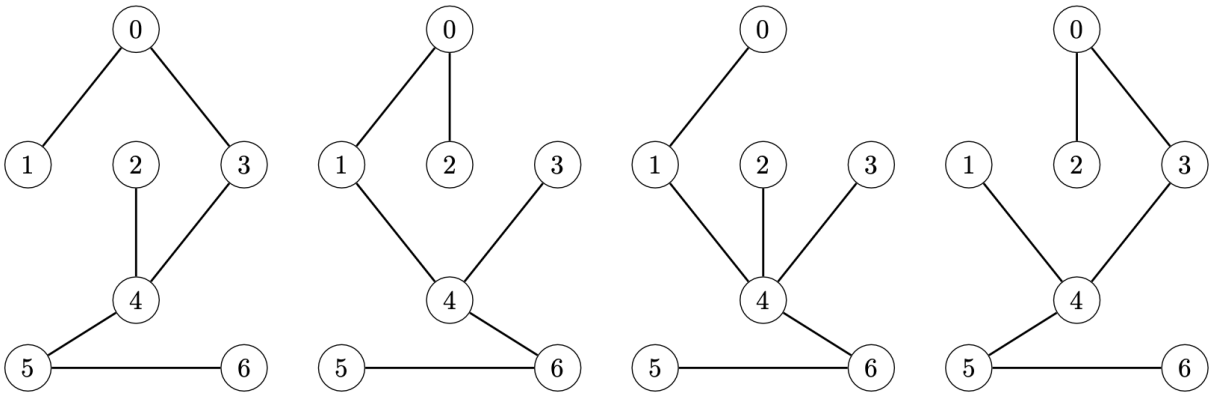
`2 4 0 5 6 3 1`

Lequel des dictionnaires ci-dessous **ne peut pas** être une représentation par listes d'adjacence de `G`?

- ☐ `G = {0:[2, 3, 1], 1:[4, 0], 2:[4, 0], 3:[4, 0], 4:[5, 6, 2, 3, 1], 5:[4, 6], 6:[4, 5]}`
- ☐ `G = {0:[1, 2, 3], 1:[0, 4], 2:[4, 0], 3:[0, 4], 4:[5, 6, 3, 2, 1], 5:[6, 4], 6:[5, 4]}`
- ☐ `G = {0:[3, 2, 1], 1:[0, 4], 2:[4, 0], 3:[0, 4], 4:[5, 2, 6, 1, 3], 5:[6, 4], 6:[5, 4]}`
- ☐ `G = {0:[1, 3, 2], 1:[4, 0], 2:[4, 0], 3:[4, 0], 4:[2, 5, 6, 3, 1], 5:[6, 4], 6:[4, 5]}`

Question 7 (2 points)

On donne ci-dessous quatre arbres couvrants de `G`.



Lequel des quatre arbres couvrants **ne peut pas** avoir été produit par **`DFS_arbre(G, 2)`**?

- ☐ T2
- ☐ T4
- ☐ T3
- ☐ T1

Deuxième partie, questions de type ouvert

Répondre dans l'espace dédié. Laisser libres les cases à cocher : elles sont réservées au correcteur.

Question 8: *Cette question est notée sur 7 points.*

0 1 2 3 4 5 6 7 .5

On donne l'algorithme ci-dessous.

```
def mafonction(L):
    n = len(L)
    for i in range(n-1,-1,-1):
        j = i
        while j < n-1 and L[j] > L[j+1]:
            L[j], L[j+1] = L[j+1], L[j]
            j += 1

        print(f"{i}e element: {L}")
```

(a) Qu'affichent les instructions suivantes?

```
L = [1, 3, 0, -6, 100, -1]
mafonction(L)
```

(b) Quel algorithme vu en cours fonctionne selon le même principe que celui-ci?

(c) On dénote par $T(n)$ le temps de parcours de l'algorithme **`mafonction()`** lorsqu'il prend en entrée une liste de taille n, dans le pire des cas. Donner l'ordre de croissance de $T(n)$ en notation $\Theta(\cdot)$, en justifiant brièvement votre réponse.

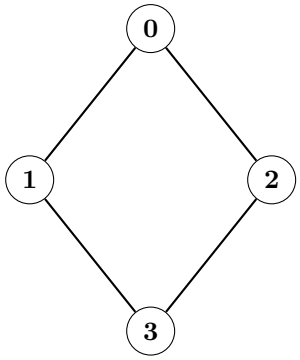
(d) Pour quelles instances le temps de parcours est-il minimal? Donner l'ordre de croissance du temps de parcours **dans le meilleur des cas** en notation $\Theta(\cdot)$, en justifiant brièvement votre réponse.

Question 9: *Cette question est notée sur 12 points, dont 2 points de bonus.*

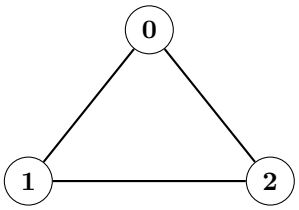
0 1 2 3 4 5 6 7 8 9 10 11 12

Un graphe G non dirigé est dit **2-coloriable** si on peut colorier ses sommets avec deux couleurs, par exemple rouge et vert, de sorte que chaque arête du graphe relie deux sommets de couleurs différentes (donc aucune arête ne relie deux sommets rouges, ou deux sommets verts).

Par exemple, le graphe G ci-dessous est 2-coloriable puisqu'on peut colorier 0 et 3 en rouge et 1 et 2 en vert (ou 0 et 3 en vert et 1 et 2 en rouge). Par contre H n'est pas 2-coloriable: en effet si on donne une couleur à 0 (par exemple rouge), alors 1 et 2 doivent être verts car ils sont voisins de 0; or 1 et 2 sont voisins entre eux donc ils ne peuvent pas avoir la même couleur.

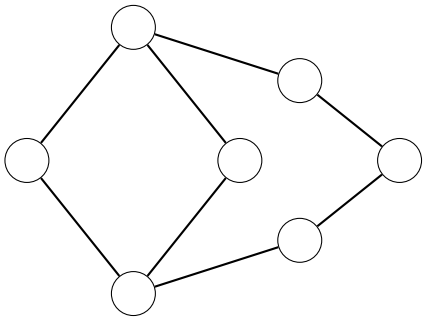


G est 2-coloriable

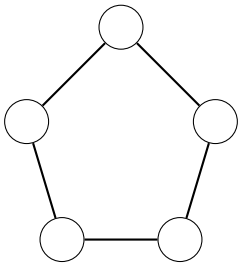


H n'est pas 2-coloriable

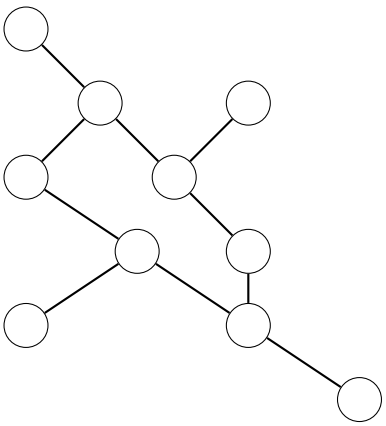
(a) Pour chacun des trois graphes ci-dessous, s'il existe un 2-coloriage valide (c'est-à-dire tel qu'il n'existe aucune arête liant deux sommets de même couleur) du graphe, donnez ce coloriage en écrivant "R" ou "V" dans le sommet (ou à côté du sommet) correspondant. Sinon, indiquez "non 2-coloriable" sous le graphe.



G1



G2



G3

(b) Donnez un algorithme **coloriage** qui

- prend en entrée un graphe **G** **2-coloriable** représenté par un dictionnaire de listes d'adjacences
- retourne une liste **couleur** indexée par les sommets de **G** et contenant un coloriage valide de **G**; c'est-à-dire que pour tout sommet **u** de **G**, **couleur[u]** a la valeur **"R"** ou **"V"**, et si deux sommets **u** et **v** sont reliés par une arête, ils n'ont pas la même couleur.

Par exemple, les instructions

```
G = {0:[1, 2], 1:[0, 3], 2:[0, 3], 3:[1, 2]}
print(coloriage(G))
```

doivent afficher

```
["R", "V", "V", "R"]
```

ou bien

```
["V", "R", "R", "V"]
```

Remarques:

- Vous n'avez pas besoin de vérifier que le graphe donné en entrée est bien 2-coloriable.
- Vous ne pouvez appeler aucun algorithme dont vous n'écrivez pas le code en entier dans votre réponse.

(c) Donnez le temps de parcours de votre algorithme en notation $\Theta(\cdot)$. Justifiez brièvement ce temps de parcours.

(d) **Question bonus.** Pouvez-vous deviner quel est le critère qui permet de déterminer si un graphe G quelconque est 2-coloriable?

Question 10: *Cette question est notée sur 9 points.*

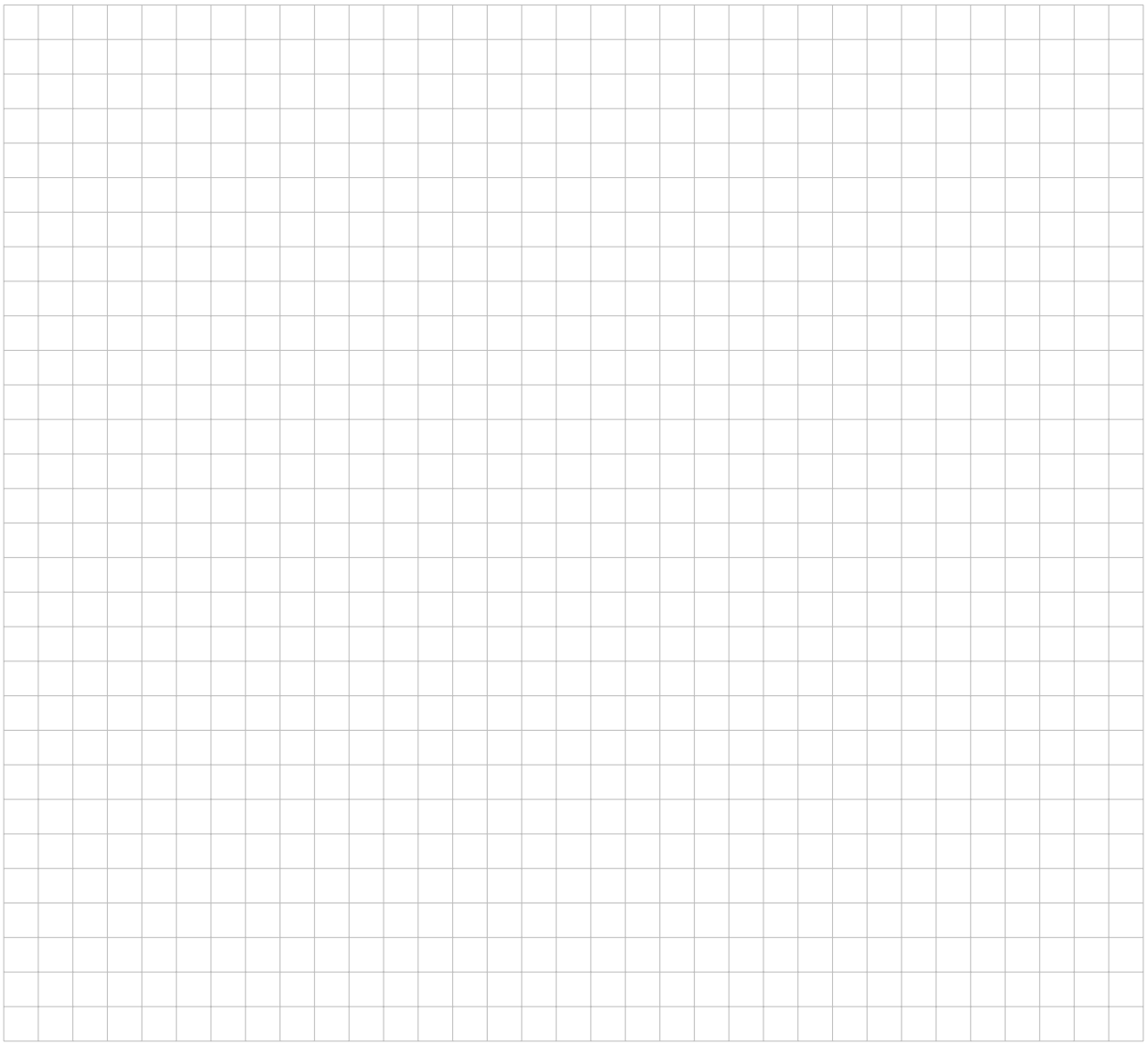
La fonction **`surprise()`** ci-dessous est une implémentation d'un algorithme itératif qui prend un nombre réel **`N`** en entrée.

```
def surprise(N):
    co = 0
    N /= 2
    while N>=1:
        co += 1
        N /= 2
    return co
```

(a) Qu'affichent les instructions suivantes?

```
L = [1, 1.7, 2, 5, 8, 15]

for i in L:
    print(surprise(i))
```



(b) Soit $T(N)$ le temps de parcours de la fonction `surprise()` en fonction de **la valeur** du nombre en entrée N. Donnez l'ordre de croissance de $T(N)$ en notation $\Theta(\cdot)$, en justifiant brièvement votre réponse.

(c) Réécrivez la fonction `surprise()` de manière récursive. Celle-ci ne doit contenir **aucune boucle**.