Bac Général Maths + NSI (Mathématiques et Numérique et sciences informatiques)

SUJET D'EXAMEN (SESSION OFFICIELLE)

Numérique et sciences informatiques - Session 2025

Sujet de l'examen national tel que publié pour la session indiquée. Source : sujets nationaux publiés par le ministère chargé de l'examen et les services académiques. Document mis à disposition par FormaV pour la révision.

Toutes les annales et leurs corrigés : https://bac-maths-nsi.fr/annales-sujets-et-corriges

BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2025

NUMÉRIQUE ET SCIENCES INFORMATIQUES

JOUR 1

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice n'est pas autorisé.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.

Ce sujet comporte 21 pages numérotées de 1/21 à 21/21.

Le sujet est composé de trois exercices indépendants.

Le candidat traite les trois exercices.

Exercice 1 (6 points)

Cet exercice porte sur les bases de données et les requêtes SQL, les arbres binaires et les algorithmes sur les arbres binaires.

Partie A

Dans cet exercice, on pourra utiliser les clauses du langage SQL pour :

- construire des requêtes d'interrogation à l'aide de `SELECT` , `FROM` , `WHERE` (avec les opérateurs logiques `AND` , `OR`), `JOIN ... ON` ;
- construire des requêtes d'insertion et de mise à jour à l'aide de `UPDATE` , `INSERT` , `DELETE`.

Une exoplanète est une planète située hors du système solaire. La plupart des exoplanètes découvertes à ce jour orbitent autour d'une unique étoile.

Une étoile est repérée précisément dans le ciel par son ascension droite et sa déclinaison (voir Figure 1). La direction de coordonnées `(0, 0)` est une direction fixe du ciel servant d'origine de ce système de coordonnées.

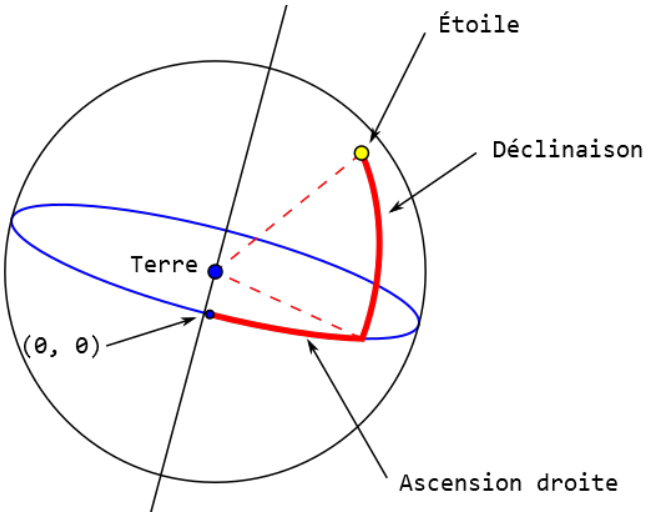


Figure 1. Coordonnées d'une étoile (adaptée depuis *https://commons.wikimedia.org/wiki/File:Coordonnees_equatoriales.svg*)

On considère dans cet exercice deux relations décrivant des étoiles et les exoplanètes orbitant autour d'elles :

- la relation `Etoiles` contient les informations décrivant des étoiles :
  - `id_etoile` : l'identifiant unique de l'étoile (nombre entier) ;
  - `nom` : le nom de l'étoile (chaîne de caractères) ;
  - `ascension` : l'ascension droite de l'étoile en degré (nombre réel) ;
  - `declinaison` : la déclinaison de l'étoile en degré (nombre réel).

- la relation `Exoplanetes` contient les informations décrivant des exoplanètes :
  - `id_exoplanete` : l'identifiant unique de l'exoplanète (nombre entier) ;
  - `masse` : la masse de l'exoplanète, exprimée sous la forme d'une fraction de la masse de la planète Jupiter (nombre réel) ;
  - `rayon` : le rayon de l'exoplanète, exprimée sous la forme d'une fraction du rayon de la planète Jupiter (nombre réel) ;
  - `id_etoile` : l'identifiant de l'étoile autour de laquelle orbite l'exoplanète (nombre entier).

Une exoplanète dont l'attribut `masse` est égal à `6.84` a une masse 6,84 fois plus grande que celle de la planète Jupiter.

On fournit ci-dessous des extraits de ces deux tables :

Etoiles			
id_etoile	nom	ascension	declinaison
1	109 Psc	26.23	20.08
2	beta Pic	86.82	-51.07
3	K2-21	340.30	-14.49
4	Kepler-11	297.12	41.91

Exoplanetes			
id_exoplanete	masse	rayon	id_etoile
1	6.84	1.15	1
2	11.90	1.65	2
3	8.89	1.20	2
4	0.01	0.16	3
5	0.02	0.22	3
6	0.01	0.16	4
7	0.01	0.26	4

L'attribut `id_exoplanete` est la clé primaire de la relation `Exoplanetes`. L'attribut `id_etoile` est la clé primaire de la relation `Etoiles`.

1. Expliquer pourquoi l'attribut `masse` de la relation `Exoplanetes` ne peut pas servir de clé primaire de cette relation.

2. Donner le nom de l'attribut pouvant être utilisé comme clé étrangère dans la relation `Exoplanetes`. Expliquer son rôle.

3. Donner le résultat de la requête SQL suivante :

```
SELECT masse, rayon
FROM Exoplanetes
WHERE id_exoplanete = 4;
```

4. Écrire une requête SQL permettant d'obtenir l'identifiant et le nom des étoiles dont l'ascension droite est supérieure ou égale à 100 degrés.

On souhaite insérer une nouvelle exoplanète de rayon égal à 0,37 fois celui de Jupiter et pesant 0,03 fois la masse de Jupiter. Cette exoplanète orbite autour de l'étoile Kepler-11 dont l'identifiant est `4`. On pourra attribuer à cette nouvelle exoplanète l'identifiant `9` qui n'apparaît pas dans la relation `Exoplanetes`.

5. Écrire une requête SQL permettant d'insérer cette nouvelle exoplanète dans la base de données.

6. Écrire une requête SQL permettant d'obtenir les rayons des exoplanètes orbitant autour de l'étoile nommée Kepler-11, dont l'identifiant est supposé non connu.

Partie B

On souhaite désormais écrire une application Python permettant de classer et de retrouver efficacement les étoiles selon leur position dans le ciel.

On rappelle qu'une étoile est repérée par son ascension droite et sa déclinaison. Par souci de simplicité, on considère désormais que deux étoiles ont toujours des coordonnées entières et distinctes. On représente en Python les coordonnées d'une étoile par un tuple d'entiers `(ascension, declinaison)`.

Dans la suite, on considère les étoiles dont les coordonnées sont contenues dans la liste de tuples `etoiles` définie par `etoiles = [(29, 21), (17, 14), (10, 30), (35, 13), (30, 63), (15, 20)]`.

On cherche à construire un arbre binaire de recherche à partir des coordonnées présentes dans la liste `etoiles` afin d'accélérer les opérations de traitement sur celles-ci. Pour cela :

- on commence par trier la liste `etoiles` par ordre croissant, afin que l'arbre résultant soit de hauteur minimale ;
- pour construire l'arbre binaire de recherche à partir des éléments de la liste `etoiles` compris entre les indices `debut` (inclu) et `fin` (exclu) :
  - la racine de l'arbre est l'élément d'indice `milieu` définit par

    `milieu = (debut + fin)//2` ;

– on construit récursivement le sous arbre gauche à l'aide des éléments de la liste `etoiles` compris entre les indices `debut` (inclu) et `milieu` (exclu) ;
– on construit récursivement le sous arbre droit à l'aide des éléments de la liste `etoiles` compris entre les indices `milieu + 1` (inclu) et `fin` (exclu).

Pour implémenter cet algorithme, on représente en Python les arbres binaires non vides à l'aide de tuples de trois éléments `(sag, position, sad)` dans lesquels :

- `position` est la valeur de la racine. Cette valeur est le couple de coordonnées permettant de repérer l'étoile ;
- `sag` et `sad` sont respectivement les sous-arbres gauche et droit de l'arbre.

L'arbre vide est quant à lui représenté par `None`.

On rappelle que l'on peut comparer des tuples en Python à l'aide de l'opérateur < : on compare tout d'abord les valeurs à l'indice `0` de chaque couple puis, en cas d'égalité, celles à l'indice `1`.

Ainsi, les expressions `(1, 4) < (2, 3)` et `(1, 4) < (1, 6)` s'évaluent toutes les deux à `True`.

La fonction `sorted` de Python prend en argument une liste et renvoie une nouvelle liste contenant les mêmes valeurs triées dans l'ordre croissant à l'aide de l'opérateur <.

7. Donner la liste renvoyée par l'instruction `sorted(etoiles)`.

8. Dessiner l'arbre binaire représenté par le tuple `(((None, (1, 34), None), (2, 35), None), (11, 36), (None, (17, 30), None))`.

L'arbre construit à partir de la liste `etoiles` a donc pour représentation Python :

```
(((None, (10, 30), None), (15, 20), (None, (17, 14), None)),
(29, 21), ((None, (30, 63), None), (35, 13), None))
```

Il est représenté sur la Figure 2 ci-après.

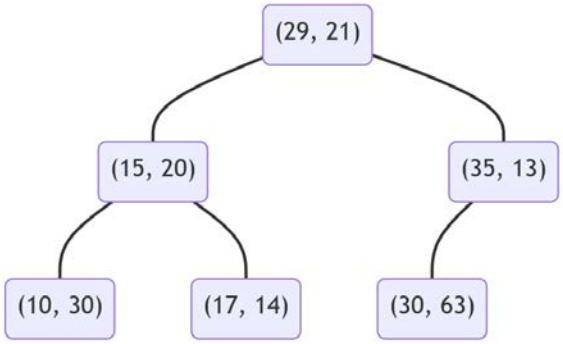


Figure 2. Arbre associé à la liste `etoiles`

9. Dessiner l'arbre binaire de recherche obtenu à partir de la liste :

   `[(1, 33), (2, 30), (2, 33), (4, 30), (8, 39)]`

10. Recopier et compléter les lignes 3, 7, 8, 9 et 11 du code de la fonction `construction` qui prend en paramètres une liste `etoiles` supposée triée par ordre croissant, ainsi que deux entiers `debut` et `fin`. Cette fonction renverra l'arbre binaire de recherche associé aux coordonnées présentes entre les indices `debut` (inclus) et `fin` (exclu) de la liste `etoiles`.

    Par exemple, l'appel initial permettant de construire l'arbre associé à la liste `etoiles` est `construction(etoiles, 0, 6)`.

    L'indice du milieu est 3, le sous-arbre gauche est renvoyé par l'appel `construction(etoiles, 0, 3)` et le sous-arbre droit par `construction(etoiles, 4, 6)`.

```
 1 def construction(etoiles, debut, fin):
 2     if debut == fin:
 3         return ...
 4
 5     milieu = (debut + fin) // 2
 6
 7     sag = construction(...)
 8     racine = ...
 9     sad = ...
10
11     return ...
```

11. Écrire le code de la fonction `en_arbre` qui prend en paramètre une liste `etoiles` de couples de coordonnées non triés et renvoie l'arbre construit selon la démarche décrite plus haut. On pourra utiliser la fonction `construction` de la question précédente.

On souhaite désormais écrire une fonction `contient` qui prend en paramètres un arbre binaire de recherche `arbre` tel que renvoyé par la fonction `construction` ainsi

qu'un tuple d'entiers `position` représentant les coordonnées d'une étoile. Cette fonction renvoie `True` si l'arbre contient cette étoile, `False` dans le cas contraire.

12. Recopier et compléter les lignes 3, 8, 9, 10 et 12 du code de la fonction `contient`.

```
 1 def contient(arbre, position):
 2     if arbre is None:
 3         return ...
 4
 5     sag, valeur, sad = arbre
 6
 7     if position < valeur:
 8         return contient(..., ...)
 9     elif ...:
10         return ...
11     else:
12         return ...
```

Exercice 2 (6 points)

Cet exercice porte sur les systèmes d'exploitation, les processus, les structures de données linéaires, la programmation en Python et en particulier la programmation orientée objet.

Partie A

"Le système d'exploitation est chargé d'allouer les ressources (mémoires, temps processeur, entrées/sorties) nécessaires aux processus et d'assurer que le fonctionnement d'un processus n'interfère pas avec celui des autres."

Source : Wikipédia, extrait de l'article consacré aux processus.

1. Expliquer succinctement, dans ce contexte, ce qu'est un processus.

On rappelle qu'un processus peut-être soit élu, soit bloqué, soit prêt.

2. Recopier et compléter le schéma ci-dessous avec les termes suivants :

   élu, *bloqué*, *prêt*, *élection*, *blocage*, *déblocage*.

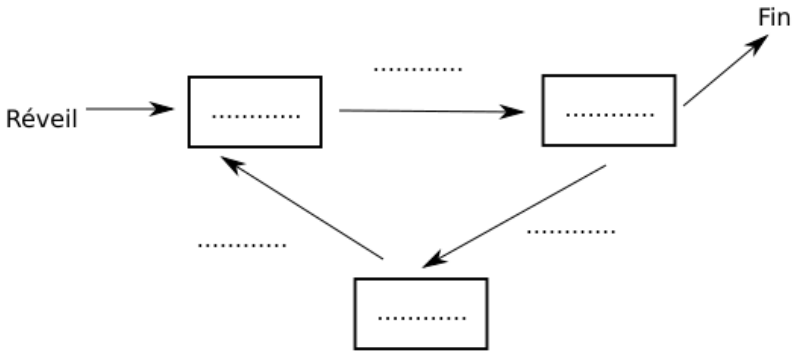


Figure 1. Schéma processus

On considère qu'un monoprocesseur est utilisé. Le système d'exploitation tel un chef d'orchestre, gère l'accès au processeur selon la règle du « premier arrivé, premier servi ». À chaque processus, on associe un instant d'arrivée (instant où le processus demande l'accès au processeur pour la première fois) et une durée d'exécution (durée d'accès au processeur nécessaire pour que le processus s'exécute entièrement).

3. Donner la structure de données la plus adaptée pour gérer l'accès des processus au processeur selon la règle du « premier arrivé, premier servi ».

Le tableau ci-dessous présente les instants d'arrivées et les durées d'exécution de quatre processus :

4 processus		
Processus	instant d'arrivée	durée d'exécution
P1	0	4
P2	2	2
P3	3	4
P4	4	2

4. Recopier et compléter, à l'aide du tableau, le schéma ci-dessous avec les processus P1 à P4 en utilisant la règle du « premier arrivé premier servi ».

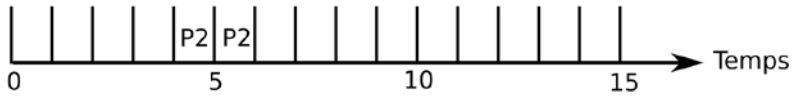


Figure 2. Utilisation du processeur

5. Déterminer le temps qu'a dû attendre le processus P4 avant de pouvoir accéder au processeur.

Partie B

6. Expliquer en quoi consiste la notion d'interblocage.

Afin d'éviter une situation d'interblocage, une solution consiste à attribuer à chaque processus un numéro de priorité.

On souhaite modéliser ce mode de fonctionnement mettant en jeu des numéros de priorité :

- en utilisant une liste de tuples, tuple constitué d'un entier représentant le numéro de priorité ainsi que d'une chaîne de caractères représentant le nom du processus ;
- le processus prioritaire est celui dont le numéro de priorité est le plus petit.

Il est donc important que la liste soit et reste triée dans l'ordre décroissant des numéros de priorités.

Exemple :

```
>>> exemple = [(10, 'Edge'), (8, 'Firefox'), (5, 'Chrome'), (1, 'Vivaldi')]
>>> # La liste est triée, le processus le plus prioritaire est 'Vivaldi'
```

On considère la classe `Priority_Queue` dont l'attribut `liste_priorite` est une liste de tuples, constitués d'un numéro de priorité et d'un nom de processus comme dans l'exemple ci-avant.

```
 1 class Priority_Queue:
 2     def __init__(self):
 3         self.liste_priorite = []
 4 
 5     def est_vide(self):
 6         """Renvoie Vrai si la liste_priorite
 7            est vide, Faux sinon
 8         """
 9         return self.liste_priorite == []
10 
11     def sortir(self):
12         """Retire et renvoie le dernier élément de
13            liste_priorite"""
14         assert ...
15         ...
16 
17     def index_insertion_element(self, element):
18         """Renvoie la position/index d'insertion
19            d'element dans liste_priorite triée
20            par ordre décroissant
21            de numéro priorité
22         """
23         if self.est_vide():
24             ...
25         else:
26             debut = 0
27             fin = len(self.liste_priorite) - 1
28             milieu = (debut + fin) // 2
29             while ... <= fin:
30                 if self.liste_priorite[milieu][0] > ...:
31                     debut = ...
32                 elif self.liste_priorite[milieu][0] < ...:
33                     fin = ...
34                 else:
35                     # cas d'égalité de priorité
36                     ... milieu
37                 milieu = ...
38             return milieu + 1
39 
40     def inserer(self, element):
41         """Modifie liste_priorite en insérant
42            element à la position adéquate
43            dans l'ordre décroissant de
44            numéro de priorité"""
```

7. Écrire l'instruction permettant d'instancier `navigateurs` un objet de la classe `Priority_Queue`.

On rappelle que la méthode `pop()`, appelée sans argument, supprime et renvoie le dernier élément d'une liste.

```
>>> fruits = ['pomme', 'pomme', 'raisin', 'orange', 'poire']
>>> fruits.pop()
'poire'
>>> fruits
['pomme', 'pomme', 'raisin', 'orange']
```

8. Recopier et compléter les lignes 14 et 15 du code de la méthode `sortir` qui après avoir vérifié, sous la forme d'une précondition, que l'objet n'est pas vide, retire et renvoie le dernier élément de `liste_priorite`.

Pour maintenir la liste de priorités triée dans l'ordre décroissant des numéros de priorités, il est indispensable de savoir à quelle position on doit insérer un nouvel élément en fonction de sa priorité.

Cette question ne porte que sur la détermination de la position à laquelle devrait être inséré un élément et cela sans effectuer d'insertion.

On considère, par exemple, que `navigateurs.liste_priorite` contient

`[(10, 'Edge'), (8, 'Firefox'), (5, 'Chrome'), (1, 'Vivaldi')]`.

Si on souhaite insérer :

- l'élément `(12, 'Opera')` on devrait l'insérer au tout début, à la position 0 de `navigateurs.liste_priorite` ;
- l'élément `(6, 'Brave')` on devrait l'insérer à la position 2 juste avant `(5, 'Chrome')` ;
- l'élément `(0, 'Safari')` on devrait l'insérer à la position 4 c'est-à-dire l'ajouter à la fin de la liste.

Pour déterminer la position d'insertion d'un nouvel élément on adapte la méthode dite de recherche dichotomique dans une liste triée dans l'ordre décroissant des numéros de priorités (voir la méthode `index_insertion_element`).

On compare la priorité du tuple `element` à la priorité du tuple se situant au milieu de la `liste_priorite`.

- si elle est strictement supérieure on recommence dans la moitié gauche de `liste_priorite` ;
- si elle est strictement inférieure on recommence dans la moitié droite de `liste_priorite` ;
- si elle est égale la position devra être le milieu.

9. Donner le coût en temps de la recherche dichotomique.

10. Recopier et compléter les huit lignes 24, 29, 30, 31, 32, 33, 36, et 37 du code de la méthode `index_insertion_element` qui prend en paramètre un élément `element` et qui renvoie la position d'insertion de l'élément `element` en utilisant une méthode dichotomique.

11. Écrire, sans utiliser la méthode `insert` des listes Python, une méthode `inserer` qui prend en paramètre un élément `element`, et modifie `liste_priorite` en insérant l'élément `element` à la position adéquate de la liste triée par ordre décroissant des numéros de priorités.

    Exemples :

```
>>> navigateurs.liste_priorite
[(10, 'Edge'), (8, 'Firefox'), (5, 'Chrome'), (1,
'Vivaldi')]
>>> navigateurs.inserer((16, 'Brave'))
>>> navigateurs.liste_priorite
[(16, 'Brave'), (10, 'Edge'), (8, 'Firefox'), (5,
'Chrome'), (1, 'Vivaldi')]
>>> navigateurs.inserer((6, 'Safari'))
>>> navigateurs.liste_priorite
[(16, 'Brave'), (10, 'Edge'), (8, 'Firefox'),(6,
'Safari'), (5, 'Chrome'), (1, 'Vivaldi')]
>>> navigateurs.inserer((0, 'Lynx'))
>>> navigateurs.liste_priorite
[(16, 'Brave'), (10, 'Edge'), (8, 'Firefox'),(6,
'Safari'), (5, 'Chrome'), (1, 'Vivaldi'), (0, 'Lynx')]
```

Exercice 3 (8 points)

Cet exercice porte principalement sur les systèmes d'exploitation, les réseaux et la programmation de base en Python.

Une association de jardinage anime un réseau d'échange de plantes.

Partie A

Les échanges de plantes sont traités à travers un système de gestion de fichiers dans un espace de stockage partagé.

Le répertoire `association` est positionné à la racine du système de fichiers.

Un répertoire `annonces` contient des fichiers au format `HTML` décrivant l'échange souhaité. Le fichier contient notamment le nom de la personne qui soumet l'annonce et décrit la plante proposée ainsi que le type de plante souhaitée en échange. Chaque adhérent dispose d'un répertoire à son nom à l'intérieur du répertoire `adherents`.

La figure ci-dessous donne un extrait de l'arborescence :

```
|---+ association
    |---+ adherents
    |   |---+ abi
    |   |   |--tulipe_chen.html
    |   |   |--muguet_abi.html
    |   |
    |   |---+ bachir
    |   |   |--reseda_dana.html
    |   |
    |   |  [...]
    |
    |---+ annonces
    |   |--rosier_abi.html
    |   |--pivoine_bachir.html
```

L'utilisateur manipule le système de fichiers à travers un terminal (en ligne de commande). Il travaille sur un système de type Unix.

1. Décrire le résultat de la commande `ls /association` saisie dans un terminal.

2. Le répertoire courant dans le terminal est `/association`. Parmi les commandes systèmes suivantes, identifier celle qui permet de définir `annonces` comme répertoire courant.

- `cd /annonces`
- `cd ../annonces`
- `cd annonces`
- `cd association/annonces`

Les quatre premières lignes de la documentation obtenue avec la commande `cp --help` sont les suivantes :

```
Utilisation : cp [OPTION]... [-T] SOURCE DEST
         ou : cp [OPTION]... SOURCE... RÉPERTOIRE
         ou : cp [OPTION]... -t RÉPERTOIRE SOURCE...
Copier la SOURCE vers DEST ou plusieurs SOURCEs vers
RÉPERTOIRE.
```

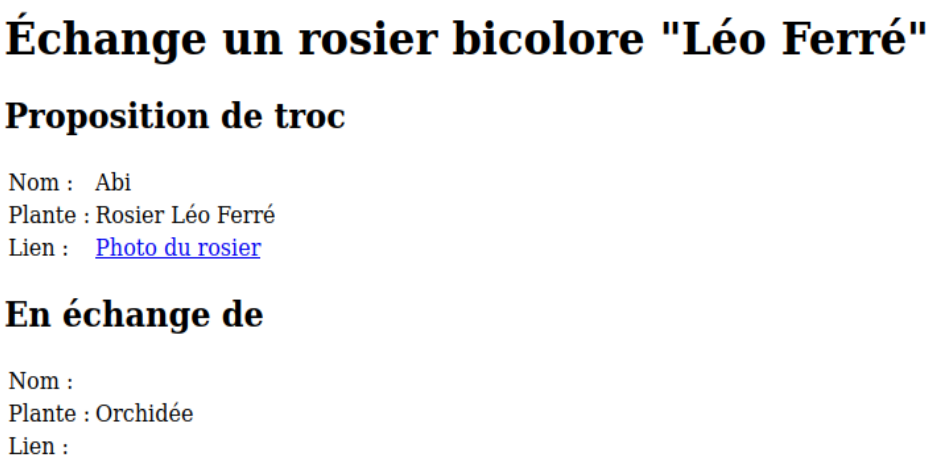
Les quatre premières lignes du résultat de `mv --help` sont :

```
Utilisation : mv [OPTION]... [-T] SOURCE DEST
         ou : mv [OPTION]... SOURCE... RÉPERTOIRE
         ou : mv [OPTION]... -t RÉPERTOIRE SOURCE...
Renommer SOURCE en DEST, ou déplacer le ou les SOURCEs vers
RÉPERTOIRE.
```

3. Expliquer la différence entre les commandes suivantes en précisant le nombre d'exemplaires du fichier `rosier_abi.html` à l'issue de chacune d'elles.

   - `cp rosier_abi.html ../adherents/abi/`
   - `mv rosier_abi.html ../adherents/abi/`

Abi propose une annonce à laquelle Bachir voudra répondre. L'annonce d'Abi figure dans la page `rosier_abi.html` qui s'affiche ainsi dans un navigateur :



Échange un rosier bicolore "Léo Ferré"

Proposition de troc

Nom : Abi
Plante : Rosier Léo Ferré
Lien : Photo du rosier

En échange de

Nom :
Plante : Orchidée
Lien :

Figure 1. rosier_abi.html

Bachir devra répondre à l'annonce dans le code `HTML` de la page `rosier_abi.html` ci-dessous.

```
<html><body>
<h1>Échange un rosier bicolore "Léo Ferré"</h1>

<h2>Proposition de troc</h2>
<table>
    <tr><td>Nom :</td> <td>Abi</td></tr>
    <tr><td>Plante :</td><td>Rosier Léo Ferré</td></tr>
    <tr><td>Lien : </td><td><a
href="monsiteperso.fr/abi/photo_rosier.jpg">Photo du
rosier</a></td></tr>
</table>

<h2>En échange de</h2>
<table>
    <tr><td>Nom :</td><td></td></tr>
    <tr><td>Plante :</td><td>Orchidée</td></tr>
    <tr><td>Lien : </td><td></td></tr>
</table>
</body></html>
```

Pour répondre à l'annonce Bachir doit écrire au bon endroit dans le code source de la page `HTML` les informations suivantes :

- Nom : Bachir
- Plante : Orchidée noire Cymbidium
- Lien : `monsiteperso.fr/bachir/orchidee.jpg`

4. Recopier et compléter la section de code que Bachir doit modifier pour répondre à l'annonce.

Lorsqu'une personne veut répondre à l'annonce, elle procède selon l'algorithme suivant :

- Étape 1 : elle ouvre la page `HTML` dans un éditeur ;
- Étape 2 : elle ajoute les informations nécessaires à l'échange ;
- Étape 3 : elle enregistre les modifications ;
- Étape 4 : elle fait une copie de l'annonce complétée vers son répertoire personnel ;
- Étape 5 : l'exemplaire original modifié est déplacé vers le répertoire personnel de la personne qui a proposé l'annonce.

À l'issue de cet algorithme, l'annonce a disparu du répertoire `annonces`.

5. Bachir et Chen veulent tous les deux répondre à l'annonce. Ils exécutent chacun l'algorithme sensiblement à la même heure.

   En détaillant étape par étape un exemple, expliquer pourquoi cet algorithme se comporte mal avec la mise en concurrence des deux propositions d'échange.

6. Proposer un nouvel algorithme qui empêche le conflit précédent.

Partie B

Les membres de l'association s'organisent en réseau afin de s'échanger les plantes de la main à la main.

Certains membres de l'association se rencontrent très régulièrement. Dans ce cas, on dira qu'ils sont amis. Quand un membre de l'association doit faire parvenir une plante, il la confie à un ami qui lui-même la confiera à quelqu'un d'autre, à l'image des routeurs au cœur d'Internet qui se transmettent les messages à router.

Pour acheminer au mieux les plantes, l'association s'inspire du protocole RIP. Ce protocole de routage s'adapte aux modifications du réseau.

On rappelle que le protocole RIP vise à minimiser le nombre de sauts sur les chemins de routage construits. Dans le contexte de l'exercice, il s'agira de minimiser le nombre d'échanges entre adhérents pour faire parvenir la plante à son destinataire.

Les membres sont donc amenés à servir d'intermédiaire. Dans ce cadre, chaque membre construit l'équivalent d'une table de routage.

La table de routage d'Abi est la suivante :

Table de routage d'Abi		
Destinataire	Intermédiaire	Distance
Bachir	Bachir	1
Chen	Bachir	2
Dana	Dana	1
Edie	Dana	2

D'après cette table, Abi est amie avec Bachir. Si Abi doit faire parvenir une plante à Chen, elle doit la confier à Bachir car il est un ami de Chen. La distance correspond au nombre de rencontres nécessaires pour faire parvenir la plante.

Exemple : Dans le tableau, on lit que la distance entre Abi et Chen vaut 2 car si Abi veut faire parvenir un rosier à Chen, il faut que Abi confie le rosier à Bachir puis que Bachir le donne à Chen. Edie n'est pas un ami d'Abi, son nom ne peut pas apparaitre dans la colonne « Intermédiaire ».

7. Reproduire le graphe ci-dessous puis ajouter une arête pour que le graphe devienne cohérent avec la table de routage d'Abi.

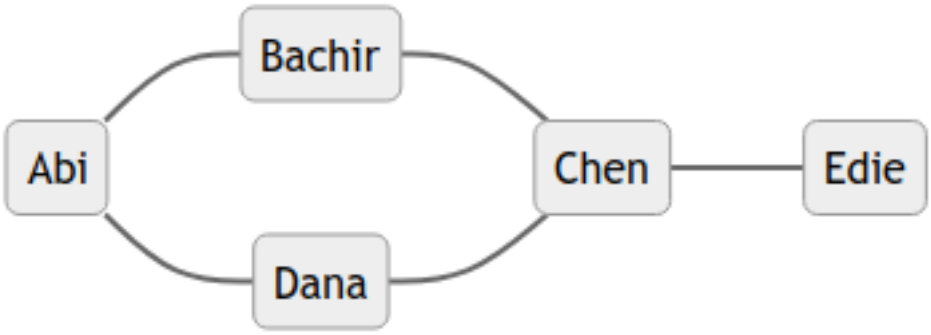


Figure 2. Relations entre les membres de l'association

Frida rejoint l'association. Elle est amie avec Abi. À son arrivée dans l'association, Frida ne connaît personne d'autre. Pour construire sa table de routage, Frida exploite les informations de la table de routage d'Abi.

8. Reproduire et compléter la table de routage de Frida.

Table de routage de Frida		
Destinataire	Intermédiaire	Distance
Abi	Abi	1
Bachir		
Chen		
Dana		
Edie		

Abi met également sa table de routage à jour pour y inclure Frida, puis elle va communiquer sa table de routage à tous ses autres amis (Bachir et Dana). Par la suite, Bachir et Dana, à leur tour, vont communiquer leur table de routage à leurs amis, et ainsi de suite.

9. Décrire les modifications que doivent faire Abi, Bachir, Chen, Dana et Edie dans leur table de routage respective suite à l'arrivée de Frida.

Après quelques semaines les relations ont évolué, certains liens se sont rompus et un nouvel adhérent est arrivé. Abi et Frida échangent les informations de leur table de routage respective afin de les actualiser.

Table de routage d'Abi		
Destinataire	Intermédiaire	Distance
Bachir	Bachir	1
Chen	Bachir	3
Dana	Dana	1
Edie	Dana	2
Frida	Frida	1
Guy	Dana	3
Hakim	Hakim	1

Table de routage de Frida		
Destinataire	Intermédiaire	Distance
Abi	Abi	1
Bachir	Abi	2
Chen	Chen	1
Dana	Abi	2
Edie	Guy	2
Guy	Guy	1

10. Décrire les modifications que chacune d'elles doit apporter à sa table de routage afin de maintenir les routes les plus courtes pour chaque destinataire.

Partie C

Dans cette partie, les tables de routage sont structurées sous forme de dictionnaires. Les clés du dictionnaire sont les destinataires. La valeur associée à un destinataire est le tuple `(intermediaire, distance)`.

La table de routage d'Abi est la suivante :

```
1 table_abi = {'Abi'      : ('Abi'    , 0),
2              'Bachir'   : ('Bachir' , 1),
```

```
3              'Chen'     : ('Bachir' , 3),
4              'Dana'     : ('Dana'   , 1),
5              'Edie'     : ('Dana'   , 2),
6              'Guy'      : ('Dana'   , 3),
7              'Hakim'    : ('Hakim'  , 1)}
```

La table de routage d'Hakim est la suivante :

```
1 table_hakim = {'Ines'    : ('Janus'  , 2),
2                'Janus'   : ('Janus'  , 1)}
```

11. En vous appuyant sur la table de routage d'Hakim, répondre par vrai ou faux à chacune des affirmations suivantes :
    - Hakim et Janus sont amis ;
    - Hakim et Ines sont amis ;
    - Janus et Ines sont amis.

La fonction `amis` prend en argument une table de routage d'une personne et elle renvoie la liste de ses amis, c'est-à-dire la liste des intermédiaires, sans doublon. La fonction a été correctement programmée, mais par erreur les lignes de codes ont été mélangées (elles ont été triées dans l'ordre alphanumérique croissant). Les espaces en début de ligne ont été conservés et ils sont donc corrects, seul l'ordre des lignes a été modifié.

```
1             liste.append(intermediaire)
2         if intermediaire not in liste:
3     """renvoie la liste des intermédiaires de la table de
routage, sans doublon"""
4     for (intermediaire, distante) in table.values():
5     liste = []
6     return liste
7 def amis(table):
```

12. Remettre les lignes de codes dans le bon ordre.

Dans le but d'automatiser la mise à jour des tables de routages, il est proposé le programme suivant :

```
1 def maj(ma_table, ami, table_ami):
2     """mise à jour de ma_table (dict) avec les
3     informations de table_ami (dict).
4     ami est du type str."""
5
6     if ami not in ma_table.keys():
7         ma_table[ami] = ( ... , ... )
8
```

```
 9     for adh in table_ami.keys():
10         (intermediaire, distance)  = table_ami[adh]
11         if adh not in ma_table.keys():
12             ma_table[adh] = (intermediaire, distance + 1)
13         if ma_table[adh][...] > distance + 1:
14             ma_table[adh] = (ami, ...)
15
16     for adh in ma_table.keys():
17         (intermediaire, distance) = ma_table[adh]
18         if adh not in table_ami.keys() and adh != ami and
intermediaire == ami:
19             ma_table[adh] = (None, float('inf'))
20
21     return ma_table
```

La fonction `maj` prends en paramètres :

- un dictionnaire `ma_table` représentant une table de routage ;
- une chaîne de caractères `ami` désignant le nom de l'ami ;
- un dictionnaire `table_ami` représentant la table de routage de l'ami. C'est de cette table de routage que sont extraites les informations utiles.

Cette fonction met à jour la table de routage `ma_table`.

13. Expliquer le test de la ligne 6 du code de la fonction `maj`.
14. Recopier et compléter la ligne 7 du code de la fonction `maj`.
15. La boucle de la ligne 9 parcourt les clés de `table_ami` pour mettre à jour les informations de `ma_table`. Recopier et compléter les lignes 13 et 14 du code de la fonction `maj`.

La fonction `nettoie`, dont le code est donné ci-après, prend en paramètre un dictionnaire `table` représentant une table de routage. Elle a pour objectif de supprimer les entrées du dictionnaire des membres devenus injoignables (suite à la mise-à-jour de la table dans la fonction `maj` aux lignes 16 à 19).

```
1 def nettoie(table):
2     """Supprime toute les noms qui ne sont pas joignables"""
3     for (adh, ligne) in table.items():
4         if ligne[0] == None:
5             del table[adh]
```

Il est impossible, avec le langage Python, de supprimer des entrées dans un dictionnaire à l'intérieur d'une boucle qui le parcourt. C'est la raison pour laquelle, à l'exécution de cette fonction avec une certaine table, on obtient l'erreur suivante :

```
Traceback [...]line 1, in nettoie
    for (adh, ligne) in table.items():
RuntimeError: dictionary changed size during iteration
```

16. Proposer une version corrigée de la fonction `nettoie` qui évite le déclenchement de l'erreur décrite.