```python
# Standard Library
from fractions import Fraction as Frac
from typing import Iterable

# Relative
from utils import ZERO, ONE, fail_msg, print_heading, FracVec, FracMatrix

def is_valid_p(p: Frac) -> bool:
    return p >= ZERO and p <= ONE

def is_square(m: FracMatrix) -> bool:
    return len(m) == len(m[0])

def test_is_square(where: str, m: FracMatrix) -> None:
    assert is_square(m), fail_msg(where, 'not an NxN matrix', 'square matrix', m)

def test_probs_eq_1(where: str, ps: Iterable[Frac]):
    assert sum(ps) == ONE or sum(ps) == 1, \
        fail_msg(
            where,
            "sum of probabilities not equal to 1",
            1.0,
            f"p = sum({', '.join(map(str, ps))}) = {float(sum(ps))}")

def test_tranition_probs(trans_table: FracMatrix, exe_no: int) -> None:
    for state in range(len(trans_table)):
        total_p: Union[Frac, int] = sum(trans_table[state])
        test_probs_eq_1('test_transition_probs', total_p)

    def between_inclusive(x, m, n) -> bool: return x >= m and x <= n

    for row in range(len(trans_table)):
        for col in range(len(trans_table[row])):
            assert between_inclusive(trans_table[row][col], ZERO, ONE), \
                fail_msg(
                    f"exercise {exe_no}",
                    f'p({row + 1} -> {col + 1}) no in range [0, 1]',
                    'p in range [0, 1]',
                    '%2.2f' % float(trans_table[row][col]))

def test_exercise_1() -> None:
    SSP, trans_table = exercise_1()
    test_is_square('test_exercise_1', trans_table)
    test_tranition_probs(trans_table, 1)


def test_exercise_2() -> None:
    SSP, trans_table = exercise_1()
    test_is_square('test_exercise_2', trans_table)
    test_tranition_probs(trans_table, 2)

def test_exercise_3() -> None:
    std, p1, p3, p9 = exercise_3()

    def check(n: int, p: Frac) -> None:
        assert is_valid_p(p), fail_msg(
            'exercise 3',
            f'invalid SSP for state {n}',
            'valid SSP in range [0, 1]',
            '%2.2f' % float(p))

    check(1, p1)
    check(3, p3)
    check(9, p9)

    assert (p1 + p3 + p9) <= ONE, \
        fail_msg('exercise 3',
                 'invalid SSP for states 1, 3, 9'
```

```
                'sum(p(1), p(3), p(9)) < 1.0',
                '%2.2f' % float(p1 + p3 + p9))


def test_exercise_4() -> None:
    std, p1, p3, p9 = exercise_4()

    def check(n: int, p: Frac) -> None:
        assert is_valid_p(p), fail_msg("exercise 4",
                                       f"invalid SSP for state {n}",
                                       f"p({n}) in range [0, 1]",
                                       p)

    assert (p1 + p3 + p9) <= ONE, \
        fail_msg("exercise 4",
                 "invalid SSP for states 1, 3, 9",
                 "sum(p(1), p(3), p(9)) < 1.0",
                 p1 + p3 + p9)

    check(1, p1)
    check(3, p3)
    check(9, p9)
```