Devoir sur table

Eléments de correction

Jeudi 12 décembre 2024

Durée de l'épreuve : 1 heure
L'usage de la calculatrice n'est pas autorisé.

Le sujet est à rendre avec le devoir

Dès que ce sujet vous est remis, assurez-vous qu'il est complet. Ce sujet comporte 10 pages numérotées de 1 à 10.

Exercice 1 – Le Tri Fusion (5 points)

Partie A : Comprendre et compléter la fonction fusion

```
def fusion(T1: list, T2: list) -> list:
    """
    Fusionne les deux tableaux triés T1 et T2
    """

    T = [0] * (len(T1) + len(T2))
    i1 = i2 = j = 0  # indices des piles

    # Tant que les deux piles sont non vides
    while i1 < len(T1) and i2 < len(T2):
        if T1[i1] < T2[i2]:
            T[j] = T1[i1]
            i1 += 1
        else:
            T[j] = T2[i2]
            i2 += 1
        j += 1

    # Une des deux piles est vide :
    # seule une des deux boucles while est exécutée
    while i1 < len(T1):
        T[j] = T1[i1]
        i1 += 1
        j += 1

    while i2 < len(T2):
        T[j] = T2[i2]
        i2 += 1
        j += 1

    return T
```

Question 2 (Réponse)

L'algorithme de fusion est conçu pour combiner deux listes déjà triées en une seule liste triée. Il repose sur le fait que les éléments dans chaque liste d'entrée sont déjà dans l'ordre.

La fonction compare les éléments de T1 et T2 en commençant par le début de chaque liste. Si les listes sont triées, elle peut efficacement déterminer quel élément doit être placé en premier dans la liste fusionnée.

En comparant et en sélectionnant le plus petit élément à chaque étape, l'algorithme garantit que la liste résultante T sera également triée, mais seulement si les listes d'entrée le sont. L'algorithme n'a besoin de parcourir chaque liste qu'une seule fois, ce qui lui donne une complexité de O(n+m), où n et m sont les longueurs de T1 et T2. Cette efficacité n'est possible que si les listes sont déjà triées.

Les boucles finales qui gèrent les éléments restants d'une liste lorsque l'autre est épuisée ne fonctionnent correctement que si les listes sont triées. Elles supposent que tous les éléments restants sont plus grands que ceux déjà ajoutés à T.

Partie B : Le tri fusion

```
def tri_fusion(T: list) -> list:
    """
    Renvoie un tableau T trié
    """

    n = len(T)
    # Cas de base
    if n < 2:
        return T[:]  # copie de T

    # Diviser : découpe de T
    T1 = T[0 : n // 2]
    T2 = T[n // 2 : n]

    # Résoudre : appels récursifs
    T1 = tri_fusion(T1)
    T2 = tri_fusion(T2)

    # Combiner : fusion des tableaux triés
    return fusion(T1, T2)
```

Exercice 2 - Cryptage selon le « Code de César » (5 points)

1

Résultat d'exécution :

D
A

2

```
def cryptage(self,texte):
        c = ""
        for l in texte:
            c = c + self.decale(l)
        return c
```

3

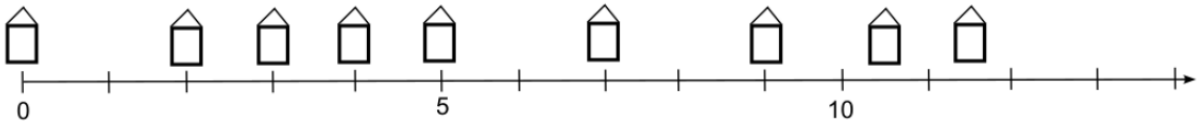
```
cle = input("saisir la clé de chiffrement : ")
cle = int(cle)
c = CodeCesar(cle)
txt = input("saisir le texte à chiffrer : ")
print("le message chiffré est : "+c.cryptage(txt))
```

4

La ligne `print(CodeCesar(10).transforme("PSX"))` va permettre d'afficher **FIN**

EXERCICE 3 (5 points)

1.
```
m1 = Maison(1)
m2 = Maison(3.5)
```
2.
```
a = Antenne(2.5, 1)
```
3.



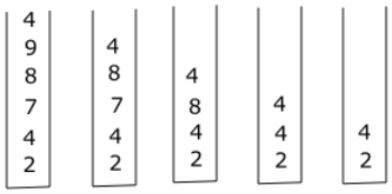
4.
```
def creation_rue(pos):
    pos.sort()
    maisons = []
    for p in pos:
        m = Maison(p)
        maisons.append(m)
    return maisons
```

5.
```
def couvre(self, maison):
    return abs(self.get_pos_antenne() - maison.get_pos_maison()) <= self.get_rayon()
```

EXERCICE 4 (5 points)

1.

a.



b.

La pile gagnante est la pile B

2.

```
def reduire_triplet_au_sommet(p):
    a = depiler(p)
    b = depiler(p)
    c = sommet(p)
    if a % 2 != c % 2 :
        empiler(p, b)
    empiler(p, a)
```

3.

a. La taille minimum d'une pile pour être réductible est 3.

b.

```
def parcourir_pile_en_reduisant(p):
    q = creer_pile_vide()
    while taille(p) >= 3:
        reduire_triplet_au_sommet(p)
        e = depiler(p)
        empiler(q, e)
    while not est_vide(q):
        e = depiler(q)
        empiler(p, e)
    return p
```

4.

```
def jouer(p):
    q = parcourir_pile_en_reduisant(p)
    if taille(q) == taille(p) :
        return p
    else:
        return jouer(q)
```