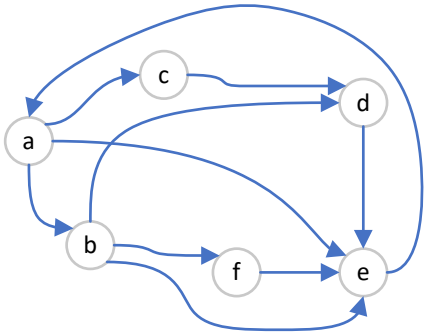
Terminale NSI — **Devoir surveillé n°7** — le 26-03-21

Durée : 2 heures — ***__Calculatrices autorisées__***

Exercice 1 — 6 points

Partie A



1. Donner le tableau des successeurs, des prédécesseurs et la matrice d'adjacence correspondant au graphe G ci-contre.

2. Représenter le graphe non orienté de sommets $\{a, b, c, d, e\}$, ordonnés par ordre alphabétique, défini par la matrice d'adjacence

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

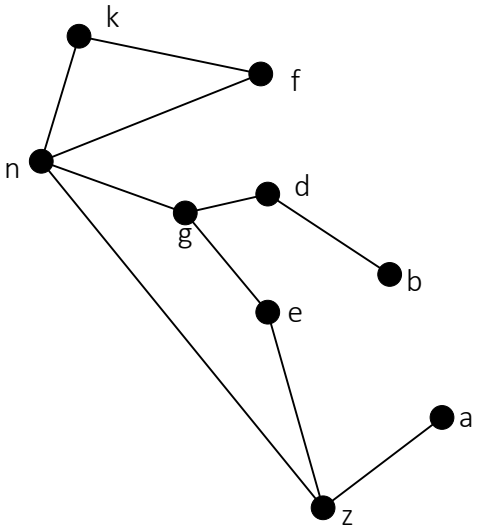
3. Donner la matrice d'adjacence du graphe défini par le tableau des successeurs

Sommets	a	b	c	d	e	f
Successeurs	b, c	a, d	a, e, f	a	b, c, f	d

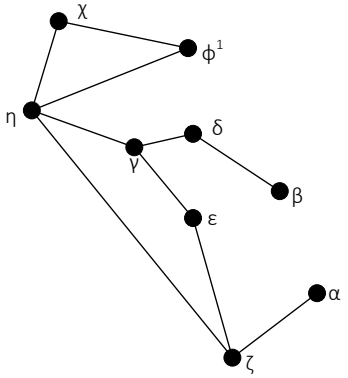
Partie B

G désigne le graphe défini dans la première question de la partie A.

Le graphe L est défini par la représentation ci-dessous.



Les étoiles de la constellation du Lupus sont habituellement désignées avec des lettres grecques, mais on va éviter les erreurs liées à l'ordre alphabétique sur les lettres grecques !



On considère les trois parcours suivants.

```
parcours_1(G, s, visités):
    ajouter s à visités
    afficher_sommet(s)
    pour v dans voisins(s):
        si v n'est pas dans visités :
            parcours_1(G, v, visités)
```

```
parcours_2(G, s):
        f = file_vide()
        enfiler(f, s)
        découverts = {s}
        tant que f n'est pas vide :
           s = défiler(f)
           afficher_sommet(s)
           pour v dans voisins(s):
                si v n'est pas dans découverts :
                   ajouter v à découverts
                   enfiler(f, v)
```

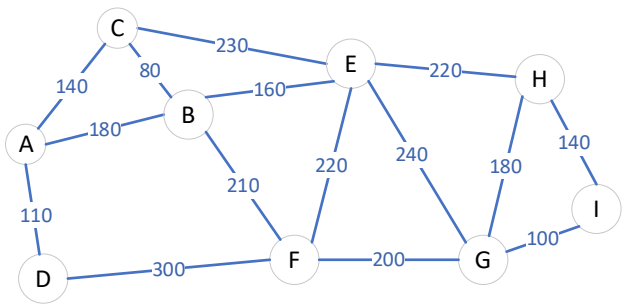
```
parcours_3(G, s):
        p = pile_vide()
        empiler(p, s)
        visités = {}
        Tant que p n'est pas vide :
           s = depiler(p)
           si s n'est pas déjà visité :
              afficher_sommet(s)
              ajouter s à visités
              pour v dans voisins(s):
                 si v n'est pas dans visités :
                    empiler(p, v)
```

Sachant que la boucle pour v dans voisins(s): va parcourir la liste des voisins (ou des successeurs dans le cas d'un graphe orienté) dans l'ordre alphabétique, dans quel ordre vont s'afficher les sommets lors des parcours suivants ?

- parcours_1(G, b, {})
- parcours_1(G, e, {})
- parcours_2(G, b)
- parcours_3(G, b)
- parcours_2(L, a)
- parcours_3(L, d)

Exercice 2 — 6 points

Le graphe ci-dessous représente un réseau routier.



1. Quelle est la longueur du chemin A-D-F-E-H-I ?

On décide d'implémenter ce graphe en Python avec un dictionnaire de dictionnaire.

```
reseau = {'A': {'B':180, 'C':140, 'D':110},
          'B': {'A':180, 'C':80, 'E':160, 'F':210},
          'C': {'A':140, 'B':80, 'E':230},
          'D': {'A':110, 'F':300},
          'E': {'B':160, 'C':230, 'F':220, 'G':240, 'H':220},
          'F': {'B':210, 'D':300, 'E':220, 'G':200},
          'G': {'E':240, 'F':200, 'H':180, 'I':100},
          'H': {'E':220, 'G':180, 'I':140},
          'I': {'G':100, 'H': 140}}
```

2. Décrire une autre implémentation possible.

3. Compléter la fonction `longueur_chemin` permettant de calculer la longueur d'un chemin donné sous forme de liste de sommets, par exemple chemin = ['A', 'D', 'F', 'E', 'H', 'I'].

```
def longueur_chemin(reseau, chemin):
    precedent = ...
    longueur = 0
    for i in range(1, len(chemin)):
        longueur += ...
        precedent = ...
    return longueur
```

Une fonction voisins renvoie la liste des voisins d'un sommet :

```
def voisins(G, s):
    return list(G[s].keys())
```

Une fonction parcours permet d'effectuer un parcours de graphe :

```
def parcours(G, depart, arrive, chemin=None):
    if chemin==None:
        chemin = [depart]
    for sommet in voisins(G, depart):
        if sommet not in chemin:
            parcours(G, sommet, arrive, chemin + [sommet])
```

4. Écrire une autre version de cette fonction parcours d'entête

```
def parcours(G, depart, arrive, lst_chemins, chemin=None):
```

qui va rajouter dans la liste lst_chemins tous les couples (longueur, chemin) de chemins allant du sommet depart au sommet arrive avec leurs longueurs.

5. Écrire une fonction d'entête **def** chemin_plus_court(G, depart, arrive) renvoyant le chemin le plus court allant du sommet de départ au sommet d'arrivé ainsi que sa longueur. On pourra utiliser les fonctions intégrées à Python.

Exercice 3 **4 points**

1. On considère la fonction qui, à tout graphe pondéré fini, associe la longueur maximale d'un chemin du graphe passant au plus une fois par chaque sommet. Cette fonction est-elle calculable ? Justifier votre réponse.

2. Qu'est-ce qu'un problème indécidable ?

3. Qu'est-ce que le problème de l'arrêt ?

4. En s'appuyant sur la fonction contradiction suivante, montrer que le problème de l'arrêt est indécidable. On expliquera en quoi consiste la fonction halt.

```
def contradiction(P: str):
    if halt(P, P):
        while True:
            print("Le code boucle")
    else:
        print("Le code s'arrête")
```

Exercice 4 **4 points**

1. **a.** Expliquer les différences entre chiffrement symétrique et chiffrement asymétrique. Citer un algorithme de chaque sorte.

b. Décrire les échanges permettant de sécuriser une communication HTTPS.

Kid-RSA

L'algorithme Kid-RSA est un algorithme à but pédagogique proposé par Neal Koblitz.

Alice choisit quatre nombres *a*, *b*, *a1*, *b1*, puis calcule :

$M = a \times b - 1$
$e = a1 \times M + a$
$d = b1 \times M + b$
$n = (e \times d - 1)/M$

La clé publique d'Alice est (*n*, *e*) et sa clé privée est *d*.

Un message est sous la forme d'un nombre entier *P* strictement inférieur à *n*.
Le message codé *C* est le reste de la division euclidienne de $e \times P$ par *n*.
À partir du message codé *C*, on retrouve le message d'origine *P* en prenant le reste de la division euclidienne de $C \times d$ par *n*.

On prend *a* = 12, *b* = 16, *a1* = 7, *b1* = 20.

2. **a.** Déterminer la clé publique et la clé privée.

b. Coder le message *P* = 18 245.

c. Décoder le message *C* = 4 664.

Eléments de correction du devoir surveillé n°7

Exercice 1 **6 points**

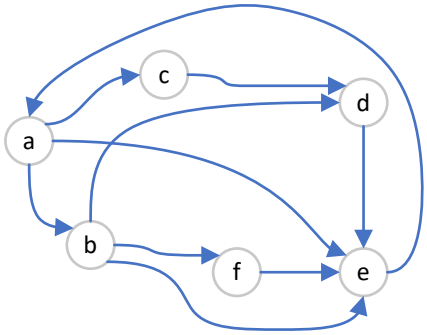
Partie A

1.

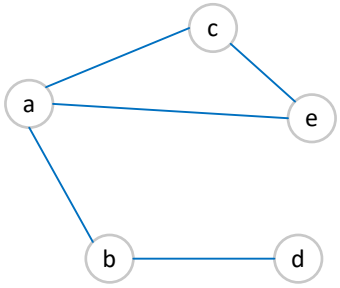
Sommets	a	b	c	d	e	f
Successeurs	b, c, e	d, e, f	d	e	a	e

Sommets	a	b	c	d	e	f
Prédécesseurs	e	a	a	b, c	a, b, d, f	b

Matrice d'adjacence : $\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$



2.

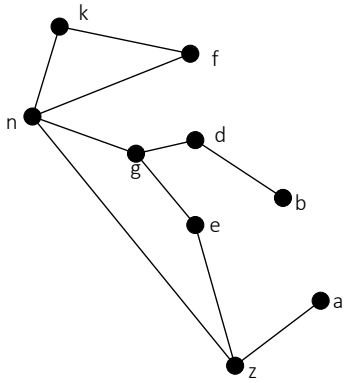


3.

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

Partie B

parcours_1(G, b, {}) affiche : b – d – e – a – c – f
parcours_1(G, e, {}) affiche : e – a – b – d – f – c
parcours_2(G, b) affiche : b – d – e – f – a – c
parcours_3(G, b) affiche : b – f – e – a – c – d
parcours_2(L, a) affiche : a – z – e – n – g – f – k – d – b
parcours_3(L, d) affiche : d – g – n – z – e – a – k – f – b



Exercice 2 **6 points**

1. La longueur du chemin A-D-F-E-H-I est 110 + 300 + 220 + 220 + 140 = 990

2. On aurait pu utiliser une matrice d'adjacence dans laquelle le coefficient ligne i colonne j est égal à la distance entre les sommets i et j si ceux-ci sont voisins, à 0 sinon, les sommets étant ordonnés par ordre alphabétique.

3.

```
def longueur_chemin(reseau, chemin):
    precedent = chemin[0]
    longueur = 0
    for i in range(1, len(chemin)):
        longueur += reseau[precedent][chemin[i]]
        precedent = chemin[i]
    return longueur
```

4.

```
def parcours(G, depart, arrive, lst_chemins, chemin=None):
    if chemin==None:
        chemin = [depart]
    if depart == arrive:
        lst_chemins.append((longueur_chemin(reseau, chemin), chemin))
    for sommet in voisins(G, depart):
        if sommet not in chemin:
            parcours(G, sommet, arrive, lst_chemins, chemin + [sommet])
```

5.

```
def chemin_plus_court(G, depart, arrive):
    lst_chemins = []
    parcours(G, depart, arrive, lst_chemins)
    return min(lst_chemins)
```

Exercice 3 **4 points**

1. Cette fonction est calculable. En effet, il y un nombre fini de chemins possibles (au plus *n*! pour un graphe à *n* sommets). Il est possible de calculer leurs longueurs puis de trouver la longueur maximale.

2. Un problème indécidable est un problème de décision pour lequel il n'existe pas d'algorithme qui, pour chaque entrée, réponde par oui ou non à la question posée par le problème.

3. Le problème de l'arrêt pose la question, pour un programme donné *P* associée à une entrée *x*, à dire si le programme *P* s'arrête sur l'entrée *x* ou pas. Alan Turing a prouvé que ce problème est indécidable en 1936.

4. On suppose l'existence d'une fonction halt(P, x) qui, pour tout programme *P* et toute entrée *x* de *P* renvoie True si le programme *P* s'arrête sur *x* et False sinon.
On appelle ensuite contradiction sur son propre code, donc contradiction(contradiction).
Si cet appel s'arrête, alors halt(P, P) renvoie True et on entre dans une boucle infinie. C'est absurde.
Si cet appel ne s'arrête pas, alors halt(P, P) renvoie False et le code s'arrête.
Dans les deux cas, on aboutit à une contradiction, ce qui montre que la fonction halt ne peut pas exister.

Exercice 4 **4 points**

1. **a.** Alice et Bob veulent échanger des messages.
Avec un chiffrement symétrique, ils utilisent tous les deux la même clé de chiffrement, qui permet de chiffrer et de déchiffrer les messages.
Avec un chiffrement asymétrique, une clé est publique et une clé est privée. Tout le monde possède la clé publique et peut chiffer les messages mais seul le détenteur de la clé privée peut les déchiffrer (ou le contraire dans le cas des signatures électroniques par exemple).
AES et DES sont des algorithmes de chiffrement symétrique ; RSA est un algorithme de chiffrement asymétrique.

b. Les échanges HTTPS se déroulent ainsi :

1. Le client contacte le serveur.
2. Le serveur répond avec son certificat, qui contient sa clé publique et la signature d'une autorité de certification (CA).
3. Le client déchiffre la signature avec la clé publique du CA, vérifie la validité du certificat, et envoie une clé de chiffrement symétrique cryptée avec la clé publique du serveur.
4. Le serveur déchiffre la clé de chiffrement symétrique avec sa clé privée.
5. Les échanges suivants sont cryptés avec la clé de chiffrement symétrique.

2. **a.** $M = 12 \times 16 - 1 = 191$
$e = 7 \times 191 + 12 = 1349$
$d = 20 \times 191 + 16 = 3836$
$n = (1349 \times 3836 - 1)/191 \ = 27\,093$

b. $e \times P = 1349 \times 18\,245 = 24\,612\,505$
$24\,612\,505 = \ 27\,093 \times 908 + 12\,061$
D'où le message codé : C = 12 061.

c. $C \times d = 4664 \times 3836 = 17\,891\,104$
$17\,891\,104 = 27\,093 \times 660 + 9\,724$
D'où le message décodé : P = 9 724.

Source Kid-RSA : https://www.cs.uri.edu/cryptography/publickeykidkrypto.htm