Le candidat doit choisir 3 exercices qu'il traitera sur les 4 exercices proposés.
Calculatrice interdite

Exercice n° 1

Cet exercice porte sur la recherche textuelle

1. Avec le programme ci-dessous, combien de comparaisons de caractères sont effectuées pendant le calcul de :

```
searchV1("encore chercher toujours chercher.", "chercher")
```

```
#Programme :
def searchV1(texte,motif) :
    """
    :param texte : (str) un texte
    :param motif : (str) un motif
    :return : (list) la liste des occurences de motif dans texte
    :Exemple :
    >>> searchV1( "aabbabbabaa ", "abba")
    [1, 4]
    """
    j = 0
    n = len(texte)
    m = len(motif)
    res = []
    i = 0
    for i in range(n-m+1) :
        j = 0
        while j < m and texte[i+j] == motif[j] :
            j = j+1
        if j == m :
            res.append(i)
    return res
```

2. Quel nom porte l'algorithme mis en œuvre dans **searchV1** ?

3. Construire à la main la table de décalage de l'algorithme de Boyer Moore pour le motif : `"banane"`

Aide :

```
def bons_decalages(motif, alphabet) :
    """
    :param motif : (str) le motif
    :param alphabet : (set) l'alphabet utilisé
    :return : (dict) le dictionnaire associant à chaque lettre de alphabet
    l'indice de sa dernière occurrence dans motif, len(motif) sinon
    :Exemple :
    >>> bons_decalages("abbabc", set("abcd"))
    { "c" : 6, "d" : 6, "b" : 1, "a" : 2 }
    """
    dico={}
    m=len(motif)
    for lettre in alphabet :
        dico[lettre]=m
    for k in range(0,m-1) :
        dico[motif[k]]=m-1-k
    return dico
```

4. En utilisant le résultat de la question précédente, dérouler manuellement l'exécution du programme ci-dessous pendant le calcul de :

```
searchV2("encore chercher toujours chercher.", "chercher")
```

Vous donnerez notamment les valeurs successives de la variable i et `res`.
Indiquer le nombre total de comparaisons de caractère.

```
#Programme :
def searchV2(texte, motif) :
    """
    :param texte : (str) un texte
    :param motif : (str) un motif
    :return : (list) la liste des occurences de motif dans texte
    :Exemple :
    >>> searchV2("aabbabbabaa", "abba")
    [1, 4]
    """
    bd = bons_decalages(motif, set(texte))
    n, m = len(texte), len(motif)
    res = []
    i = 0
    while i <= n - m :
        j = m - 1
        while j >=0 and motif[j] == texte[i+j] :
            j = j - 1
        if j < 0 :
            res.append(i)
            d = bd[texte[i + m - 1]]
        else :
            d = max(1, bd[texte[i+j]] - ( m - 1 - j ) )
        i + = d
    return res
```

5. Comparer les résultats des questions 1 et 4.

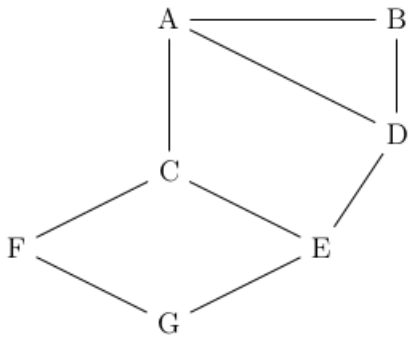
6. Quel nom porte l'algorithme mis en œuvre dans **searchV2** ?

7. Construire à la main la table des **meilleurs** décalages de l'algorithme de Knuth-Mooris-Pratt pour le motif : `"chercher"`.

Exercice n° 2

Cet exercice porte sur les réseaux en général et les protocoles RIP et OSPF en particulier.

On considère un réseau composé de plusieurs routeurs reliés de la façon suivante :



Le protocole RIP

Le protocole RIP permet de construire les tables de routage des différents routeurs, en indiquant pour chaque routeur la distance, en nombre de sauts, qui le sépare d'un autre routeur. Pour le réseau ci-dessus, on dispose des tables de routage suivantes :

Table de routage du routeur A		
Destination	Routeur suivant	Distance
B	B	1
C	C	1
D	D	1
E	C	2
F	C	2
G	C	3

Table de routage du routeur B		
Destination	Routeur suivant	Distance
A	A	1
C	A	2
D	D	1
E	D	2
F	A	3
G	D	3

Table de routage du routeur C		
Destination	Routeur suivant	Distance
A	A	1
B	A	2
D	E	2
E	E	1
F	F	1
G	F	2

Table de routage du routeur D		
Destination	Routeur suivant	Distance
A	A	1
B	B	1
C	E	2
E	E	1
F	A	3
G	E	2

Table de routage du routeur E		
Destination	Routeur suivant	Distance
A	C	2
B	D	2
C	C	1
D	D	1
F	G	2
G	G	1

Table de routage du routeur F		
Destination	Routeur suivant	Distance
A	C	2
B	C	3
C	C	1
D	C	3
E	G	2
G	G	1

Question 1

1. Le routeur A doit transmettre un message au routeur G, en effectuant un nombre minimal de sauts. Déterminer le trajet parcouru.
2. Déterminer une table de routage possible pour le routeur G obtenu à l'aide du protocole RIP.

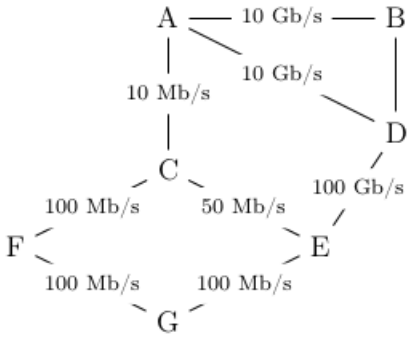
Question 2 Le routeur C tombe en panne. Reconstruire la table de routage du routeur A en suivant le protocole RIP.

Le protocole OSPF

Contrairement au protocole RIP, l'objectif n'est plus de minimiser le nombre de routeurs traversés par un paquet. La notion de distance utilisée dans le protocole OSPF est uniquement liée aux coûts des liaisons. L'objectif est alors de minimiser la somme des coûts des liaisons traversées.
Le coût d'une liaison est donné par la formule suivante :

$$\text{coût} = \frac{10^8}{d}$$

où d est la bande passante en bits/s entre les deux routeurs.
On a rajouté sur le graphe représentant le réseau précédent les différents débits des liaisons.
On rappelle que 1 Gb/s = 1 000 Mb/s = 10^9 bits/s.



Question 3

1. Vérifier que le coût de la liaison entre les routeurs A et B est 0,01.
2. La liaison entre le routeur B et D a un coût de 5. Quel est le débit de cette liaison ?

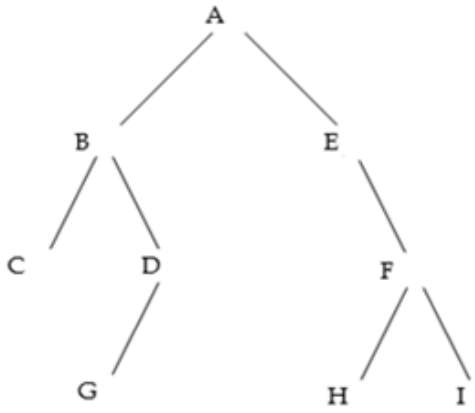
Question 4 Le routeur A doit transmettre un message au routeur G, en empruntant le chemin dont la somme des coûts sera la plus petite possible. Déterminer le chemin parcouru. On indiquera le raisonnement utilisé.

Exercice 3

Cet exercice porte sur les arbres binaires de recherche et la programmation orientée objet.

Dans cet exercice, on utilisera la convention suivante : la hauteur d'un arbre binaire ne comportant qu'un nœud est 1.

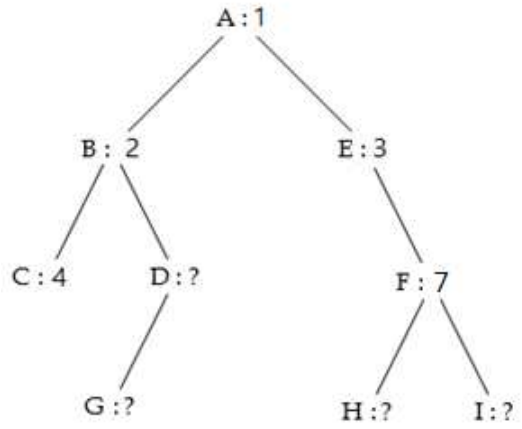
Question 1 : Déterminer la hauteur de l'arbre binaire suivant :



Question 2 : On décide d'étiqueter les nœuds d'un arbre binaire, à l'aide de nombre entier positifs, de la façon suivante :

- L'étiquette de la racine correspond à 1 ;
- L'étiquette d'un fils gauche s'obtient en multipliant par 2 l'étiquette de son père ;
- L'étiquette d'un fils droit s'obtient en ajoutant 1 au double de l'étiquette de son père ;

Par exemple, dans l'arbre ci-dessous, on a utilisé ce procédé pour numéroter les nœuds *A*, *B*, *C*, *E* et *F*.
Remarque : les lettres ne sont apparentes que pour désigner les nœuds.



Question 3 : Dans l'exemple précédent, quelle est l'étiquette du nœud *H* ?

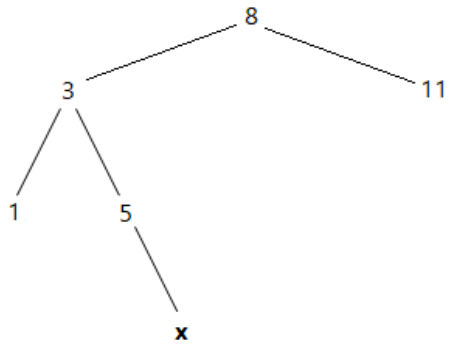
Question 4 : Dans l'exemple précédent, quelle lettre désigne le nœud dont l'étiquette vaut 15 ?

Question 5 : En notant *h* la hauteur d'un arbre binaire quelconque dont les nœuds seraient étiquetés de la manière précédente, qu'elle serait, en fonction de *h*, la valeur de la plus grande étiquette pouvant être présente.

Dans toute la suite de cet exercice, on se place dans le cas particulier d'un **arbre binaire de recherche**, c'est-à-dire un arbre binaire où l'étiquette de chaque nœud est supérieure à celles des nœuds de son fils gauche, et inférieure à celles des nœuds de son fils droit (il existe donc une relation d'ordre entre les étiquettes)

Les étiquettes de cet arbre binaire de recherche seront des **nombres entiers positifs.**

Question 6 : Dans l'arbre binaire de recherche suivant, quel est l'ensemble des valeurs possibles pour l'étiquette **x** du nœud présent ?



Question 7 : Recopier sur votre copie l'arbre binaire de recherche précédent puis y insérer successivement, dans l'ordre, les nœuds dont les étiquettes sont : 4, 9 et 10.

Pour manipuler un arbre binaire de recherche, noté ABR1, on envisage plusieurs possibilités de codages (programmation fonctionnelle, programmation orientée objet)

Question 8 : Associer à chacun des 4 morceaux de codes suivants (écrits en python) le qualificatif correspondant parmi ceux proposés.

Morceau de code	**Qualificatif**
`ABR1.fils_gauche`	tableau indicé
`Fils_gauche(ABR1)`	méthode
`ABR1.fils_gauche()`	attribut
`ABR1[fils_gauche]`	fonction

On envisage maintenant, en programmation orientée objet, une classe `arbre_binaire_de_recherche` dont vous trouverez un descriptif des méthodes en **ANNEXE** à la fin du sujet.

Question 9 : En utilisant les méthodes déjà définies, écrire une méthode `recherche(self,valeur)` retournant :

- *`True`* si le paramètre `valeur` est une étiquette d'un des nœuds présents dans l'arbre binaire de recherche autoréférencé.
- *`False`* dans le cas contraire

Exercice 4

Cet exercice porte sur la programmation de base en Python, les bases de données relationnelles et le langage SQL.

Cet exercice porte sur les piles mais pas sur les piles.

L'entreprise *UpEnergy* est spécialisée dans la vente de piles, accumulateurs, batteries et chargeurs.

Son gérant utilise la structure de données suivante pour gérer, suivant les marques distribuées, son stock de piles plates CR2032 (3 volts) utilisées dans les petits appareils électroménagers (balances de cuisine, pèse-personnes…) :

```
stockCR2032 = {"Energizer" : 25, "Varta" : 12 , "Maxell" : 17}
```

Question 1 : De quel type de structure de données s'agit-il ?

Question 2 : Ecrire en langage python, l'instruction permettant de rajouter dans son stock 30 piles CR2032 de marques "Duracell" (nouveau fournisseur).

Question 3 : Ecrire en langage python, l'instruction permettant de supprimer de son stock les piles CR2032 achetées à son fournisseur "Maxell" car le gérant décide de ne plus vendre cette marque (il ne s'agit donc pas simplement de rendre le stock nul mais bien de supprimer cet article du stock).

Soit la fonction suivante, écrite en python :

```
def mystere(stock,marque):
    if marque in stock.keys():
        return stock[marque]
    else:
        return 0
```

Question 4 : Quelles valeurs sont renvoyées lors de l'exécution des instructions suivantes :

a– `>>> mystere(stockCR2032,"Varta")`

b– `>>> mystere(stockCR2032,"Powercell")`

Le format des piles électriques est normalisé par la Commission électrotechnique internationale (CEI) et par l'*American National Standards Institute* (ANSI). Il existe donc 2 codes pour désigner le même type de pile.

Pour gérer les commandes en ligne, le gérant de l'entreprise *UpEnergy* propose une base de données relationnelles pour obtenir les informations correspondant aux piles classiques (souvent appelées "piles bâton"). Voici les tables constituant cette base de données :

Stock			
Reference	**CEI**	**Marque**	**Quantite**
25684	LR03	Varta	200
25685	LR03	Duracell	250
25686	LR03	Energizer	190
25687	LR06	Duracell	250
25688	LR06	Varta	225
25689	LR14	Duracell	180
25690	LR20	Energizer	170
25691	LR12	Duracell	150
25692	LR12	Varta	150
25693	LR22	Energizer	200

Descriptif			
Code_CEI	**Tension**	**Hauteur**	**Capacite**
LR03	1.5	44	1250
LR06	1.5	50	2850
LR14	1.5	50	8350
LR20	1.5	58	20500
3LR12	4.5	67	6100
LR22	9	47	500

Equivalence	
Code_ANSI	**Code_CEI**
AAA	LR03
AA	LR06
C	LR14
D	LR20
4,5V	3LR12
9V	LR22

On rappelle quelques mots utilisables en langage SQL :
SELECT, FROM, WHERE, JOIN, INSERT INTO, UPDATE, VALUES, COUNT, ORDER BY.

Question 5 : Quel est le nombre de relations constituant le modèle relationnel utilisé ?

Question 6 : Quel est le domaine de l'attribut `Marque` ?

Question 7 : Pour quelle(s) raison(s) l'attribut `Reference` peut-il servir de clé primaire à la table `Stock` ?

Question 8 : Ecrire la requête SQL permettant d'afficher la quantité de piles enregistrées avec la référence 25687.

Question 9 : Ecrire la requête SQL permettant d'afficher le nombre de marques différentes caractérisant les piles LR06 vendues par *UpEnergy*.

Question 10 : Quelle est la tension (en volt) d'une pile dont le code ANSI est AA ?

Question 11 : Ecrire la requête SQL permettant, en effectuant une jointure, d'afficher la tension d'une pile dans le code ANSI est AA.

Suite à une commande effectuée chez un de ses fournisseur, la quantité de piles référencée 25688 prend la nouvelle valeur 300.

Question 12 : Ecrire la requête SQL permettant de mettre à jour la table `Stock`.

ANNEXE

Exercice 3

Descriptifs de quelques méthodes de la classe `arbre_binaire_de_recherche :`

```
def get_etiquette(self) :
     méthode (accesseur) retournant l'étiquette (de type entier int)
     de l'arbre binaire de recherche autoréférencé


def sous_arbre_gauche(self):
     méthode retournant le sous-arbre gauche (de type
     arbre_binaire_de_recherche) de l'arbre binaire de recherche
     autoréférencé


def sous_arbre_droit(self):
     méthode retournant le sous-arbre droit (de type
     arbre_binaire_de_recherche) de l'arbre binaire de recherche
     autoréférencé


def est_une_feuille(self):
     méthode retournant un booléen :
     True si l'arbre binaire de recherche autoréférencé est une
     feuille, False sinon.
```