BACCALAUREAT

SESSION 2022

Épreuve de l'enseignement de spécialité

NUMERIQUE et SCIENCES INFORMATIQUES

Partie pratique

Classe Terminale de la voie générale

Sujet n°14

DUREE DE L'EPREUVE : 1 heure

Le sujet comporte 2 pages numérotées de 1 / 3 à 3 / 3
Dès que le sujet vous est remis, assurez-vous qu'il est complet.

Le candidat doit traiter les 2 exercices.

EXERCICE 1 (4 points)

On considère des mots à trous : ce sont des chaînes de caractères contenant uniquement des majuscules et des caractères `'*'`. Par exemple `'INFO*MA*IQUE'`, `'***I***E**'` et `'*S*'` sont des mots à trous.

Programmer une fonction `correspond` qui :

- prend en paramètres deux chaînes de caractères `mot` et `mot_a_trous` où `mot_a_trous` est un mot à trous comme indiqué ci-dessus,
- renvoie :
  - `True` si on peut obtenir `mot` en remplaçant convenablement les caractères `'*'` de `mot_a_trous`.
  - `False` sinon.

Exemples :

```
>>> correspond('INFORMATIQUE', 'INFO*MA*IQUE')
True

>>> correspond('AUTOMATIQUE', 'INFO*MA*IQUE')
False
```

EXERCICE 2 (4 points)

On considère au plus 26 personnes A, B, C, D, E, F ... qui peuvent s'envoyer des messages avec deux règles à respecter :

- chaque personne ne peut envoyer des messages qu'à la même personne (éventuellement elle-même),
- chaque personne ne peut recevoir des messages qu'en provenance d'une seule personne (éventuellement elle-même).

Voici un exemple - avec 6 personnes - de « plan d'envoi des messages » qui respecte les règles ci-dessus, puisque chaque personne est présente une seule fois dans chaque colonne :

- A envoie ses messages à E
- E envoie ses messages à B
- B envoie ses messages à F
- F envoie ses messages à A
- C envoie ses messages à D
- D envoie ses messages à C

Et le dictionnaire correspondant à ce plan d'envoi est le suivant :

```
plan_a = {'A':'E', 'B':'F', 'C':'D', 'D':'C', 'E':'B', 'F':'A'}
```

Sur le plan d'envoi `plan_a` des messages ci-dessus, il y a deux cycles distincts : un premier cycle avec A, E, B, F et un second cycle avec C et D.

En revanche, le plan d'envoi `plan_b` ci-dessous :

```
plan_b = {'A':'C', 'B':'F', 'C':'E', 'D':'A', 'E':'B', 'F':'D'}
```

comporte un unique cycle : A, C, E, B, F, D. Dans ce cas, lorsqu'un plan d'envoi comporte un *unique cycle,* on dit que le plan d'envoi est *cyclique*.

Pour savoir si un plan d'envoi de messages comportant `N` personnes est cyclique, on peut utiliser l'algorithme ci-dessous :

On part de la personne `A` et on inspecte les `N - 1` successeurs dans le plan d'envoi :

- Si un de ces `N - 1` successeurs est `A` lui-même, on a trouvé un cycle de taille inférieure ou égale à `N - 1`. Il y a donc au moins deux cycles et le plan d'envoi n'est pas cyclique.
- Si on ne retombe pas sur `A` lors de cette inspection, on a un unique cycle qui passe par toutes les personnes : le plan d'envoi est cyclique.

Compléter la fonction suivante en respectant la spécification.

Remarque : la fonction python `len` permet d'obtenir la longueur d'un dictionnaire.

```
def est_cyclique(plan):
    '''
    Prend en paramètre un dictionnaire plan correspondant
    à un plan d'envoi de messages entre N personnes A, B, C,
    D, E, F ...(avec N <= 26).
    Renvoie True si le plan d'envoi de messages est cyclique
    et False sinon.
    '''
    personne = 'A'
    N = len(...)
    for i in range(...):
        if plan[...] == ...:
            return ...
        else:
            personne = ...
    return ...
```

Exemples :

```
>>> est_cyclique({'A':'E', 'F':'A', 'C':'D', 'E':'B', 'B':'F', 'D':'C'})
False

>>> est_cyclique({'A':'E', 'F':'C', 'C':'D', 'E':'B', 'B':'F', 'D':'A'})
True

>>> est_cyclique({'A':'B', 'F':'C', 'C':'D', 'E':'A', 'B':'F', 'D':'E'})
True

>>> est_cyclique({'A':'B', 'F':'A', 'C':'D', 'E':'C', 'B':'F', 'D':'E'})
False
```

```python
from doctest import testmod


def correspond(mot: str, mot_a_trous: str) -> bool:
    """Est ce que mot_a_trou peut correspondre à mot ?

    Args:
        mot (str): chaîne de caractère majuscule
        mot_a_trous (str): chaîne à trous (majuscule + '*')

    Returns:
        bool: True si on peut obtenir mot en remplaçant convenablement
        les caractères '*' de mot_a_trous

    Tests et Exemples:
    >>> correspond('INFORMATIQUE', 'INFO*MA*IQUE')
    True

    >>> correspond('AUTOMATIQUE', 'INFO*MA*IQUE')
    False
    """
    n = len(mot)
    # compteur pour parcourir les chaînes 0..n-1
    i = 0
    # booléen qui indique que mot et mot_a_trous sont identiques
    est_egal = True

    # condition d'arrêt de la boucle :
    #  * le compteur i dépasse la longueur du mot
    #  * les chaînes sont incompatibles
    while not (i >= n or est_egal == False):
        # jusqu'à présent les chaînes ont été compatibles
        if mot_a_trous[i] !=  '*':
            if mot_a_trous[i] != mot[i]:
                # les caractères sont différents et
                # ce n'est pas un joker '*'
                # donc les chaines sont désormais incompatibles
                est_egal = False
        i = i + 1

    return est_egal


if __name__ == '__main__':
    testmod()
```

```python
from doctest import testmod


def est_cyclique(plan):
    '''
    Prend en paramètre un dictionnaire `plan` correspondant
    à un plan d'envoi de messages entre `N` personnes A, B, C,
    D, E, F ...(avec N <= 26).
    Renvoie True si le plan d'envoi de messages est cyclique
    et False sinon.

    Tests et Exemples:
    >>> plan_a = {'A':'E', 'B':'F', 'C':'D', 'D':'C', 'E':'B', 'F':'A'}
    >>> est_cyclique(plan_a)
    False
    >>> plan_b = {'A':'C', 'B':'F', 'C':'E', 'D':'A', 'E':'B', 'F':'D'}
    >>> est_cyclique(plan_b)
    True
    '''
    # initialisation de la clé avec la
    # valeur initiale : 'A'
    personne = 'A'
    # longueur du plan
    N = len(plan)
    # inspecter au plus les n-1 successeurs
    for i in range(N - 1):
        if plan[personne] == 'A':
            # un des successeur est 'A', il y a donc au
            # moins 2 cycles
            return False
        else:
            # prochain successeur à explorer
            personne = plan[personne]

    # tous les successeurs ont été exploré
    # et aucun n'a bouclé sur 'A'. Il n'y a donc qu'un cycle
    return True


if __name__ == '__main__':
    testmod()
```