Correction sujet 0B 2024

Exercice 1

Cet exercice porte sur la notion de listes, la récursivité et la programmation dynamique.

Pour extraire de l'eau dans des zones de terrain instable, on souhaite forer un conduit dans le sol pour réaliser un puits tout en préservant l'intégrité du terrain. Pour représenter cette situation, on va considérer qu'en forant à partir d'une position en surface, on s'enfonce dans le sol en allant à gauche ou à droite à chaque niveau, jusqu'à atteindre le niveau de la nappe phréatique.

Le sol pourra donc être représenté par une pyramide d'entiers où chaque entier est le *score de confiance* qu'on a dans le forage de la zone correspondante. Une telle pyramide est présentée sur la figure 1, à gauche, les flèches indiquant les différents déplacements possibles d'une zone à une autre au cours du forage.

Un conduit doit partir du sommet de la pyramide et descendre jusqu'au niveau le plus bas, où se situe l'eau, en suivant des déplacements élémentaires, c'est-à-dire en choisissant à chaque niveau de descendre sur la gauche ou sur la droite. Le score de confiance d'un conduit est la somme des nombres rencontrés le long de ce conduit. Le conduit gris représenté à droite sur la figure 1 a pour score de confiance $4+2+5+1+3=15$.

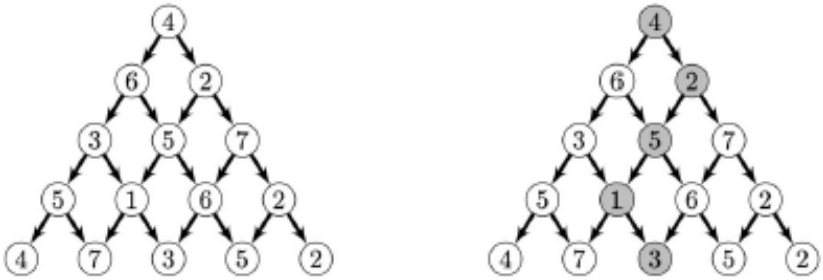


Figure 1.

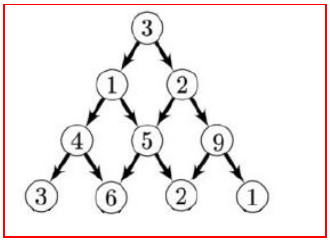
On va utiliser un ordinateur pour chercher à résoudre ce problème. Pour cela, on représente chaque niveau par la liste des nombres de ce niveau et une pyramide par une liste de niveaux.

La pyramide ci-dessus est donc représentée par la liste de listes

```
ex1 = [[4],[6,2],[3,5,7],[5,1,6,2],[4,7,3,5,2]].
```

1. Dessiner la pyramide représentée par la liste de listes

```
ex2 = [[3],[1,2],[4,5,9],[3,6,2,1]].
```



2. Déterminer un conduit de score de confiance maximal dans la pyramide `ex2` et donner son score.

Le score de confiance maximal est 16 avec les chemins :

3+2+5+6 = 16 ou 3+2+9+2 = 16

On souhaite déterminer le score de confiance maximal pouvant être atteint pour une pyramide quelconque. Une première idée consiste à énumérer tous les conduits et à calculer leur score pour déterminer les meilleurs.

3. Énumérer les conduits dans la pyramide de trois niveaux représentée sur la figure 2.

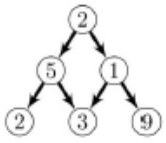


Figure 2.

Les conduits possibles sont : 2−5−2 ou 2−5−3 ou 2−1−3 ou 2−1−9

Afin de compter le nombre de conduits pour une pyramide de `n` niveaux, on remarque qu'un conduit est uniquement représenté par une séquence de `n` déplacements `gauche` ou `droite`.

4. En considérant un codage binaire d'un tel conduit, où `gauche` est représenté par `0` et `droite` par `1`, déterminer le nombre de conduits dans une pyramide de `n` niveaux.

Un conduit est représenté par un mot contenant des 0 ou des 1. On décide si on va à droite ou à gauche pour les niveaux de 1 à *n*−1. Quand on arrive au niveau *n*, on ne fait pas de choix.

Pour représenter les conduits dans une pyramide de 3 niveaux on peut utiliser un nombre écrit en binaire : 11 (à gauche puis à gauche) ; 10 (à droite puis à gauche) ; 01 (à gauche puis à droite) ; 00 (à droite puis à droite).

Le mot contient donc *n* −1 chiffres (0 ou 1). Il y a 2^{n-1} mots possibles donc 2^{n-1} conduits possibles.

5. Justifier que la solution qui consiste à tester tous les conduits possibles pour calculer le score de confiance maximal d'une pyramide n'est pas raisonnable.

Tester tous les conduits reviendrait à tester 2^{n-1} conduits. À chaque fois que l'on augmente la hauteur d'une unité, on double le nombre de conduits possibles La complexité est donc exponentielle en $O(2^n)$. Ce n'est pas raisonnable.

On dira dans la suite qu'un conduit est maximal si son score de confiance est maximal. Afin de pouvoir calculer efficacement le score maximal, on peut analyser la structure des conduits maximaux.

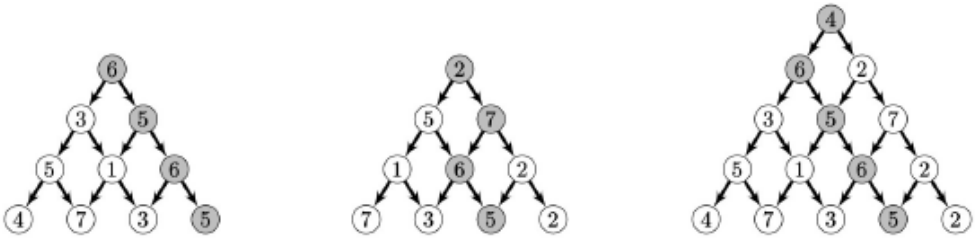


Figure 3.

- **Première observation :** si on a des conduits maximaux `cm1` et `cm2` (représentés en gris dans la figure 3) pour les deux pyramides obtenues en enlevant le sommet de `ex1`, on obtient un conduit maximal en ajoutant le sommet `4` devant le conduit de plus grand score parmi `cm1` et `cm2`. Ici le score de `cm1` est `6+5+6+5=22` et le score de `cm2` est `2+7+6+5=20` donc le conduit maximal dans `ex1` est celui obtenu à partir de `cm1` et dessiné à droite dans la figure 3.
- **Deuxième observation :** si la pyramide n'a qu'un seul niveau, il n'y a que le sommet, dans ce cas, il n'y a pas de choix à faire, le seul conduit possible est celui qui contient le sommet et le nombre de ce sommet est le score maximal que l'on peut obtenir.

Avec ces deux observations, on peut calculer le score maximal possible pour un conduit dans une pyramide `p` par récurrence. Posons `score_max(i,j)` le score maximal possible depuis le nombre d'indice `j` du niveau `i`, c'est-à-dire dans la petite pyramide issue de ce nombre. On a alors les relations suivantes :

- `score_max(len(p)-1,j,p) = p[len(p)-1][j] ;`
- `score_max(i,j,p) = p[i][j] + max(score_max(i+1,j,p),score_max(i+1,j+1,p)).`

Le score maximal possible pour `p` toute entière sera alors `score_max(0,0,p)`.

6. Écrire la fonction récursive `score_max` qui implémente les règles précédentes.

```
def score_max(i,j,p):
    if i == len(p) - 1 :
        return p[len(p) - 1][ j ]
    else :
        return p[ I ][ j ] + max(score_max(i+1, j, p), score_max(i+1, j+1, p))
```

Si on suit à la lettre la définition de `score_max`, on obtient une résolution dont le coût est prohibitif à cause de la redondance des calculs. Par exemple `score_max(3,1,p)` va être calculé pour chaque appel à `score_max(2,0,p)` et `score_max(2,1,p)`. Pour éviter cette redondance, on décide de mettre en place une approche par programmation dynamique. Pour cela, on va construire une pyramide `s` dont le nombre à l'indice `j` du niveau `i` correspond à `score_max(i,j,p)`, c'est-à-dire au score maximal pour un conduit à partir du nombre correspondant dans `p`.

7. Écrire une fonction `pyramide_nulle` qui prend en paramètre un entier `n` et construit une pyramide remplie de `0` à `n` niveaux.

```
def pyramide_nulle(n):
    pyr = [ ]
    for i in range(1,n+1):
        pyr.append([0]*i)
    return pyr
```

8. Compléter la fonction `prog_dyn` ci-dessous qui prend en paramètre une pyramide `p`, et qui renvoie le score maximal pour un conduit dans `p`. Pour cela, on construit une pyramide `s` remplie de `0` de la même taille et la remplit avec les valeurs de `score_max` en commençant par le dernier niveau et en appliquant petit à petit les relations données ci-dessus.

Exemple :

Avec la pyramide :

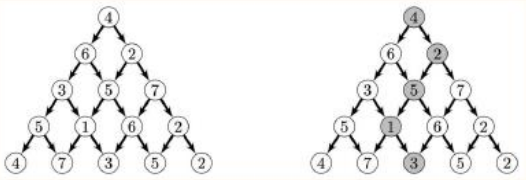
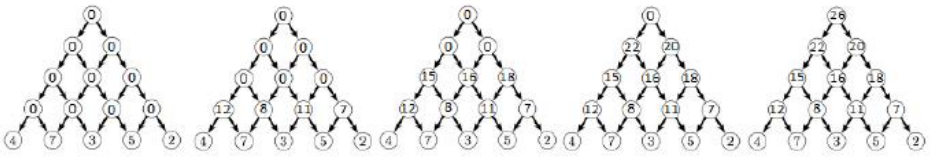


FIGURE 1 –

On représente le remplissage de la pyramide `s` au fur et à mesure.



```
def prog_dyn(p):
    n = len(p)
    s = pyramide_nulle(n)
    # remplissage du dernier niveau
    for j in range(n):
        s[n-1][ j ] = p[n-1][ j ]
    # remplissage des autres niveaux
    for i in range(n-2,-1,-1):
        for j in range(i+1):
            s[ I ][ j ] = p[ I ][ j ] + max( s[i+1][ j ] , s[i+1][j+1] )
    # renvoie du score maximal
    return s[0][0]
```

9. Montrer que le coût d'exécution de cette fonction est quadratique en `n` pour une pyramide à `n` niveaux.

Il y a une double boucle for. On fait de 0 à *i* +1 comparaisons sur une boucle de *n* −2 à 0.

- ***i* = *n* −2, on fait de 0 à *n* −1 max (*n* opérations)**
- ***i* = *n* −3, on fait de 0 à *n* −2 max (*n* −1 opérations)**
- **...**
- ***i* = *i* , on fait de 0 à *i* max (*i* −1 opérations)**
- **...**
- ***i* = 1, on fait de 0 à 1 max (2 opérations)**
- ***i* = 0, on fait de 0 à 1 max (1 opération)**

On fait donc $1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$ **comparaisons**

L'algorithme a donc une complexité quadratique en $O(n^2)$.

10. Expliquer comment adapter la fonction `score_max` pour éviter la redondance des calculs afin d'obtenir également un coût quadratique, tout en gardant une approche récursive.

Il faut rajouter pyramide nulle appelée *score* en paramètre qui va servir de mémoire.
Si la valeur contenu de *score*[*i*][*j*] est différente de zéro, on renvoie cette valeur sinon on fait le calcul, on le mémorise et on le renvoie.

```
def score_max_dyn(i, j, p, score):
if i == len(p)-1:
    return p[len(p)-1][ j ]
if score[ i ][ j ] != 0:
    return score[ i ][ j ]
else:
    gauche = score_max_dyn(i+1, j, p, score)
    droite = score_max_dyn(i+1, j+1, p, score)
    maxi = max(gauche, droite)
    score[ i ][ j ] = p[ i ][ j ] + maxi
    return score[ i ][ j ]
pyrNulle = pyramide_nulle(len(ex1))
print(score_max_rec(0, 0, ex1, pyrNulle))
```

Exercice 2

Cet exercice porte sur les systèmes d'exploitation, les commandes UNIX, les structures de données (de type LIFO et FIFO) et les processus.

"Linux ou GNU/Linux est une famille de systèmes d'exploitation open source de type Unix fondée sur le noyau Linux, créé en 1991 par Linus Torvalds. De nombreuses distributions Linux ont depuis vu le jour et constituent un important vecteur de popularisation du mouvement du logiciel libre."

Source : Wikipédia, extrait de l'article consacré à GNU/Linux.

"Windows est au départ une interface graphique unifiée produite par Microsoft, qui est devenue ensuite une gamme de systèmes d'exploitation à part entière, principalement destinés aux ordinateurs compatibles PC. Windows est un système d'exploitation propriétaire."

Source : Wikipédia, extrait de l'article consacré à Windows.

1. Expliquer succinctement les différences entre les logiciels libres et les logiciels propriétaires.

• Logiciels libres : Ces logiciels permettent aux utilisateurs de les utiliser, de les étudier, de les modifier et de les redistribuer. Ils sont souvent développés par une communauté d'utilisateurs et sont généralement gratuits. Les utilisateurs ont accès au code source, ce qui leur permet d'apporter des modifications pour répondre à leurs besoins spécifiques.

• Logiciels propriétaires : Ces logiciels sont la propriété d'individus ou d'entreprises. Les utilisateurs doivent généralement payer (redevance) pour les utiliser et n'ont pas accès au code source. Ils ne peuvent pas modifier le logiciel et la redistribution est généralement interdite.

En résumé, la différence clé réside dans les libertés accordées aux utilisateurs. Les logiciels libres offrent plus de liberté, tandis que les logiciels propriétaires ont tendance à avoir des restrictions plus strictes.

2. Expliquer le rôle d'un système d'exploitation.

Un système d'exploitation (OS) est un logiciel qui sert d'intermédiaire entre l'utilisateur d'un ordinateur et le matériel informatique. Voici quelques-uns de ses rôles principaux :

(a)Gestion du matériel : L'OS contrôle et coordonne l'utilisation du matériel parmi les différents programmes d'application et les utilisateurs.

(b)Organisation des fichiers : Il gère les fichiers sur les dispositifs de stockage, ce qui implique leur stockage, leur récupération et leur recherche.

(c)Gestion des tâches : L'OS est responsable de l'ordonnancement des tâches et de leur exécution, ainsi que de la gestion des ressources que les tâches peuvent nécessiter.

(d)Sécurité : Il assure la sécurité des informations stockées et des ressources du système.

(e)Interface utilisateur : L'OS fournit une interface pour les utilisateurs pour interagir avec le système. Cette interface peut être une ligne de commande (CLI) ou une interface graphique (GUI).

On donne ci-dessous une arborescence de fichiers sur un système GNU/Linux (les noms encadrés représentent des répertoires, les noms non encadrés représentent des fichiers, / correspond à la racine du système de fichiers) :

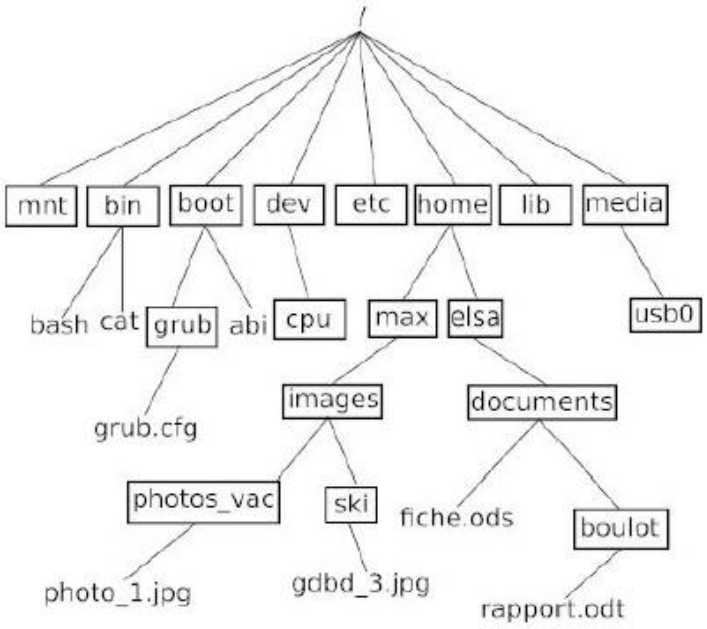


Figure 1. Arborescence de fichiers

3. Indiquer le chemin absolu du fichier `rapport.odt`.

Le chemin absolu du fichier rapport.odt dans l'arborescence de fichiers donnée est :
/home/elsa/documents/boulot/rapport.odt

On suppose que le répertoire courant est le répertoire `elsa`.

4. Indiquer le chemin relatif du fichier `photo_1.jpg`.

Le chemin relatif du fichier photo_1.jpg est :
../max/images/photo_vac/photo_1.jpg

L'utilisatrice Elsa ouvre une console (aussi parfois appelée terminal), le répertoire courant étant toujours le répertoire `elsa`. On fournit ci-dessous un extrait du manuel de la commande UNIX `cp` :

```
NOM
    cp - copie un fichier

UTILISATION
    cp fichier_source fichier_destination
```

5. Déterminer le contenu du répertoire `documents` et du répertoire `boulot` après avoir exécuté la commande suivante dans la console :

```
cp documents/fiche.ods documents/boulot
```

Après avoir exécuté la commande dans la console, le fichier fiche.ods est copié dans le répertoire boulot sous le même nom.
Donc le répertoire documents ne change pas de contenu (fichier fiche.ods et répertoire boulot). Le répertoire boulot contient maintenant deux fichiers, fiche.ods et rapport.odt.

"Un système d'exploitation est multitâche (en anglais : multitasking) s'il permet d'exécuter, de façon apparemment simultanée, plusieurs programmes informatiques. GNU/Linux, comme tous les systèmes d'exploitation modernes, gère le multitâche."

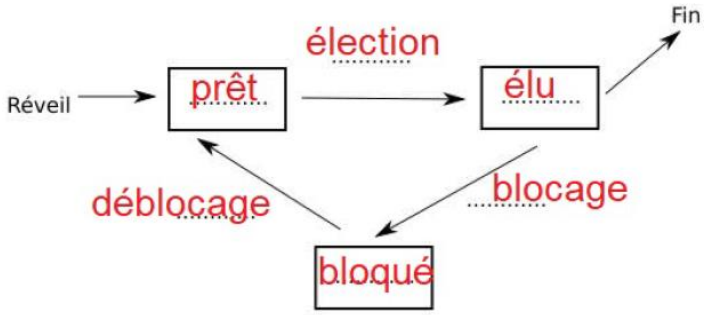
"Dans le cas de l'utilisation d'un monoprocesseur, la simultanéité apparente est le résultat de l'alternance rapide d'exécution des processus présents en mémoire."

Source : Wikipédia, extraits de l'article consacré au Multitâche.

Dans la suite de l'exercice, on s'intéresse aux processus. On considère qu'un monoprocesseur est utilisé. On rappelle qu'un processus est un programme en cours d'exécution. Un processus est soit élu, soit bloqué, soit prêt.

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

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



7. Donner l'exemple d'une situation qui contraint un processus à passer de l'état élu à l'état bloqué.

Attente d'une ressource (accès imprimante ou accès fichier) : Supposons qu'un processus en cours d'exécution a besoin d'accéder à un fichier sur le disque dur pour lire des données mais ce fichier est actuellement utilisé par un autre processus qui l'a verrouillé pour écriture. Le processus en cours d'exécution ne peut pas continuer tant qu'il n'a pas accès au fichier. Il passe donc de l'état élu (en train d'utiliser le microprocesseur) à l'état bloqué (en attente de la libération du fichier). Une fois que le fichier est disponible, le processus peut reprendre son exécution.

"Dans les systèmes d'exploitation, l'ordonnanceur est le composant du noyau du système d'exploitation choisissant l'ordre d'exécution des processus sur le processeur d'un ordinateur."

Source : Wikipédia, extrait de l'article consacré à l'ordonnancement.

L'ordonnanceur peut utiliser plusieurs types d'algorithmes pour gérer les processus.

L'algorithme d'ordonnancement par "ordre de soumission" est un algorithme de type FIFO (First In First Out), il utilise donc une file.

8. Nommer une structure de données linéaire de type LIFO (Last In First Out).

Une structure de données linéaire de type LIFO (Last In, First Out) est communément appelée « pile ». Dans une pile, les éléments sont insérés et supprimés selon le principe du dernier arrivé, premier sorti. Cela signifie que l'élément inséré en dernier est le premier à être retiré.

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).

Par exemple, l'exécution d'un processus P4 qui a un instant d'arrivée égal à 7 et une durée d'exécution égale à 2 peut être représentée par le schéma suivant :

Figure 3. Utilisation du processeur

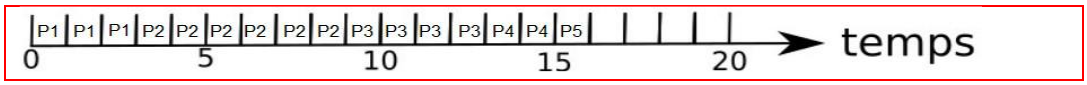
L'ordonnanceur place les processus qui ont besoin d'un accès au processeur dans une file, en respectant leur ordre d'arrivée (le premier arrivé étant placé en tête de file). Dès qu'un processus a terminé son exécution, l'ordonnanceur donne l'accès au processus suivant dans la file.

Le tableau suivant présente les instants d'arrivées et les durées d'exécution de cinq processus :

5 processus		
Processus	instant d'arrivée	durée d'exécution
P1	0	3
P2	1	6
P3	4	4
P4	6	2
P5	7	1

9. Recopier et compléter le schéma ci-dessous avec les processus P1 à P5 en utilisant les informations présentes dans le tableau ci-dessus et l'algorithme d'ordonnancement "par ordre de soumission".

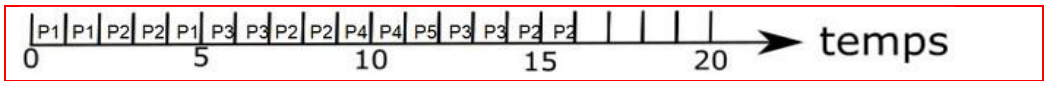
P1	P1	P1	P2	P2	P2	P2	P2	P2	P3	P3	P3	P3	P4	P4	P5					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20



On utilise maintenant un autre algorithme d'ordonnancement : l'algorithme d'ordonnancement "par tourniquet". Dans cet algorithme, la durée d'exécution d'un processus ne peut pas dépasser une durée Q appelée quantum et fixée à l'avance. Si ce processus a besoin de plus de temps pour terminer son exécution, il doit retourner dans la file et attendre son tour pour poursuivre son exécution.

Par exemple, si un processus P1 a une durée d'exécution de 3 et que la valeur de Q a été fixée à 2, P1 s'exécutera pendant deux unités de temps avant de retourner à la fin de la file pour attendre son tour ; une fois à nouveau élu, il pourra terminer de s'exécuter pendant sa troisième et dernière unité de temps d'exécution.

10. Recopier et compléter le schéma ci-dessous, en utilisant l'algorithme d'ordonnancement "par tourniquet" et les mêmes données que pour la question 9, en supposant que le quantum Q est fixé 2.



P1	P1	P2	P2	P1	P3	P3	P2	P2	P4	P4	P5	P3	P3	P2	P2					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

On considère deux processus P1 et P2, et deux ressources R1 et R2.

11. Décrire une situation qui conduit les deux processus P1 et P2 en situation d'interblocage.

Solution 1

L'interblocage, ou *deadlock* en anglais, est une situation dans laquelle deux ou plusieurs processus sont incapables de progresser car chacun attend que l'autre libère une ressource. Supposons qu'il y ait deux ressources importantes, *R*1 et *R*2, et que chaque processus a besoin des deux ressources pour terminer son exécution. Les processus *P*1 et *P*2 suivent les étapes suivantes :

(a) *P*1 acquiert la ressource *R*1.
(b) *P*2 acquiert la ressource *R*2.

Maintenant, chaque processus attend l'autre pour libérer la ressource nécessaire à son achèvement. La séquence des événements est la suivante :

- ***P*1 détient *R*1 et attend *R*2.**
- ***P*2 détient *R*2 et attend *R*1.**

Aucun des processus ne peut progresser, car chaque processus retient la ressource nécessaire pour l'autre. Cela crée une impasse, où les deux processus restent bloqués indéfiniment.

Un exemple plus concret pourrait être l'utilisation de deux imprimantes connectées à un seul ordinateur. Si *P*1 a déjà imprimé un document avec l'imprimante 1 et tente d'accéder à l'imprimante 2 pour imprimer un autre document, tandis que *P*2 a déjà utilisé l'imprimante 2 et tente d'accéder à l'imprimante 1, les deux processus peuvent se retrouver dans un état d'interblocage si les ressources ne sont pas gérées correctement.

Solution 2

Soit 2 processus P1 et P2, soit 2 ressources R1 et R2. Initialement, les 2 ressources sont "libres" (utilisées par aucun processus). Le processus P1 commence son exécution (état élu), il demande la ressource R1. Il obtient satisfaction puisque R1 est libre, P1 est donc dans l'état "prêt".

Pendant ce temps, le système a passé P2 à l'état élu : P2 commence son exécution et demande la ressource R2. La ressource R2 est libre, P2 obtient donc la ressource R2. P2 passe à l'état "prêt" et P1 passe à l'état "élu".

Au cours de son exécution, P1 a besoin de la ressource R2, la ressource R2 n'est pas libre (R2 a été attribuée à P2), P1 passe à l'état "bloqué". P2 passe à l'état "élu", au cours de son exécution, P2 a besoin de la ressource R1, la ressource R1 n'est pas disponible (toujours attribuée à P1), P2 passe à l'état "bloqué".

P1 et P2 sont alors en situation d'interblocage.

Exercice 3

Cet exercice porte sur la programmation Python (dictionnaire), la programmation orientée objet, les bases de données relationnelles et les requêtes SQL.

Cet exercice est composé de 3 parties indépendantes.

On veut créer une application permettant de stocker et de traiter des informations sur des livres de science-fiction. On désire stocker les informations suivantes :

- l'identifiant du livre (`id`) ;
- le titre (`titre`) ;
- le nom de l'auteur (`nom_auteur`) ;
- l'année de première publication (`ann_pub`) ;
- une note sur 10 (`note`).

Voici un extrait des informations que l'on cherche à stocker :

Livres de science-fiction				
id	titre	auteur	ann_pub	note
1	1984	Orwell	1949	10
2	Dune	Herbert	1965	8
14	Fondation	Asimov	1951	9
4	Ubik	K.Dick	1953	9
8	Blade Runner	K.Dick	1968	8
7	Les Robots	Asimov	1950	10
15	Ravage	Barjavel	1943	6
17	Chroniques martiennes	Bradbury	1950	7
9	Dragon déchu	Hamilton	2003	8
10	Fahrenheit 451	Bradbury	1953	8

Partie A

Dans cette première partie, on utilise un dictionnaire Python. On considère le programme suivant :

```
1 dico_livres = {
    'id' : [1, 2, 14, 4, 5, 8, 7, 15, 9, 10],
    'titre' : ['1984', 'Dune', 'Fondation', 'Ubik', 'Blade Runner',
        'Les Robots', 'Ravage', 'Chroniques martiennes',
        'Dragon déchu', 'Fahrenheit 451'],
    'auteur' : ['Orwell', 'Herbert', 'Asimov',
        'K.Dick', 'K.Dick', 'Asimov', 'Barjavel',
        'Bradbury', 'Hamilton', 'Bradbury'],
    'ann_pub' : [1949, 1965, 1951, 1953,1968,
        1950, 1943, 1950, 2003, 1953],
    'note' : [10, 8, 9, 9, 8, 10, 6, 7, 8, 8]
  }

2 a = dico_livres['note']
3 b = dico_livres['titre'][2]
```

1. Déterminer les valeurs des variables `a` et `b` après l'exécution de ce programme.

Les valeurs des variables a et b après l'exécution de ce programme sont :

- **a = [10, 8, 9, 9, 8, 10, 6, 7, 8, 8]**
- **b = 'Fondation'**

La fonction `titre_livre` prend en paramètre un dictionnaire (de même structure que `dico_livres`) et un identifiant, et renvoie le titre du livre qui correspond à cet identifiant. Dans le cas où l'identifiant passé en paramètre n'est pas présent dans le dictionnaire, la fonction renvoie `None`.

```
1 def titre_livre(dico, id_livre):
2     for i in range(len(dico['id'])):
3         if dico['id'][i] == ... :
4             return dico['titre'][...]
5     return ...
```

2. Recopier et compléter les lignes 3, 4 et 5 de la fonction `titre_livre`.

Explications :

• À la ligne 3, on vérifie si l'identifiant du livre à la position i dans le dictionnaire est égal à l'identifiant fourni en paramètre (id_livre).

• Si c'est le cas, à la ligne 4, on renvoie le titre du livre correspondant à cette position i dans le dictionnaire.

• Si on parcourt toute la liste sans trouver d'identifiant correspondant, à la ligne 5, on renvoie None pour indiquer que l'identifiant n'a pas été trouvé dans le dictionnaire.

```
def titre_livre(dico , id_livre):
    for i in range(len(dico['id'])):
        if dico['id'][i] == id_livre :
            return dico['titre'][i]
    return None
```

3. Écrire une fonction `note_maxi` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et qui renvoie la note maximale.

```
def note_maxi(dico):
    n_max = 0
    for n in dico['note']:
        if n > n_max:
            n_max = n
    return n_max
```

4. Écrire une fonction `livres_note` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et une note `n`, et qui renvoie la liste des titres des livres ayant obtenu la note `n` (on rappelle que `t.append(a)` permet de rajouter l'élément `a` à la fin de la liste `t`).

```
def livres_note(dico, n):
tab_titres = [ ]
for i in range(len(dico['note'])):
    if n == dico['note'][ i ]:
        tab_titres.append(dico[ 'titre' ][ i ])
return tab_titres
```

```
def livres_note(dico, n):
    livres_avec_note_n = []
    for i in range(len(dico['note'])):
        if dico['note'][i] == n:
            livres_avec_note_n.append(dico['titre'][i])
return livres_avec_note_n
```

5. Écrire une fonction `livre_note_maxi` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et qui renvoie la liste des titres des livres ayant obtenu la meilleure note sous la forme d'une liste Python.

```
def livre_note_maxi(dico):
    n_max = note_maxi(dico)
    return livres_note(dico, n_max)
```

```
def livre_note_maxi(dico):
    if not dico['note']:
        return [ ] # Si liste des notes vide, renvoyer liste vide
    max_note = note_maxi(dico) # on utilise la fonction de la question 3
    livres_max_note = [ ]
    for i in range(len(dico['note'])):
        if dico['note'][i] == max_note:
            livres_max_note.append(dico['titre'][ i ])
    return livres_max_note
```

Partie B

Dans cette partie, on utilise le paradigme orientée objet (POO). On propose deux classes : `Livre` et `Bibliotheque`.

```
1  class Livre:
2      def __init__(self, id_livre, titre, auteur, ann_pub, note):
3          self.id = id_livre
4          self.titre = titre
5          self.auteur = auteur
6          self.ann_pub = ann_pub
7          self.note = note
8      def get_id(self):
9          return self.id
10     def get_titre(self):
11         return self.titre
12     def get_auteur(self):
13         return self.auteur
14     def get_ann_pub(self):
15         return self.ann_pub
16
17 class Bibliotheque:
18     def __init__(self):
19         self.liste_livre = []
20     def ajout_livre(self, livre):
21        self.liste_livre.append(livre)
22     def titre_livre(self, id_livre):
23         for livre in self.liste_livre :
24             if ... == id_livre :
25                 return ...
26         return ...
```

6. Citer un attribut et une méthode de la classe `Livre`.

Dans la classe Livre :

- **un attribut : self.id ;**
- **une méthode : get_id() ;**

7. Écrire la méthode `get_note` de la classe `Livre`. Cette méthode devra

```
def get_note(self):
        return self.note
```

8. Écrire le programme permettant d'ajouter le livre Blade Runner à la fin de la "bibliothèque" en utilisant la classe `Livre` et la classe `Bibliotheque` (voir le tableau en début d'exercice).

```
livre_blade_runner = Livre(8, 'Blade Runner', 'K.Dick', 1968,8)
biblio = Bibliotheque( )
biblio.ajout_livre(livre_blade_runner)
```

9. Recopier et compléter la méthode `titre_livre` de la classe `Bibliotheque`. Cette méthode prend en paramètre l'identifiant d'un livre et renvoie le titre du livre si l'identifiant existe, ou `None` si l'identifiant n'existe pas.

```
def titre_livre(self, id_livre):
    for livre in self.liste_livre :
        if livre.get_id() == id_livre :
            return livre.get_titre()
    return None
```

Partie C

On utilise maintenant une base de données relationnelle. Les commandes nécessaires ont été exécutées afin de créer une table `livres`. Cette table `livres` contient toutes les données sur les livres. On obtient donc la table suivante :

livres				
id	titre	auteur	ann_pub	note
1	1984	Orwell	1949	10
2	Dune	Herbert	1965	8
14	Fondation	Asimov	1951	9
4	Ubik	K.Dick	1953	9
8	Blade Runner	K.Dick	1968	8
7	Les Robots	Asimov	1950	10
15	Ravage	Barjavel	1943	6
17	Chroniques martiennes	Bradbury	1950	7
9	Dragon déchu	Hamilton	2003	8
10	Fahrenheit 451	Bradbury	1953	8

L'attribut `id` est la clé primaire pour la table `livres`.

10. Expliquer pourquoi l'attribut `auteur` ne peut pas être choisi comme clé primaire.

La clé primaire doit être unique. Un auteur peut écrire plusieurs livres, l'attribut `auteur` ne peut donc pas être une clé primaire.

11. Donner le résultat renvoyé par la requête SQL suivante :

```
SELECT titre
FROM livres
WHERE auteur = 'K.Dick';
```

Ubik
Blade Runner

12. Écrire une requête SQL permettant d'obtenir les titres des livres écrits par Asimov publiés après 1950.

```
SELECT titre
FROM livres
WHERE auteur = 'Asimov' AND ann_pub > 1950;
```

13. Écrire une requête SQL permettant de modifier la note du livre Ubik en la passant de 9/10 à 10/10.

```
UPDATE livres
SET note = 10
WHERE titre = 'Ubik';
```

On souhaite proposer plus d'informations sur les auteurs des livres. Pour cela, on crée une deuxième table `auteurs` avec les attributs suivants :

- `id` de type INT ;
- `nom` de type TEXT ;
- `prenom` de type TEXT ;
- `annee_naissance` de type INT (année de naissance).

auteurs			
id	nom	prenom	annee_naissance
1	Orwell	George	1903
2	Herbert	Franck	1920
3	Asimov	Isaac	1920
4	K.Dick	Philip	1928
5	Bradbury	Ray	1920
6	Barjavel	René	1911
7	Hamilton	Peter	1960

La table `livres` est aussi modifiée comme suit :

livres				
id	titre	id_auteur	ann_pub	note
1	1984	1	1949	10
2	Dune	2	1965	8
14	Fondation	3	1951	9
4	Ubik	4	1953	9
8	Blade Runner	4	1968	8
7	Les Robots	3	1950	10
15	Ravage	6	1943	6
17	Chroniques martiennes	5	1950	7
9	Dragon déchu	7	2003	8
10	Fahrenheit 451	5	1953	8

14. Expliquer l'intérêt d'utiliser deux tables (`livres` et `auteurs`) au lieu de regrouper toutes les informations dans une seule table.

Cela permet d'éviter la duplication des données : évite de réécrire l'ensemble des informations personnelles d'un auteur pour chacun de ses livres.

15. Expliquer le rôle de l'attribut `id_auteur` de la table `livres`.

`id_auteur` est une clé étrangère, elle permet d'établir une relation entre la table `auteurs` et la table `livres` (jointure). À chaque valeur de l'attribut `id_auteur` correspond un auteur dans la table `auteurs`.

16. Écrire une requête SQL qui renvoie le nom et le prénom des auteurs des livres publiés après 1960.

```
SELECT nom, prenom
FROM auteurs
JOIN livre ON id_auteur = auteurs.id
WHERE ann_pub > 1960;
```

17. Décrire par une phrase en français le résultat de la requête SQL suivante :

```
SELECT titre
FROM livres
JOIN auteurs ON id_auteur = auteurs.id
WHERE ann_pub - annee_naissance < 30;
```

Cette requête permet d'obtenir le titre des livres écrits par des auteurs ayant moins de 30 ans au moment de leur publication.

Un élève décide de créer une application d'annuaire pour sa classe. On pourra retrouver, grâce à cette application, différentes informations sur les élèves de la classe : nom, prénom, date de naissance, numéro de téléphone, adresse email, etc.

18. Expliquer en quoi la réalisation de ce projet pourrait être problématique.

Création d'une base de données, contenant des données personnelles, sujette à déclaration préalable auprès de la CNIL. Ce projet ne respecte pas la protection de la vie privée : divulgation des données personnelles (date de naissance, numéro de téléphone…).