TP1

DUONG Minh Nghia - KRAAIJENBRINK Antony

1. Test d'une loi

1.1 Codage d'une loi sur un ensemble

Question 1:

Liste des éléments E4 et le tableau correspondant de loi t4 de (Z/4Z, +, 0)

```
C4= groups.permutation.Cyclic(4)
t4 = C4.cayley_table().table()
E4 = C4.cayley_table().column_keys()
```

```
E4
```

```
((), (1,2,3,4), (1,3)(2,4), (1,4,3,2))
```

```
t4
```

```
[[0, 1, 2, 3], [1, 2, 3, 0], [2, 3, 0, 1], [3, 0, 1, 2]]
```

Liste des éléments E2 et le tableau correspondant de loi t2 de (Z/2Z x Z/2Z, +, 0)

```
C2 = groups.permutation.Cyclic(2)
C2xC2 = cartesian_product([C2,C2])
t2 = C2xC2.cayley_table().table()
E2 = C2xC2.cayley_table().column_keys()
C2xC2.list()
```

```
[((), ()), ((), (1,2)), ((1,2), ()), ((1,2), (1,2))]
```

```
E2
```

```
(((), ()), ((), (1,2)), ((1,2), ()), ((1,2), (1,2)))
```

```
t2
```

```
[[0, 1, 2, 3], [1, 0, 3, 2], [2, 3, 0, 1], [3, 2, 1, 0]]
```

1.2 Elément neutre

Question 2:

Le procedure teste si une table t admet un élément neutre

```
def testeElementNeutre(E,t):
```

```
    for alpha in range(len(E)):
        compteur = 0
        for i in range(len(E)):
            if (t[i][alpha] == i and t[alpha][i] == i):
                compteur += 1
        if (compteur == len(E)):
            return E[alpha]
    return []
```

- Element neutre de (Z/4Z, +, 0)

```
elementNeutre_E4 = testeElementNeutre(E4,t4)
elementNeutre_E4
```

```
()
```

- Element neutre de (Z/2Z x Z/2Z, +, 0)

```
elementNeutre_E2 = testeElementNeutre(E2,t2)
elementNeutre_E2
```

```
((), ())
```

- Cherche directement élément neutre de (Z/4Z, +, 0) et (Z/2Z x Z/2Z, +, 0) à l'aide de SageMath:

```
C4.identity()
```

```
()
```

```
C2xC2.one()
```

```
((), ())
```

1.3 Element symétrique

Question 3:

Procedure contruire la table symétrique

```
def contruireSymetrie(E,t):
    symetrie = []
    indice_Neutre = E.index(testeElementNeutre(E,t))
    for i in range(len(E)):
        for j in range(len(E)):
            if t[i][j] == indice_Neutre and t[j][i] ==
indice_Neutre :
                symetrie.append(E[j])
    return symetrie
```

- La table symetique de (Z/4Z, +, 0)

```
symetrie_E4 = contruireSymetrie(E4,t4)
symetrie_E4
```

```
[(), (1,4,3,2), (1,3)(2,4), (1,2,3,4)]
```

- La table symetique de (Z/2Z x Z/2Z, +, 0)

```
symetrie_E2 = contruireSymetrie(E2,t2)
symetrie_E2
```

```
[((), ()), ((), (1,2)), ((1,2), ()), ((1,2), (1,2))]
```

- Cherche directement des tables symétriques de (Z/4Z, +, 0) et (Z/2Z x Z/2Z, +, 0) à l'aide de SageMath:

```
map(lambda n: n.inverse(),C4.list())
```

```
[(), (1,4,3,2), (1,3)(2,4), (1,2,3,4)]
```

```
map(lambda n: (n[0].inverse(), n[1].inverse()),C2xC2.list())
```

```
[((), ()), ((), (1,2)), ((1,2), ()), ((1,2), (1,2))]
```

1.4 Associativité

:ion 4:
Procedure de tester si la loi d'un groupe est associative

```
def estAssociative(E,t):
    associative = true
    for i in range(len(E)):
        for j in range(len(E)):
            for k in range(len(E)):
                if t[t[i][j]][k] != t[i][t[j][k]]:
                    associative = false
                    break
    return associative
```

- Teste si (Z/4Z, +, 0) est associative

```
estAssociative(E4,t4)
```

```
True
```

- Teste si (Z/2Z x Z/2Z, +, 0) est associative

```
estAssociative(E2,t2)
```

```
True
```

- Reverifie avec la fonction de SageMath

```
C4.is_abelian()
```

```
True
```

```
C2.is_abelian()
```

```
True
```

2.Test d'un morphisme

Question 5

1. Application de tester si une application est un morphisme

```
def estMorphisme(G, Gp, t, tp, f):
    morphisme = true
    for i in range(len(G)):
        for j in range(len(Gp)):
            if f[t[i][j]] != tp[f[i]][f[j]]:
                morphisme = false
                break
    return morphisme
```

2. Cherche tous le homomorphisme

Tous les applications bijective possibles de (Z/4Z, +, 0) et (Z/2Z x Z/2Z, +, 0) sont les permutations possibles de 4 éléments

```
tousPermutations = Permutations([0,1,2,3]).list()
```

```
tousPermutations
```

```
[[0, 1, 2, 3],
 [0, 1, 3, 2],
 [0, 2, 1, 3],
 [0, 2, 3, 1],
 [0, 3, 1, 2],
 [0, 3, 2, 1],
 [1, 0, 2, 3],
 [1, 0, 3, 2],
 [1, 2, 0, 3],
 [1, 2, 3, 0],
 [1, 3, 0, 2],
 [1, 3, 2, 0],
 [2, 0, 1, 3],
 [2, 0, 3, 1],
 [2, 1, 0, 3],
 [2, 1, 3, 0],
 [2, 3, 0, 1],
 [2, 3, 1, 0],
 [3, 0, 1, 2],
 [3, 0, 2, 1],
 [3, 1, 0, 2],
 [3, 1, 2, 0],
 [3, 2, 0, 1],
 [3, 2, 1, 0]]
```

- Pour cherche tous les homomorphisme on doit tester tous les applications bijectives et puis choisir les morphisme entre eux

```
def chercheMorphisme(G, Gp, t, tp, applications):
    listeMorphisme = []
    for i in range(len(applications)):
        if estMorphisme(G,Gp,t,tp,applications[i]):
            listeMorphisme.append(applications[i])
    return listeMorphisme
```

=> Les homomorphisme de (Z/4Z, +, 0) :

```
chercheMorphisme(E4, E4, t4, t4, tousPermutations)
```

```
[[0, 1, 2, 3], [0, 3, 2, 1]]
```

=> Les homomorphisme de (Z/2Z x Z/2Z, +, 0):

```
chercheMorphisme(E2, E2, t2, t2, tousPermutations)
```

```
[[0, 1, 2, 3],
 [0, 1, 3, 2],
 [0, 2, 1, 3],
 [0, 2, 3, 1],
 [0, 3, 1, 2],
 [0, 3, 2, 1]]
```

3. Pour montrer que (Z/4Z, +, 0) et (Z/2Z x Z/2Z, +, 0) ne sont pas isomorphes, il est nécessaire de montrer qu'il n'existe pas de isomorphisme entre eux

```
chercheMorphisme(E4,E2,t4,t2,tousPermutations)
```

```
[]
```

Il n'existe pas de morphisme entre (Z/4Z, +, 0) et (Z/2Z x Z/2Z, +, 0), donc ils sont pas isomorphes

4. Teste directement à l'aide de fonction SageMath

```
groups.permutation.KleinFour().is_isomorphic(C4)
```

```
False
```

3. Ou l’on identifie tous les groupes d’ordre 4

Question 6

Dans un groupe (E, *, e), pour tous a_i appartien à E, $a_i*a_j = a_i*a_k \iff a_j = a_k$. Donc dans la table de *, chaque élément apparait une seule fois par ligne et une seule fois par colonne.

Question 7

1. Les tables sont des tables d'opérations de (E, *, a_0) parce qu'elles admentent un élément neutre a_0, chaque élément apparait une seule fois par ligne et une seule fois par colonne et admet un élément symétrique unique.

Celles sont les seules tables d'opération car les constraints du groupe ne permentent qu'un groupe de 4 éléménts d'avoir au plus 4 tables d'opération possibles.

2. Tester les isomorphes de (Z/4Z, +, 0) et (Z/2Z x Z/2Z, +, 0)

```
table0 = [
[0,1,2,3],
[1,2,3,0],
[2,3,0,1],
[3,0,1,2]
]

table1 = [
[0,1,2,3],
[1,0,3,2],
[2,3,1,0],
[3,2,0,1]
]

table2 = [
[0,1,2,3],
[1,0,3,2],
[2,3,1,0],
[3,2,1,0]
]

table3 = [
[0,1,2,3],
[1,0,3,2],
[2,0,3,1],
[3,2,1,0]
]
```

```
chercheMorphisme(E4,E4,t4,table0,tousPermutations)
```

```
[[0, 1, 2, 3], [0, 3, 2, 1]]
```

```
chercheMorphisme(E4,E4,t4,table1,tousPermutations)
```

```
[[0, 2, 1, 3], [0, 3, 1, 2]]
```

```
chercheMorphisme(E4,E4,t4,table2,tousPermutations)
```

```
[]
```

```
chercheMorphisme(E4,E4,t4,table3,tousPermutations)
```

```
[]
```

Donc les tables 0 et 1 sont isomorphes de (Z/4Z, +, 0)

```
chercheMorphisme(E2,E4,t2,table0,tousPermutations)
```

    []

```
chercheMorphisme(E2,E4,t2,table1,tousPermutations)
```

    []

```
chercheMorphisme(E2,E4,t2,table2,tousPermutations)
```

    []

```
chercheMorphisme(E2,E4,t2,table3,tousPermutations)
```

    []

Donc les 4 tables ne sont pas isomorphes de (Z/2Z x Z/2Z, +, 0)

Question 8

1. Test du Théorème de Lagrange

Soit G un groupe, Voici la fonction *testLagrange* teste la divisibilité de l'ordre des sous-groupe de G et G

```
def testLagrange(G):
    ordre_G = G.order()
    for ordre in map(order,G.subgroups()):
        if ordre_G%ordre != 0:
            return false
    return true
```

```
testLagrange(SymmetricGroup(5))
```

    True

b. Groupe symétrie de n-gon

```
testLagrange(DihedralGroup(6))
```

    True

c. Groupe de permutation cyclique

```
testLagrange(CyclicPermutationGroup(7))
```

    True

d. KleinFour Groupe

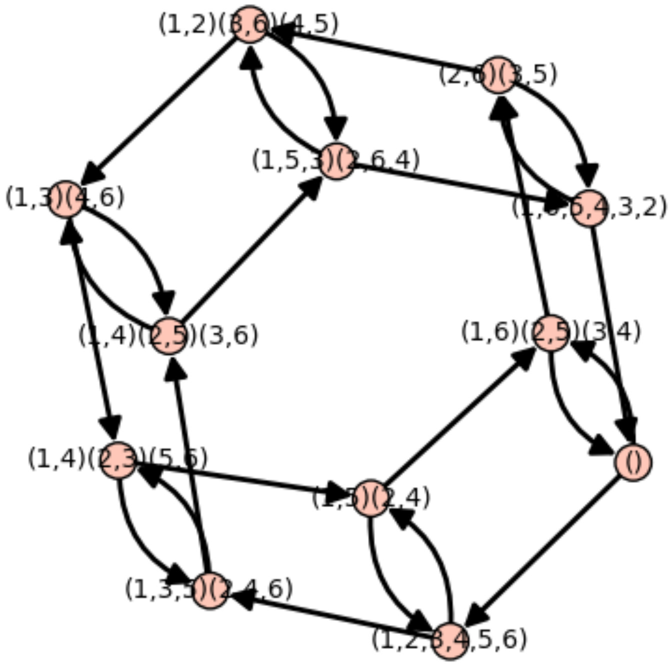
```
testLagrange(KleinFourGroup())
```

    True

2.Graphe de Cayley

On appele le graphe de Cayley pour un groupe les sous-groupe cyclique du groupe.

```
show(DihedralGroup(6).cayley_graph())
```



Question 9

Pour trouver la taille minimale de carré latin qui n'est pas un group, on cherche dans les combinaison possibles des permutations les carré latin qui n'est pas un groupe.

- La fonction qui teste si une table est un carré latin:

```
def estCarreLatin(t):
    for i in range(len(t)):
        res1 = []
        res2 = []
        for j in range(len(t)):
            if t[i][j] not in res1:
                res1.append(t[i][j])
            if t[j][i] not in res2:
                res2.append(t[j][i])
        if len(res1) != len(t) or len(res2) != len(t):
             return false
    return true
```

- La fonction qui cherche dans les combinaison possibles des permutations les carré latin qui n'est pas un groupe.

```
def pasGroupe(CP,E):
    res = []
    for P in CP:
        if estCarreLatin(P):
            if testeElementNeutre(E,P) == [] or
not(estAssociative(E,P)) or len(contruireSymetrie(E,P)) != len(E):
```

```
                res.append(P)
    return res
```

- La fonction qui retourne le carré latin de taille minimum qui n'est pas un groupe:

```
def tailleMinimale():
    i = 1
    E = [0,1]
    P = Permutations(E).list()
    CP = Permutations(P,i+1).list()
    while carrelatin(CP,E) == []:
        i += 1
        E.append(i)
        P = Permutations(E).list()
        CP = Permutations(P,len(E)).list()
    return carrelatin(CP,E)
```

```
tailleMinimale()
```

```
    [[0, 1, 2], [2, 0, 1], [1, 2, 0]]
```

Alors la taille minimale:

```
len(tailleMinimale()[0])
```

```
    3
```