**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ Кафедра прикладних інформаційних систем**

ЗВІТ

Про виконання лабораторної роботи №5

з дисципліни: «Технології розроблення програмних систем»

Студент групи ПП-42

Мельник Валентин

Київ – 2022

**Тема**: Модульні тести

**Код програми:**

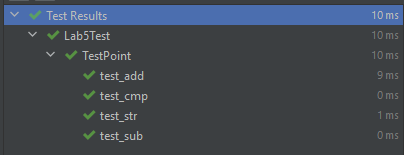
import sys  
import dataclasses  
from typing import List, Iterable, Tuple  
  
@dataclasses.dataclass  
class Point:  
 x: int = 0  
 y: int = 0  
  
 def clone(self):  
 return dataclasses.replace(self)  
  
 def \_\_add\_\_(self, other: 'Point'):  
 return Point(self.x + other.x, self.y + other.y)  
  
 def \_\_sub\_\_(self, other: 'Point'):  
 return Point(self.x - other.x, self.y - other.y)  
  
 def \_\_lt\_\_(self, other: 'Point'):  
 return self.y < other.y or (self.y == other.y and self.x < other.x)  
  
 def \_\_le\_\_(self, other: 'Point'):  
 return self.y < other.y or (self.y == other.y and self.x <= other.x)  
  
 def \_\_str\_\_(self):  
 return f'({self.x},{self.y})'  
  
  
class Field:  
 def \_\_init\_\_(self, width, height, default\_value=None):  
 self.\_width = width  
 self.\_height = height  
 self.\_data = [  
 [default\_value] \* width  
 for \_ in range(height)  
 ]  
  
 def \_\_getitem\_\_(self, key):  
 x, y = key  
 return self.\_data[y][x]  
  
 def \_\_setitem\_\_(self, key, value):  
 *"""****:key*** *x, y coordinates"""* x, y = key  
 self.\_data[y][x] = value  
  
 @property  
 def width(self):  
 return self.\_width  
  
 @property  
 def height(self):  
 return self.\_height  
  
 def reset(self, value):  
 for y in range(self.height):  
 for x in range(self.width):  
 self.\_data[y][x] = value  
  
 def iterate\_4\_neighbourhood(self, x: int, y: int, radius: int) -> Iterable[Tuple[int, int]]:  
 for dy in range(radius):  
 for dx in range(1, radius - dy + 1):  
 if y >= dy and x >= dx:  
 yield x - dx, y - dy  
 if y+dy < self.height and x+dx < self.width:  
 yield x + dx, y + dy  
 if y+dx < self.height and x >= dy:  
 yield x - dy, y + dx  
 if y >= dx and x+dy < self.width:  
 yield x + dy, y - dx  
  
@dataclasses.dataclass  
class OptimalLocation:  
 shop\_count: int = 0  
 position: Point = dataclasses.field(default\_factory=Point)  
  
  
class Simulator:  
 def \_\_init\_\_(self, city\_width: int, city\_length: int, shops: List[Point], queries: List[int]):  
 self.\_shops = Field(city\_width, city\_length, 0)  
 for s in shops:  
 self.\_shops[s.x, s.y] += 1  
 self.\_queries = queries  
  
 def run(self)-> List[OptimalLocation]:  
 result = [  
 self.\_find\_optimal\_location(radius)  
 for radius in self.\_queries  
 ]  
 return result  
  
 def \_find\_optimal\_location(self, radius):  
 location = OptimalLocation()  
 for y in range(self.\_shops.height):  
 for x in range(self.\_shops.width):  
 if self.\_shops[x, y] > 0:  
 continue  
 p = Point(x, y)  
 local\_count = self.\_get\_shop\_count(radius, x, y)  
 if location.shop\_count > local\_count:  
 continue  
 if local\_count > location.shop\_count or location.position > p:  
 location.shop\_count = local\_count  
 location.position = p  
 return location  
  
 def \_get\_shop\_count(self, radius, x, y):  
 shop\_count = 0  
 for neighbour in self.\_shops.iterate\_4\_neighbourhood(x, y, radius):  
 shop\_count += self.\_shops[neighbour]  
 return shop\_count  
  
  
MIN\_GRID\_SIZE = 1  
MAX\_GRID\_SIZE = 1000  
GRID\_SIZE\_BOUNDS = f'[{MIN\_GRID\_SIZE} .. {MAX\_GRID\_SIZE}]'  
  
MIN\_SHOP\_COUNT = 0  
MAX\_SHOP\_COUNT = 5105  
SHOP\_COUNT\_BOUNDS = f'[{MIN\_SHOP\_COUNT} .. {MAX\_SHOP\_COUNT}]'  
  
MIN\_QUERY\_COUNT = 1  
MAX\_QUERY\_COUNT = 20  
QUERY\_COUNT\_BOUNDS = f'[{MIN\_QUERY\_COUNT} .. {MAX\_QUERY\_COUNT}]'  
  
MIN\_WALK\_DISTANCE = 0  
MAX\_WALK\_DISTANCE = 106  
WALK\_DISTANCE\_BOUNDS = f'[{MIN\_WALK\_DISTANCE} .. {MAX\_WALK\_DISTANCE}]'  
  
POSITION\_OUTPUT\_OFFSET = Point(1, 1)  
  
EXIT\_FAIL = 1  
  
  
def main(argv):  
 f = sys.stdin  
 case\_number = 1  
 while True:  
 l = f.readline()  
 if l is None:  
 print('Unexpected EOF, cases list should be followed by line',  
 'of 4 zeros separated with whitespace')  
 return EXIT\_FAIL  
  
 l = l.split()  
 try:  
 if len(l) < 4:  
 raise ValueError()  
 dx, dy, shop\_count, query\_count = map(int, l)  
 except ValueError:  
 print('First line of each case should contain 4 integers:',  
 'city grid dimensions, coffee chop and query counts')  
 return EXIT\_FAIL  
  
 if dx == dy == shop\_count == query\_count == 0:  
 break  
  
 print(f'\nCase {case\_number}:')  
  
 if not(MIN\_GRID\_SIZE <= dx <= MAX\_GRID\_SIZE  
 and MIN\_GRID\_SIZE <= dy <= MAX\_GRID\_SIZE):  
 print(f'City grid dimensions {dx}x{dy} are not within allowed',  
 'range:', GRID\_SIZE\_BOUNDS)  
 return EXIT\_FAIL  
  
 if not(MIN\_SHOP\_COUNT <= shop\_count <= MAX\_SHOP\_COUNT):  
 print(f'Shop count {shop\_count} is not within allowed range:',  
 SHOP\_COUNT\_BOUNDS)  
 return EXIT\_FAIL  
  
 if not(MIN\_QUERY\_COUNT <= query\_count <= MAX\_QUERY\_COUNT):  
 print(f'Query count {query\_count} is not within allowed range:',  
 QUERY\_COUNT\_BOUNDS)  
 return EXIT\_FAIL  
  
 shops: List[Point] = []  
 for i in range(shop\_count):  
 l = f.readline()  
 if l is None:  
 print(f'Unexpected EOF, shop list should have {shop\_count}',  
 'lines with shop coordinates')  
 return EXIT\_FAIL  
 try:  
 shop\_x, shop\_y = map(int, l.split())  
 except ValueError:  
 print('Each line in shop list should contain 2 integers - shop',  
 'coordinates')  
 return EXIT\_FAIL  
 if not(MIN\_GRID\_SIZE <= shop\_x <= dx  
 and MIN\_GRID\_SIZE <= shop\_y <= dy):  
 print(f'Shop coordinates ({shop\_x}, {shop\_y}) are outside of',  
 'the city bounds or in the wrong order')  
 return EXIT\_FAIL  
 shops.append(Point(shop\_x - 1, shop\_y - 1))  
  
 queries: List[int] = []  
 for i in range(query\_count):  
 l = f.readline()  
 if l is None:  
 print(f'Unexpected EOF, query list should have {query\_count}',  
 'lines with maximum walk distances')  
 return EXIT\_FAIL  
 try:  
 query = int(l)  
 except ValueError:  
 print('Each line in query list should contain integer -',  
 'maximum walk distance for the query')  
 return EXIT\_FAIL  
 if not(MIN\_WALK\_DISTANCE <= query <= MAX\_WALK\_DISTANCE):  
 print(f'Walk distance {query} is not within allowed range:',  
 WALK\_DISTANCE\_BOUNDS)  
 return EXIT\_FAIL  
 queries.append(query)  
  
 s = Simulator(dx, dy, shops, queries)  
 best\_locations = s.run()  
 for l in best\_locations:  
 print(l.shop\_count, l.position+POSITION\_OUTPUT\_OFFSET)  
 case\_number += 1  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 sys.exit(main(sys.argv))

**Код тестів:**

import unittest  
  
from Lab5TRPS import Point  
  
  
class TestPoint(unittest.TestCase):  
  
 def setUp(self):  
 self.point\_1 = Point(4, 5)  
 self.point\_2 = Point(1, -2)  
 self.point\_3 = Point(4, 6)  
 self.point\_4 = Point(5, 5)  
 self.point\_5 = Point(4, -5)  
 self.point\_6 = Point(4, 5)  
  
 def test\_add(self):  
 test\_point = self.point\_1 + self.point\_2  
 self.assertEqual(test\_point.x, 5)  
 self.assertEqual(test\_point.y, 3)  
  
 def test\_sub(self):  
 test\_point = self.point\_1 - self.point\_2  
 self.assertEqual(test\_point.x, 3)  
 self.assertEqual(test\_point.y, 7)  
  
 def test\_str(self):  
 self.assertEqual(str(self.point\_1), "(4,5)")  
  
 def test\_cmp(self):  
 try:  
 self.assertTrue(self.point\_1 <= self.point\_1)  
 except NotImplementedError:  
 None  
 try:  
 self.assertFalse(self.point\_1 < self.point\_1)  
 self.assertFalse(self.point\_1 < self.point\_2)  
 self.assertTrue(self.point\_1 < self.point\_3)  
 self.assertTrue(self.point\_1 < self.point\_4)  
 except NotImplementedError:  
 None  
  
 try:  
 self.assertTrue(self.point\_1 >= self.point\_1)  
 except NotImplementedError:  
 None  
  
 try:  
 self.assertFalse(self.point\_1 > self.point\_1)  
 self.assertFalse(self.point\_2 > self.point\_1)  
 self.assertTrue(self.point\_3 > self.point\_1)  
 self.assertTrue(self.point\_4 > self.point\_1)  
 except NotImplementedError:  
 None  
  
 try:  
 self.assertFalse(self.point\_1 == self.point\_2)  
 self.assertFalse(self.point\_1 == self.point\_5)  
 self.assertTrue(self.point\_1 == self.point\_6)  
 except NotImplementedError:  
 None  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

**Результат роботи програми та тестів:**





**Висновок**: в межах цієї лабораторної роботи я вирішив задачу Coffe Central та розробив модульні тести до окремого компоненту