

RAPPORT DU TP TEST PYTHON

Nom : Eliezer Mahugnon DJIHINTO

Partage de code et tests boîte noire

Implémentation du test pour chacune des fonctions selon ces cas à tester :

- ✓ lancer ppcm de (0,0) doit donner 0.
- ✓ lancer ppcm de (0,X) doit donner 0.
- ✓ lancer ppcm de (X,0) doit donner 0.
- ✓ lancer ppcm de (X,X) doit donner X.
- ✓ lancer ppcm de (168, 60) doit donner 840.

Création d'un fichier `test_arithmetics.py` pour les tests. //Utilisation de VSCode comme environnement. (*Code en annexe à ce rapport*)

Résultats des tests :

Après exécution de la commande : `coverage run -m unittest .\test_arithmetics.py`

```
-----  
Ran 15 tests in 0.017s  
  
FAILED (failures=1, errors=5)
```

Tous les codes des assertions s'exécutent correctement, soient 15 assertions.

Avec 1 échec et 5 erreurs :

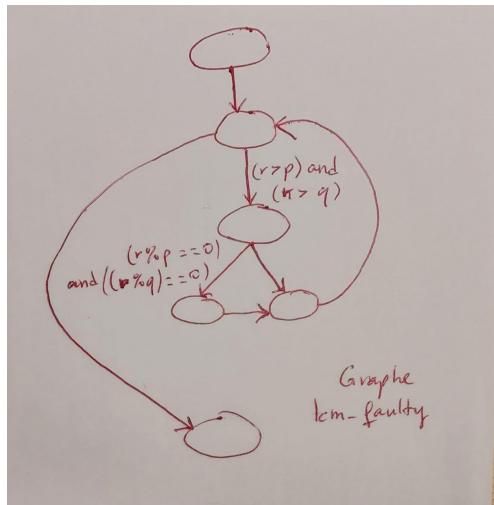
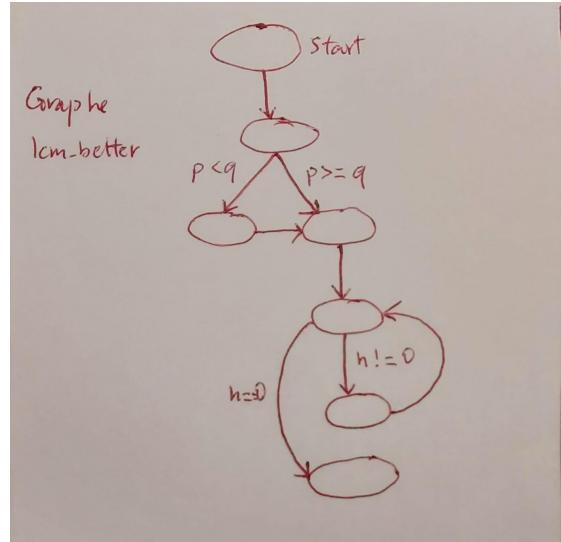
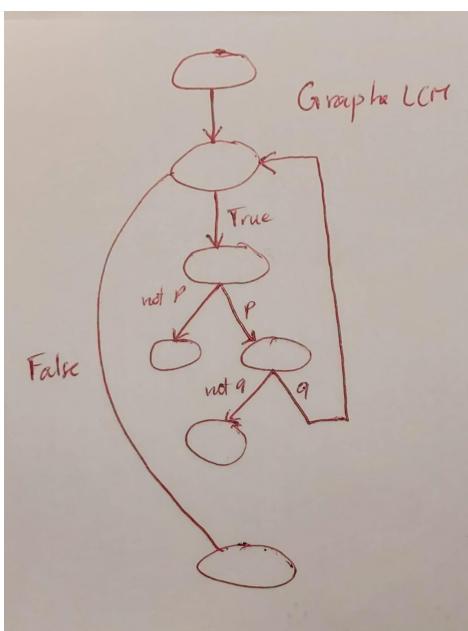
- l'échec est signalé au niveau de la fonction `lcm_faulty` lancée avec (X,X) qui ne retourne pas la bonne valeur comme dit dans les spécifications de la question parce que la définition de la fonction `lcm_faulty` est erronée dans le fichier `arithmetics.py`.
- les erreurs sont notamment : la levée de la `ZeroDivisionError` lors des tests où 0 passe en division au dénominateur, dans chacune des fonctions.

En terme de couverture, les tests ne sont pas exhaustifs.

Coverage report: 87%					
File	class	statements	missing	excluded	coverage
<small>coverage.py v7.10.7, created at 2025-10-13 11:28 +0200</small>					
arithmetics.py	(no class)	42	9	0	79%
test_arithmetics.py	TestLCM	15	0	0	100%
test_arithmetics.py	(no class)	20	1	0	95%
Total		77	10	0	87%

Couverture de code

Graphes de contrôle :



Après exécution de coverage, on constate que l'intégralité du code n'est pas couverte par les tests unitaires. Certaines branches du code, notamment dans la fonction lcm, ne sont jamais atteintes pendant les exécutions de test.

```

7     p, q = abs(p), abs(q)
8     m = p * q
9     while True:
10        p %= q
11        if not p:
12            return m // q
13        q %= p
14        if not q:
15            return m // p
16
17 def lcm_better(p, q):
18     p, q = abs(p), abs(q)
19     if (p < q):
20         r = p
21         p = q
22         q = r
23     m = p * q
24     h = p % q
25     while h != 0:

```

La dernière ligne conditionnelle n'est pas couverte par défaut, car dans la majorité des cas testés ($\text{lcm}(0,0)$, $(7,7)$, $(168,60)$ etc.), la boucle quitte par la première condition `if not p:`. Autrement dit, la deuxième condition n'est jamais vraie.

- `lcm_better(p, q)` :

Si les tests ne couvrent que des cas où $p \geq q$, la branche du `if (p < q):` n'est pas couverte non plus.

- `lcm_faulty(p, q)` :

Le code est défectueux (boucle descendante `while (r > p) and (r > q):`), donc certaines parties du `if` ne sont atteintes que pour des valeurs particulières de p et q .

La couverture y est donc partielle.

Annexe 1 : Code des tests

```
import unittest  
from arithmetics import lcm, lcm_better, lcm_faulty
```

```
class TestLCM(unittest.TestCase):
```

```
    # ----- Tests pour lcm -----  
  
    def test_lcm_branch_p_zero(self):  
        self.assertEqual(lcm(40, 50), 200)
```

```
    def test_lcm_branch_q_zero(self):  
        self.assertEqual(lcm(50, 40), 200)
```

```
    def test_lcm_equal(self):  
        self.assertEqual(lcm(7, 7), 7)
```

```
    def test_lcm_raises_exception(self):  
        with self.assertRaises(ValueError):  
            lcm(0, 7)  
        with self.assertRaises(ValueError):  
            lcm(7, 0)  
        with self.assertRaises(ValueError):  
            lcm(0, 0)
```

```
    # ----- Tests pour lcm_better -----  
  
    def test_lcm_better_swap(self):  
        self.assertEqual(lcm_better(4, 6), 12)
```

```
    def test_lcm_better_no_swap(self):  
        self.assertEqual(lcm_better(6, 4), 12)
```

```
    def test_lcm_better_equal(self):  
        self.assertEqual(lcm_better(10, 10), 10)
```

```
    def test_lcm_better_raises_exception(self):  
        with self.assertRaises(ValueError):  
            lcm_better(0, 7)  
        with self.assertRaises(ValueError):  
            lcm_better(7, 0)  
        with self.assertRaises(ValueError):  
            lcm_better(0, 0)
```

```
    # ----- Tests pour lcm_faulty -----
```

```
def test_lcm_faulty_common_case(self):
    lcm_faulty(40, 50)
    lcm_faulty(50, 40)
```

```
def test_lcm_faulty_no_loop(self):
    lcm_faulty(1, 1)
```

```
def test_lcm_faulty_raises_exception(self):
    with self.assertRaises(ValueError):
        lcm_faulty(0, 7)
    with self.assertRaises(ValueError):
        lcm_faulty(7, 0)
    with self.assertRaises(ValueError):
        lcm_faulty(0, 0)
```

```
if __name__ == '__main__':
    unittest.main()
```

Annexe 2 : Code original modifié

```
# My own Python script, written by John. All rights reserved to John.

def print_lcm(l, p, q):
    print(f'Least common multiple of {p} and {q} is {l}')
```

```
def lcm(p, q):
    if p == 0 or q == 0:
        raise ValueError("LCM is undefined when one of the arguments is zero")
    p, q = abs(p), abs(q)
    m = p * q
    while True:
        p %= q
        if not p:
            return m // q
        q %= p
        if not q:
            return m // p
```

```
def lcm_better(p, q):
    if p == 0 or q == 0:
        raise ValueError("LCM is undefined when one of the arguments is zero")
    p, q = abs(p), abs(q)
    if (p < q):
        r = p
        p = q
        q = r
    m = p * q
    h = p % q
    while h != 0:
        p = q
        q = h
        h = p % q
    h = m // q
    return h
```

```
def lcm_faulty(p, q):
    if p == 0 or q == 0:
        raise ValueError("LCM is undefined when one of the arguments is zero")
    r, m = 0, 0
    r = p * q
    while (r > p) and (r > q):
        if (r % p == 0) and (r % q == 0):
            m = r
        r = r - 1
    return m
```