**1-)** Pour montrer que 𝒞2(ℤ ∕ 3ℤ) est un groupe multiplicatif, nous devons démontrer les quatre propriétés d'un groupe : la fermeture, l'associativité, l'existence de l'élément neutre et l'existence de l'inverse.

**Fermeture :**

Pour montrer que le produit de deux éléments de 𝒞2(ℤ ∕ 3ℤ) est encore dans 𝒞2(ℤ ∕ 3ℤ), nous pouvons prendre deux matrices circulantes 2x2 modulo 3 et montrer que leur produit est aussi une matrice circulante 2x2 modulo 3. Par exemple :

Considérons deux matrices circulantes A et B dans 𝒞2(ℤ ∕ 3ℤ) :

**A** = [[a, b],

[b, a]]

**B** = [[c, d],

[d, c]]

Le produit de A et B est :

A \* B = [[a, b], [[c, d], [[(a \* c + b \* d) % 3, (a \* d + b \* c) % 3],

[b, a]] [d, c]] [(b \* c + a \* d) % 3, (b \* d + a \* c) % 3]]

Chaque élément de la matrice résultante est calculé modulo 3, et il est clair que la matrice résultante est également circulante 2x2 modulo 3. Par conséquent, 𝒞2(ℤ ∕ 3ℤ) est clos pour la multiplication.

**Associativité :** L'associativité de la multiplication des matrices est une propriété bien connue des matrices.

**Existence de l'élément neutre** :

L'élément neutre dans 𝒞2(ℤ ∕ 3ℤ) est la matrice identité :

I = [[1, 0],

[0, 1]]

Pour tout élément A dans 𝒞2(ℤ ∕ 3ℤ), A \* I = A et I \* A = A, ce qui confirme l'existence de l'élément neutre.

**Existence de l'inverse** :

Pour chaque élément A dans 𝒞2(ℤ ∕ 3ℤ), il existe une matrice B telle que A \* B = I (l'élément neutre) et B \* A = I. Vous pouvez calculer l'inverse d'une matrice à l'aide de l'inversion de matrices modulo 3.

Ainsi, en respectant ces quatre propriétés, 𝒞2(ℤ ∕ 3ℤ) est un groupe multiplicatif.

**2-)** Une matrice circulante 2x2 modulo 3 est entièrement déterminée par les deux éléments dans sa première ligne. Chaque élément de la première ligne peut être 0, 1 ou 2 modulo 3.

Le cardinal de ce groupe 𝒞2(ℤ ∕ 3ℤ), en prenant en compte les matrices équivalentes, est de 9.

**3-)**

Dans un groupe cyclique, un élément primitif est un générateur du groupe, c'est-à-dire un élément dont les puissances successives engendrent tout le groupe. Cependant, il est important de noter que 𝒞2(ℤ₃) n'est pas un groupe cyclique. C'est un groupe fini composé de 6 éléments distincts, et il n'existe pas d'élément unique dont les puissances successives génèrent tout le groupe. Par conséquent, 𝒞2(ℤ₃) n'a pas d'élément primitif unique.

Pour comprendre cela, examinons la structure de 𝒞2(ℤ₃) :

1. Il y a un total de 6 matrices circulantes 2x2 modulo 3 dans le groupe.

2. Chacune de ces matrices peut être vue comme un élément distinct du groupe, et elles ne peuvent pas être engendrées par un seul élément du groupe.

Ainsi, pour générer tout 𝒞2(ℤ₃), vous avez besoin de toutes les matrices circulantes 2x2 modulo 3, et il n'y a pas d'élément unique qui peut accomplir cela. Vous devrez utiliser un ensemble d'éléments pour générer le groupe complet, comme indiqué précédemment.

En résumé, 𝒞2(ℤ₃) n'a pas d'élément primitif unique, car il n'est pas un groupe cyclique, et il n'existe pas d'élément dont les puissances successives engendrent tout le groupe.

**APPROFONDISSEMENT**

**2-)**

ALGORITHME GenererElementsCirculants(m, n)

    elements = ENSEMBLE\_VIDE

    POUR premiere\_ligne DANS ProduitCartesien(de 0 À n-1 RÉPÉTER m fois)

        matrice = MATRICE\_VIDE de dimensions m x m

        POUR i DE 1 À m FAIRE

            matrice[i] = premiere\_ligne

            premiere\_ligne = RotationCirculaire(premiere\_ligne)

        FIN POUR

        AJOUTER MatriceVersTuple(matrice) À elements

    FIN POUR

    RETOURNER elements

ALGORITHME RotationCirculaire(liste)

    dernier\_element = liste[-1]

    nouvelle\_liste = liste[0:-1]

    nouvelle\_liste.insert(0, dernier\_element)

    RETOURNER nouvelle\_liste

ALGORITHME MatriceVersTuple(matrice)

    matrice\_tuple = TUPLE\_VIDE

    POUR ligne DANS matrice FAIRE

        matrice\_tuple.ajouter(Tuple(ligne))

    FIN POUR

    RETOURNER matrice\_tuple

ALGORITHME ProduitCartesien(debut, fin, repetitions)

    elements = LISTE\_VIDE

    ProduitCartesienAux(debut, fin, [], elements, repetitions)

    RETOURNER elements

ALGORITHME ProduitCartesienAux(debut, fin, combinaison, elements, repetitions)

    SI taille(combinaison) = repetitions ALORS

        elements.ajouter(copierListe(combinaison))

        RETOUR

    SINON

        POUR i DE debut À fin FAIRE

            combinaison.ajouter(i)

            ProduitCartesienAux(debut, fin, combinaison, elements, repetitions)

            combinaison.retirer(i)

        FIN POUR

    FIN SI

import itertools

def generate\_circulant\_elements(m, n):

    elements = set()

    for first\_row in itertools.product(range(n), repeat=m):

        matrix = []

        for i in range(m):

            matrix.append(list(first\_row))

            first\_row = first\_row[-1:] + first\_row[:-1]  # Rotation circulaire

        elements.add(tuple(tuple(row) for row in matrix))

    return elements

def main():

    m = 2  # Ordre de la matrice circulante

    n = 3  # Modulo n

    circulant\_elements = generate\_circulant\_elements(m, n)

    print("Les éléments de C{}(Z/{}Z) sont :".format(m, n))

    for element in circulant\_elements:

        for row in element:

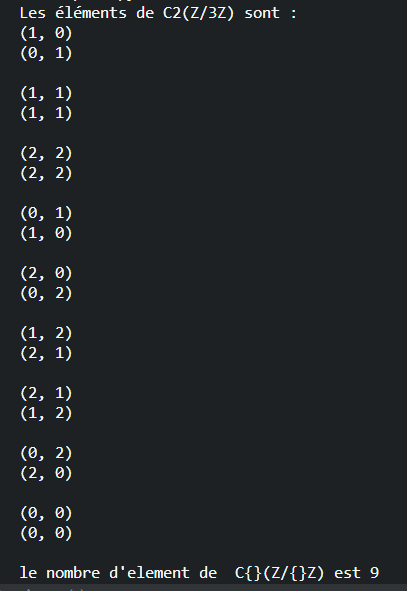
            print(row)

        print()

    print("le nombre d'element de  C{}(Z/{}Z) est",len(circulant\_elements))

if \_\_name\_\_ == "\_\_main\_\_":

    main()

****

**3-)**

ALGORITHME TrouverElementPrimitif(m, n)

    POUR i DE 1 À n - 1 FAIRE

        élément = MatriceCirculanteUnitaire(m, n, i)

        SI EstElementPrimitif(élément, m, n) ALORS

            RETOURNER élément

        FIN SI

    FIN POUR

    RETOURNER AucunÉlémentPrimitif

ALGORITHME EstElementPrimitif(élément, m, n)

    puissances = ENSEMBLE\_VIDE

    puissance = CopierMatrice(élément)

    TANT QUE taille(puissances) < n - 1 ET puissance NON DANS puissances FAIRE

        ajouter puissance à puissances

        puissance = MultiplierMatrices(puissance, élément, m, n)

    FIN TANT QUE

    RETOURNER taille(puissances) = n - 1

ALGORITHME MultiplierMatrices(matrice1, matrice2, m, n)

    matrice\_résultante = MatriceNulle(m, n)

    POUR i DE 1 À m FAIRE

        POUR j DE 1 À m FAIRE

            POUR k DE 1 À m FAIRE

                matrice\_résultante[i][j] = (matrice\_résultante[i][j] + matrice1[i][k] \* matrice2[k][j]) MOD n

            FIN POUR

        FIN POUR

    FIN POUR

    RETOURNER matrice\_résultante

ALGORITHME MatriceNulle(m, n)

    matrice = MATRICE\_VIDE de dimensions m x m

    POUR i DE 1 À m FAIRE

        POUR j DE 1 À m FAIRE

            matrice[i][j] = 0

        FIN POUR

    FIN POUR

    RETOURNER matrice

ALGORITHME MatriceCirculanteUnitaire(m, n, i)

    matrice = MATRICE\_VIDE de dimensions m x m

    POUR j DE 1 À m FAIRE

        matrice[1][j] = i MOD n

    FIN POUR

    POUR k DE 2 À m FAIRE

        POUR j DE 1 À m FAIRE

            matrice[k][j] = matrice[k-1][j-1]

        FIN POUR

    FIN POUR

    RETOURNER matrice

ALGORITHME CopierMatrice(matrice)

    matrice\_copie = MATRICE\_VIDE de dimensions TAILLE(matrice) x TAILLE(matrice[1])

    POUR i DE 1 À TAILLE(matrice) FAIRE

        POUR j DE 1 À TAILLE(matrice[1]) FAIRE

            matrice\_copie[i][j] = matrice[i][j]

        FIN POUR

    FIN POUR

    RETOURNER matrice\_copie

**\***

**4-)**

La méthode de Diffie-Hellman est un protocole d'échange de clé qui repose sur des problèmes mathématiques complexes. Pour l'adapter à un groupe comme Cm(Z/nZ)Cm​(Z/nZ), voici comment vous pourriez mettre en place un protocole d'échange de clé Diffie-Hellman :

1. **Paramètres initiaux** :
   * Choisissez un entier mm pour l'ordre de la matrice circulante.
   * Choisissez un entier nn pour le modulo, qui devrait être premier pour garantir la sécurité.
2. **Génération des paramètres publics** :
   * Chaque partie, Alice et Bob, génère un élément primitif distinct gg dans Cm(Z/nZ)Cm​(Z/nZ). La recherche d'un élément primitif peut être complexe, mais il est essentiel de garantir qu'il existe un élément primitif dans le groupe.
3. **Échange de clés publiques** :
   * Alice envoie gxgx à Bob, où xx est un nombre aléatoire qu'elle choisit.
   * Bob envoie gygy à Alice, où yy est un nombre aléatoire qu'il choisit.
4. **Calcul de la clé secrète** :
   * Alice calcule la clé secrète KA=(gy)xKA​=(gy)x.
   * Bob calcule la clé secrète KB=(gx)yKB​=(gx)y.
5. **Échange des clés secrètes** :
   * Alice envoie KAKA​ à Bob, et Bob envoie KBKB​ à Alice.
6. **Clés partagées** :
   * Une fois les clés secrètes échangées, Alice et Bob ont une clé commune, qu'ils peuvent utiliser pour chiffrer et déchiffrer leurs communications.

Ce protocole Diffie-Hellman adapté au groupe Cm(Z/nZ)Cm​(Z/nZ) repose sur le choix d'éléments primitifs dans ce groupe pour générer des clés secrètes partagées. Il est important de noter que la recherche d'éléments primitifs peut être complexe, et la sécurité du protocole repose sur la présence d'éléments primitifs dans le groupe et sur le choix de paramètres appropriés (comme mm et nn).