```python
"""
Name: Hongda Li
Class cse 417
This file is for hw1, problem 5.
* Codes require python 3.6 or above.
* Codes requires solutions of problem 4.
! Codes are slow cause it's written in python.

Here are the definition for some of the keywords listed in problem 5:
m.rank() -> The choice of m after the perfect matching algorithms.
M.goodness -> sum_{i=0}^{n-1}m.rank()/n

Output produced:
------------------------------------------------------------------------------------------------------------------------
Running on random input, we have the following values for goodness:
[(5.2368, 24.2752), (6.552000000000001, 39.0364), (7.1232, 71.23079999999999),
(7.118900000000001, 143.7869),
(8.116, 249.63195000000002), (9.064699999999998, 451.93919999999997)]
N = [125, 250, 500, 1000, 2000, 4000], n = 10
------------------------------------------------------------------------------------------------------------------------
[(5.684, 23.903200000000005), (5.954800000000001, 42.9284), (6.8506, 75.0774),
(7.1289, 144.4004), (8.005550000000001, 254.77534999999997), (8.60445, 475.20764999999994),
(9.626925, 850.7660250000001)]
N = [125, 250, 500, 1000, 2000, 4000, 8000], n = 10
------------------------------------------------------------------------------------------------------------------------
[(4.92048, 25.2448), (6.008319999999998, 41.97607999999998), (6.659639999999999,
76.63016000000002),
(7.445580000000001, 136.68726), (8.089469999999999, 253.58859000000004), (8.919784999999997,
456.4239599999999),
 (9.735907500000001, 837.95485)]
N = [125, 250, 500, 1000, 2000, 4000, 8000], n = 50
"""
from typing import List, Tuple
from random import random
from problem4 import convert, produce_stable_match


def rand_permutation(arr: List)-> List:
    """
    :param arr:
    A array with elements.
    :return:
    A new randomly permutated array from arr.
    """
    newarr = arr.copy()
    for I in range(len(newarr)):
        J = int(random()*I)
        newarr[I], newarr[J] = newarr[J], newarr[I]
    return newarr


def get_goodness(arr: List[int], M: List[List], W: List[List])->Tuple:
    """
    Function will return the measure of goodness for both the, M and W using the returned results gotten
    from problem 4.
    :param arr:
        The results produced from problem 4.
    :param M:
        The preference matrix for M.
    :param W:
    :return.
        A tuple where the first element is the goodness for M and the second is the goodness for W.
    """
    M_psum = 0
    W_psum = 0
    M_tbl = convert(M)
    W_tbl = convert(W)
    l = len(arr)
    assert arr is not None, "Why are you passing None to this function? "
    for E, I in zip(arr, range(len(arr))):
        M_psum += M_tbl[I][1][E] + 1
        W_psum += W_tbl[E][1][I] + 1
    return (M_psum/l, W_psum/l)


def goodness_for(N:int):
    """
    Function will generate a randomly permutated lists for the preferece list for M, W, then it
    will measure the goodness for W, and M, with an n starting at 1000, increments at 100 and ends at 1e4
    :param N:
        The size of the problem.
    :return:
    """
    lst = list(range(N))
    M = [rand_permutation(lst) for I in range(N)]
    W = [rand_permutation(lst) for I in range(N)]
    return get_goodness(produce_stable_match(M, W, verbo=False), M, W)


if __name__ == "__main__":
    print("Let's test something first before running everything else. ")
    print("All m has the same preference list for w while w has random preference list: ")
    n = 100
    R = list(range(n))
    M = [R for I in range(n)]
    W = [rand_permutation(R) for I in range(n)]
    result = produce_stable_match(M, W, verbo=False)
    print(result)
    goodness = get_goodness(result, M, W)
    assert goodness[0] == 5050/100, "Ok, there is something wrong please check. "
    print("Ok, for the special cause proved in problem 1, the codes seem to work. ")
    print("Running on random input, we have the following values for goodness: ")
    n = 50
    stats = [[goodness_for(J) for I in range(n)] for J in [125, 250, 500, 1000, 2000, 4000, 8000]]


    def stats_helper(row):
        m_sum, w_sum = 0, 0
        for m_Goodness, w_Goodness in row:
            m_sum += m_Goodness
            w_sum += w_Goodness
        return m_sum/len(row), w_sum/len(row)
    stats = list(map(stats_helper, stats))
    print(stats)
```