Ce document a été mis en ligne par l'organisme FormaV®.

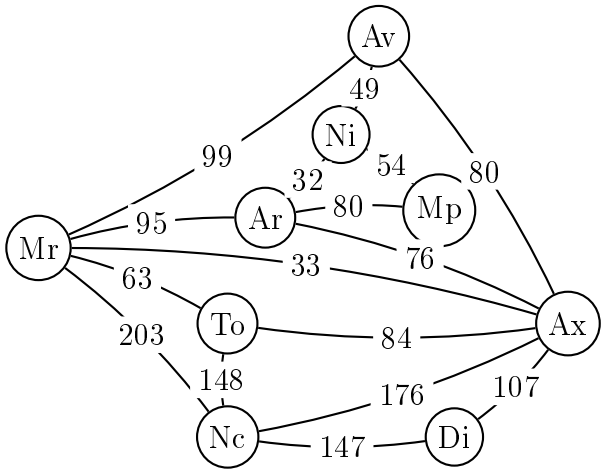
Toute reproduction, représentation ou diffusion, même partielle, sans autorisation préalable, est strictement interdite.

Pour consulter tous nos sujets d'annales et corrigés du Bac Général Maths + NSI, veuillez visiter :
bac-maths-nsi.fr/annales-sujets-et-corriges

24NSIJ2PO1

EXERCICE 1 (6 points)

Partie A



1. Le plus court chemin du sommet Mp au sommet Nc est Mp-Ar-Ax-Nc de longueur 332 km.
2. Les chemins de Mp à Nc visitant un minimum de villes sont Mp-Ar-Mr-Nc et Mp-Ar-Ax-Nc.

Partie B

3. Donner l'implémentation, en langage Python, du graphe de la figure 2 est :

```
G={
    'Ar':['Ax','Mp','Mr','Ni'],
    'Av':['Ax','Mr','Ni'],
    'Ax':['Ar','Av','Di','Mr','Nc','To'],
    'Di':['Ax','Nc'],
    'Mp':['Ar','Ni'],
    'Mr':['Ar','Av','Ax','Nc','To'],
    'Nc':['Ax','Di','Mr','To'],
    'Ni':['Ar','Av','Mp'],
    'To':['Ax','Mr','Nc']
  }
```

On a ordonné les clés et les listes de voisins dans l'ordre alphabétique.

4. LIFO signifie Last In First Out et FIFO signifie First In First Out.
5. Une file est une structure FIFO.

```
def parcours(graphe, sommet):
    f = creerFile()
    enfiler(f, sommet)
    visite = [sommet]
    while not estVide(f):
        s = defiler(f)
        for v in graphe[s]:
            if not (v in visite):
                visite.append(v)
                enfiler(f, v)
    return visite
```

6. Le résultat renvoyé par l'appel `parcours(G, 'Av')`, avec l'ordre proposé plus haut est `['Av','Ax','Mr','Ni','Ar','Di','Nc','To','Mp']` .
7. La fonction `parcours` réalise un parcours en profondeur (proposition B).

8. En modifiant la fonction `parcours`, on écrit une fonction `distance`.

```
def parcours(graphe, sommet):
    f = creerFile()
    enfiler(f, sommet)
    dico = {sommet:0}
    while not estVide(f):
        s = defiler(f)
        for v in graphe[s]:
            if not (v in dico):
                dico[v]=dico[u]+1
                enfiler(f, v)
    return dico
```

9. Le résultat renvoyé par l'appel `distance(G, 'Av')` est le dictionnaire `{'Av':0,'Ax':1,'Mr':1,'Ni':1,'Ar':2,'Di':2,'Nc':2,'To':2,'Mp':2}`.
10. On traduit l'algorithme en pseudo-code donné précédemment.

```
def parcours2(G,s):
    p=creerPile()
    empiler(p,s)
    visite = [s]
    while not estVide(p):
        x=depiler(p)
        if x not in visite:
            visite.append(x)
            for v in graphe[s]:
                empiler(p,v)
    return visite
```

11. Avec le graphe codé par `G`, le résultat renvoyé par l'appel `parcours2(G, 'Av')` est `['Av','Ni','Mp','Ar','Mr','To','Nc','Di','Ax']`.

EXERCICE 2 (6 points)

Partie A

1. – Le nœud initial est appelé racine.
   – Un nœud qui n'a pas de fils est appelé feuille.
   – Un arbre binaire est un arbre dans lequel chaque nœud a au maximum deux fils.
   – Un arbre binaire de recherche est un arbre binaire dans lequel tout nœud est associé à une clé qui est :
   • supérieure à chaque clé de tous les nœuds de son sous arbre gauche,
   • inférieure à chaque clé de tous les nœuds de son sous arbre droit.
2. Les clés obtenues lors du parcours préfixe de l'arbre 1 sont dans l'ordre 1 0 2 3 4 5 6.
3. Les clés obtenues lors du parcours suffixe de l'arbre 2 sont dans l'ordre 0 1 2 4 5 6 3.
4. Les clés obtenues lors du parcours infixe de l'arbre 3 sont dans l'ordre 0 1 2 3 4 5 6.

5. On complète les instructions pour définir et construire les trois ABR. Il y a d'autres réponses possibles.

```
arbre1 = ABR()
arbre2 = ABR()
arbre3 = ABR()
for cle_a_inserer in [1,0,2,3,4,5,6]:
    arbre1.inserer(cle_a_inserer)
for cle_a_inserer in [3,2,4,1,5,0,6]:
    arbre2.inserer(cle_a_inserer)
for cle_a_inserer in [3,1,5,0,2,4,6]:
    arbre3.inserer(cle_a_inserer)
```

6. Voici le code de la méthode hauteur de la classe `ABR` qui renvoie la hauteur d'une instance d'`ABR` :

```
    def hauteur(self):
        if self.est_vide() :
            return -1
        else :
            return 1 + max(self.sag().hauteur(),self.sad().hauteur())
```

Avec cette méthode, la hauteur des trois instances arbre1, arbre2 et arbre3 de la classe `ABR` sont 5, 3 et 2.

7. On complète le code de la méthode récursive `est_presente`.

```
def est_present(self, cle_a_rechercher):
    if self.est_vide() :
        return False
    elif cle_a_rechercher == self.cle() :
        return True
    elif cle_a_rechercher < self.cle() :
        return self.sag().est_present(cle_a_rechercher)
    else :
        return self.sad().est_present(cle_a_rechercher)
```

8. L'instruction qui nécessitera le moins d'appels récursifs avant de renvoyer son résultat est `arbre3.est_presente(7)` avec trois appels récursifs, contre 6 et 4 respectivement pour les deux premiers arbres, puisqu'on cherche une clé qui n'est pas présente, le nombre d'appels récursifs sera la hauteur de la branche suivie plus un.

Partie B

```
def est_partiellement_equilibre(self) :
    if self.est_vide():
        return True
    return abs(self.sag().hauteur()-self.sad().hauteur()) <= 1)
```

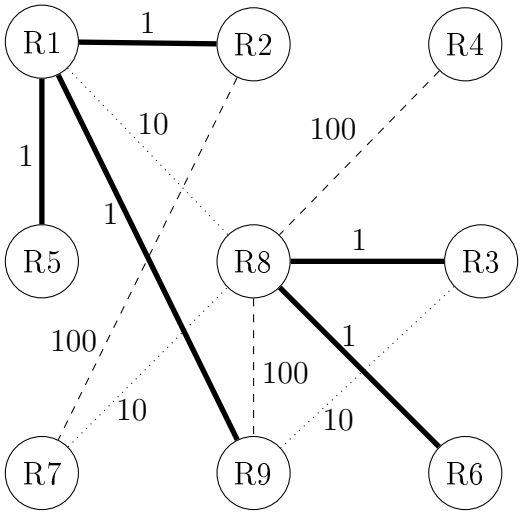
9. Ce qu'on appelle ici un arbre partiellement équilibré est un arbre dont le sous-arbre gauche et le sous-arbre droit ont la même hauteur à un près.

10. Les hauteurs des sous arbres gauche et droite sont 0 et 4 pour l'arbre1, 2 et 2 pour l'arbre 2, 1 et 1 pour l'arbre 3, donc les arbres arbre2 et arbre3 sont partiellement équilibrés.
11. Le sous arbre gauche du sous arbre gauche de arbre2 a pour hauteur 1, tandis que le sous arbre droit du sous arbre gauche de arbre 2 est vide et a donc pour hauteur -1, donc parmi les trois arbres seul arbre 3 est équilibré.
12. On code la méthode récursive `est_equilibre`.

```
    def est_equilibre(self) :
        if not self.est_partiellement_equilibre():
            return False
        return self.sag().est_equilibre() and self.sad().est_equilibre()
```

EXERCICE 3 (8 points)

Partie A : Le réseau informatique d'un hôpital



1. Les chemins possibles d'un paquet de données partant du service de neurologie à destination du service d'imagerie en utilisant le protocole RIP sont R4-R8-R7-R2 et R4-R8-R1-R2.
2. On a complété la table des coûts du nœud R2 selon le protocole RIP.

Nœud R2	
Destination	Coût
R1	1
R3	3
R4	3
R5	2
R6	3
R7	1
R8	2
R9	2

3. Le coût d'une liaison de communication par la technologie FTTH est $C = \dfrac{10^{10}}{10 \times 10^9} = 1$.
4. Avec le protocole OSPF, le chemin pris par un paquet partant du service de neurologie à destination du service d'imagerie est R4-R8-R1-R2.

Partie B : le dossier médical d'un patient

5. Le résultat obtenu avec la requête SELECT NOM,PRENOM FROM PATIENT WHERE NUM_SS LIKE '1%' ; est le tableau des noms et prénoms des hommes (dont le numéro de sécurité sociale commence par 1).
6. La requête SELECT NUM_SS FROM HOSPITALISATION WHERE SERVICE='ORTHOPÉDIQUE' AND DATE LIKE '%2023' ; permet d'afficher le numéro de Sécurité Sociale des patients hospitalisés dans le service intitulé orthopédique durant l'année 2023.
7. La requête SELECT TYPE,DATE FROM EXAMEN JOIN PATIENT ON EXAMEN.NUMSS= PATIENT.NUM_SS WHERE NOM='BAUJEAN' AND PRENOM='EMMA' ; permet d'afficher le type et la date de chaque examen de la patiente Mme Baujean Emma.
8. La requête SELECT PATIENT.NOM, PATIENT.PRENOM FROM PATIENT JOIN CONSULTATION ON PATIENT.NUM_SS=CONSULTATION.NUM_SS JOIN MEDECIN ON CONSULTATION.ID_MEDECIN=MEDECIN.ID_MEDECIN WHERE MEDECIN.NOM='ARNOS' AND MEDECIN.PRENOM='PIERRE' ; permet d'afficher le nom et le prénom de tous les patients du médecin M. ARNOS Pierre.

Partie C : la sécurité des mots de passe d'un médecin

9. On complète le script de la fonction `mdp_fort`.

```
def mdp_fort(mdp):
    if len(mdp)<12:
        return False
    majuscules = 0
    chiffres = 0
    symboles = 0
    for caractere in mdp :
        if caractere.isupper():
            majuscules+=1
        if caractere.isdigit():
            chiffres+=1
        if caractere in liste_symboles:
            symboles+=1
    if majuscules<2 or chiffres<2 or symboles<2:
        return False
    return True
```

10. On complète le script de la fonction `recherche_mot`.

```
def recherche_mot(mdp):
    mot = transforme(mdp)
    trouve = [ ]
    i = 0
    while i<len(mot):
        if mot[i].isdigit(): # si le caractère est un chiffre
            i = i+1
        elif mot[i] in liste_symboles:
            i = i+1
        else:
            # si le caractère est une lettre, on prend les
            # lettres qui la suivent jusqu'au moment où
            # on trouve un chiffre ou un symbole
            chaine = ''
            while mot[i].isalpha():
                chaine = chaine+mot[i]
                i = i+1
            trouve.append(chaine)
    return trouve
```

11. On écrit une fonction `mdp_extra_fort`.

```
def mdp_extra_fort(mdp):
    if not mdp_fort(mdp):
        return False
    mots=recherche_mot(mdp)
    for mot in mots:
        if len(mot)>3 and mot in dicoFR:
            return False
    return True
```

Copyright © 2027 FormaV®. Tous droits réservés.

FormaV® est une marque déposée auprès de l'INPI sous le numéro 5227897.

Ce document a été élaboré par FormaV® avec le plus grand soin afin d'accompagner chaque apprenant vers la réussite de ses examens. Son contenu (textes, graphiques, méthodologies, tableaux, exercices, concepts, mises en forme) constitue une œuvre protégée par le droit d'auteur.

Toute copie, partage, reproduction, diffusion ou mise à disposition, même partielle, gratuite ou payante, est strictement interdite sans accord préalable et écrit de FormaV®, conformément aux articles L.111-1 et suivants du Code de la propriété intellectuelle. Dans une logique anti-plagiat, FormaV® se réserve le droit de vérifier toute utilisation illicite, y compris sur les plateformes en ligne ou sites tiers.

En utilisant ce document, vous vous engagez à respecter ces règles et à préserver l'intégrité du travail fourni. La consultation de ce document est strictement personnelle.

Merci de respecter le travail accompli afin de permettre la création continue de ressources pédagogiques fiables et accessibles.