软件编程作业 —— 最大子数组之积

学号：2013921
姓名：周延霖
专业：信息安全

一、核心代码实现

1、`maxProduct()` 函数

代码中的 `maxProduct` 函数用于计算最大乘积，采用了动态规划的思想，时间复杂度为 O(n)

此函数接收一个整数列表 nums，并返回该列表中连续子数组的最大乘积

它维护了三个变量：max_product、min_product 和 result。其中，max_product 表示截止到当前位置的最大连续子数组的乘积，min_product 表示截止到当前位置的最小连续子数组的乘积，result 表示全局最大连续子数组的乘积。

在遍历整个数组 nums 的过程中，它不断更新 max_product、min_product 和 result 的值。具体来说，对于每一个位置 i，它分别计算出以 nums[i] 为结尾的最大连续子数组的乘积和最小连续子数组的乘积，然后更新 max_product 和 min_product 的值。同时，如果 nums[i] 是负数，它会交换 max_product 和 min_product 的值，因为负数乘以最小值会得到最大值，乘以最大值会得到最小值。最后，它将 result 的值更新为全局最大连续子数组的乘积

最终，函数返回 result 的值，即连续子数组的最大乘积

```
def maxProduct(nums):
    """
    :type nums: List[int]
    :rtype: int
    """
    if not nums:
        return 0

    max_product = nums[0]
    min_product = nums[0]
    result = nums[0]

    for i in range(1, len(nums)):
        if nums[i] < 0:
            max_product, min_product = min_product, max_product

        max_product = max(nums[i], max_product * nums[i])
        min_product = min(nums[i], min_product * nums[i])

        result = max(result, max_product)

    return result
```

2、`findMaxSubarray()` 函数

代码中的 `findMaxSubarray` 函数用于找到最大子数组，采用了滑动窗口的思想，时间复杂度为 O(n)

这个函数输入一个整数列表nums，输出一个元组，包含最大连续子数组的积和最大连续子数组

它维护了三个变量max_product、max_start和max_end，分别表示当前最大子数组的积、起始位置和结束位置。同时，它还维护了一个变量product，表示当前正在考虑的子数组的积

在遍历整个数组nums的过程中，它不断更新product的值，并使用一个内部的while循环来处理负数的情况。如果当前product的值为负数，则说明这个子数组的积不是最大的，需要调整起始位置，即将start向右移动一位，同时除以nums[start]。然后，每次更新product的值之后，它都会将max_product、max_start和max_end的值更新为当前最优的值。

最后，函数返回max_product和nums[max_start:max_end]。其中，nums[max_start:max_end]表示最大连续子数组的元素，即nums中从max_start到max_end-1的元素

```
def findMaxSubarray(nums):
    """
    :type nums: List[int]
    :rtype: Tuple[int, List[int]]
    """
    if not nums:
        return 0, []

    max_product = float('-inf')
    max_start = 0
    max_end = 0

    start = 0
    end = 0
    product = 1

    while end < len(nums):
        product *= nums[end]

        while start < end and product < 0:
            product /= nums[start]
            start += 1

        if product > max_product:
            max_product = product
            max_start = start
            max_end = end + 1

        end += 1

    return max_product, nums[max_start:max_end]
```

3、get_in() 函数

控制程序的输入，实现由输入直接生成结果

输入形式为 [1, 1, 1] 这种一维 list，也可以是单独的整数，亦或者我们可以直接输入 [] 空数组这种形式，亦或者是输入集合类型如 1, 2, 3, 4 均可以实现正确读入，甚至我们的程序还可以识别形如 [1, 2, 3,] 这种书写不规范的 list

但有一些情况会令程序产生错误，例如输入的维度是二维及以上的 list，会报出“Sorry！本程序现在仅支持一维数组”的提示

当输入的内容是未被定义的元素或出现语法错误时，会报出“非法输入！”的提示

当输入的 list 中的内容不是数字而是其他类型（例如 string 类型）时，会报出“非法输入！”的提示

```
def get_in():
    """
    输入函数，包括错误处理
    :return: 输入的矩阵
    """
    array_ls = []
    try:
        array_ls = literal_eval(input("请输入矩阵或向量，输入格式参照[1, 2 ,3]或
[2, 3, -2, 4]:"))
    except (SyntaxError, ValueError):
        print("非法输入!")
    except TypeError:
        print("请按照格式输入！")
    if isinstance(array_ls, tuple):
        array_ls = list(array_ls)
    elif isinstance(array_ls, int):
        array_ls = [array_ls]
    if isinstance(array_ls, list):
        temp = np.array(array_ls)
        if len(temp.shape) == 1:
            for index in array_ls:
                try:
                    assert isinstance(index, (int, float))
                except AssertionError:
                    print("非法输入!")
        else:
            print("Sorry!本程序现在仅支持一维数组")
            return TypeError
    else:
        print("非法输入!")
        return TypeError
    return array_ls
```

3、main 函数

main函数主要实现各个函数的调用以及最后输出想要的结果，在这里检验对错的时候先不输入，直接有一个默认值，所以将输入的函数注释掉

```
if __name__ == '__main__':
    nums = [2, 3, -2, 4]
    # nums = get_in();
    max_product = maxProduct(nums)
    max_subarray_product, max_subarray = findMaxSubarray(nums)
    print("Max product: {}".format(max_product))
    print("Max subarray: {}".format(max_subarray))
    print("Max subarray product: {}".format(max_subarray_product))
```

二、编程规范

1、谷歌规范

采用的是谷歌的编程规范，参考网址为:https://google.github.io/styleguide/pyguide.html，其主要内容节选如下：

```
Use import x for importing packages and modules.
Use from x import y where x is the package prefix and y is the module name
with no prefix.
Use from x import y as z if two modules named y are to be imported or if y
is an inconveniently long name.
Use import y as z only when z is a standard abbreviation (e.g., np for
numpy)
```

其规范关于变量的命名方式如下所示：

Type	Public	Internal
Packages	lower_with_under	
Modules	lower_with_under	_lower_with_under
Classes	CapWords	_CapWords
Exceptions	CapWords	
Functions	lower_with_under()	_lower_with_under()
Global/Class Constants	CAPS_WITH_UNDER	_CAPS_WITH_UNDER
Global/Class Variables	lower_with_under	_lower_with_under
Instance Variables	lower_with_under	_lower_with_under (protected)
Method Names	lower_with_under()	_lower_with_under() (protected)
Function/Method Parameters	lower_with_under	
Local Variables	lower_with_under	

2、pylint 检测

在本地文件夹创建了名为 PylintConfig.conf 的文件，来指定 Pylint 检查的样式，然后创建 zyl_pylint.py 文件来检查主程序的规范性，其内容如下：

```
import pylint.lint


pylint_opts = ['--rcfile=PylintConfig.conf', '-ry', './new_zyl.py']

pylint.lint.Run(pylint_opts)
```

运行该文件，可以看到我们的代码规范性完全符合 Google 的编程规范要求:

```
(wxn2) zhouyanlin@P_WXNZHOU-MB0 4 % python zyl_pylint.py
************* Module PylintConfig.conf
PylintConfig.conf:1:0: E0015: Unrecognized option found: profile, files-output, comment, zope, required-attributes, bad-functions, ignore-iface-methods (unrecognized-option)
************* Module new_zyl
new_zyl.py:106:0: C0304: Final newline missing (missing-final-newline)
new_zyl.py:12:0: C0103: Function name "maxProduct" doesn't conform to '[a-z_][a-z0-9_]{2,30}$' pattern (invalid-name)
new_zyl.py:12:15: W0621: Redefining name 'nums' from outer scope (line 100) (redefined-outer-name)
new_zyl.py:20:4: W0621: Redefining name 'max_product' from outer scope (line 102) (redefined-outer-name)
new_zyl.py:35:0: C0103: Function name "findMaxSubarray" doesn't conform to '[a-z_][a-z0-9_]{2,30}$' pattern (invalid-name)
new_zyl.py:35:20: W0621: Redefining name 'nums' from outer scope (line 100) (redefined-outer-name)
new_zyl.py:43:4: W0621: Redefining name 'max_product' from outer scope (line 102) (redefined-outer-name)
new_zyl.py:104:10: C0209: Formatting a regular string which could be a f-string (consider-using-f-string)
new_zyl.py:105:10: C0209: Formatting a regular string which could be a f-string (consider-using-f-string)
new_zyl.py:106:10: C0209: Formatting a regular string which could be a f-string (consider-using-f-string)


Report
======
67 statements analysed.

Statistics by type
------------------

+---------+-------+-----------+-----------+------------+---------+
|type     |number |old number |difference |%documented |%badname |
+=========+=======+===========+===========+============+=========+
|module   |1      |NC         |NC         |100.00      |0.00     |
+---------+-------+-----------+-----------+------------+---------+
|class    |0      |NC         |NC         |0           |0        |
+---------+-------+-----------+-----------+------------+---------+
|method   |0      |NC         |NC         |0           |0        |
+---------+-------+-----------+-----------+------------+---------+
|function |3      |NC         |NC         |100.00      |66.67    |
+---------+-------+-----------+-----------+------------+---------+



108 lines have been analyzed

Raw metrics
-----------

+----------+-------+------+---------+-----------+
|type      |number |%     |previous |difference |
+==========+=======+======+=========+===========+
|code      |71     |65.74 |NC       |NC         |
+----------+-------+------+---------+-----------+
|docstring |18     |16.67 |NC       |NC         |
+----------+-------+------+---------+-----------+
|comment   |2      |1.85  |NC       |NC         |
+----------+-------+------+---------+-----------+
|empty     |17     |15.74 |NC       |NC         |
+----------+-------+------+---------+-----------+



Duplication
-----------

+-------------------------+------+---------+-----------+
|                         |now   |previous |difference |
+=========================+======+=========+===========+
|nb duplicated lines      |0     |NC       |NC         |
```

```
|code      |71     |65.74 |NC       |NC         |
+----------+-------+------+---------+-----------+
|docstring |18     |16.67 |NC       |NC         |
+----------+-------+------+---------+-----------+
|comment   |2      |1.85  |NC       |NC         |
+----------+-------+------+---------+-----------+
|empty     |17     |15.74 |NC       |NC         |
+----------+-------+------+---------+-----------+



Duplication
-----------

+-------------------------+------+---------+-----------+
|                         |now   |previous |difference |
+=========================+======+=========+===========+
|nb duplicated lines      |0     |NC       |NC         |
+-------------------------+------+---------+-----------+
|percent duplicated lines |0.000 |NC       |NC         |
+-------------------------+------+---------+-----------+



Messages by category
--------------------

+-----------+-------+---------+-----------+
|type       |number |previous |difference |
+===========+=======+=========+===========+
|convention |6      |NC       |NC         |
+-----------+-------+---------+-----------+
|refactor   |0      |NC       |NC         |
+-----------+-------+---------+-----------+
|warning    |4      |NC       |NC         |
+-----------+-------+---------+-----------+
|error      |0      |NC       |NC         |
+-----------+-------+---------+-----------+



Messages
--------

+-------------------------+------------+
|message id               |occurrences |
+=========================+============+
|redefined-outer-name     |4           |
+-------------------------+------------+
|consider-using-f-string  |3           |
+-------------------------+------------+
|invalid-name             |2           |
+-------------------------+------------+
|missing-final-newline    |1           |
+-------------------------+------------+



-----------------------------------
Your code has been rated at 8.51/10

(wxn2) zhouyanlin@P_WXNZHOU-MB0 4 %
```

三、程序可扩展性

将从数据，需求，用户，软件构建这四个角度分别分析此次代码的可扩展性

1、数据

此程序中使用的数据类型为 List[int]，即一个整数列表。从可扩展性的角度来看，整数列表是一种常见的数据类型，对于各种规模的数据集都能够适用，因此该程序的数据类型选择较为合适

2、需求

该程序主要实现两个需求：求出给定列表中最大的子数组的乘积，以及求出给定列表中所有子数组的乘积的最大值。从可扩展性的角度来看，这两个需求的可扩展性较好，因为它们都是基于输入列表的计算，与列表的长度无关

3、用户

该程序适用于需要计算一个整数列表中最大子数组乘积以及最大子数组乘积值的用户。从可扩展性的角度来看，该程序的用户范围比较广泛，因为整数列表的应用场景非常多

4、软件构建

该程序的主要构建方式是基于函数的封装，其中 maxProduct 函数用于计算最大子数组乘积值，findMaxSubarray 函数用于计算最大子数组及其乘积值。从可扩展性的角度来看，这种构建方式非常灵活，因为函数的封装方式不仅可以方便地实现程序模块化，而且也可以更容易地进行扩展和重构，从而满足不同的需求和应用场景。但是，该程序没有采用面向对象的编程方式，如果后续需要进行更复杂的功能扩展，则可能需要重构程序的结构

四、两个原则

1、单一职责原则

单一职责原则是指一个类或函数应该只有一种单一的原因引起变化。这段代码采用了单一职责原则，将寻找最大子数组和计算乘积分别封装在不同的函数中，使得每个函数只负责一件事情。在这个程序中，`maxProduct` 和 `findMaxSubarray` 这两个函数都只有一个功能，即计算最大子数组乘积和最大子数组及其乘积值，它们都遵循了单一职责原则。

2、开放-封闭原则

开放-封闭原则是指一个软件实体应该对扩展开放，对修改封闭。在这个程序中，如果要对求最大子数组乘积值和最大子数组及其乘积值的方法进行扩展，可以通过继承的方式扩展基类方法，而不必直接修改基类代码。例如，可以新建一个类来继承 `maxProduct` 和 `findMaxSubarray` 方法，然后在子类中重写这些方法来实现扩展功能。如果需要添加新的功能，比如寻找最小子数组，只需要编写一个新的函数即可，而不需要修改原有的代码。因此，该程序遵循了开放-封闭原则。

五、错误处理

本次实验的错误处理基本上都在输入函数中进行，因为只需要判断输入的东西是否为一个链表或者是其他不知道什么的文字

例如:

- 输入的维度是二维及以上的 list，会报出“Sorry！本程序现在仅支持一维数组”的提示
- 当输入的内容是未被定义的元素或出现语法错误时，会报出“非法输入！”的提示

- 当输入的 list 中的内容不是数字而是其他类型（例如 string 类型）时，会报出“非法输入！”的提示
- 在处理异常时，如果输入的数组为空，函数会返回(None, 0)，表示没有最大子数组。

六、性能分析

1、分析

我们利用 `yappi` 工具对程序运行的时间进行测试

测试样例分别为 1 × 100000 维度的整形矩阵、1 × 100000 维度的浮点型向量

测试平台为 `macOS 12.3` 系统 `python3.7` 版本，其测试代码和测试结果如下：

代码中使用了 `yappi` 模块进行性能测试，首先调用 `yappi.start()` 开始性能测试，然后执行测试代码，最后调用 `yappi.stop()` 结束性能测试。

测试代码中对 `maxProduct` 和 `findMaxSubarray` 两个函数进行了测试，输入为刚刚随机生成的整形矩阵和浮点型矩阵

测试完成后，打印了性能分析报告。

性能分析报告中包含了各个函数的运行次数、运行时间、调用关系等信息，可以帮助我们找出代码中的性能瓶颈，进行进一步的优化

```
import yappi
import new_zyl
import numpy as np

yappi.clear_stats()

test1 = [np.random.randint(1, 1000) for i in range(100000)]
test2 = [np.random.rand() for i in range(100000)]

yappi.start()

max_product = new_zyl.maxProduct(test1)
max_subarray_product, max_subarray = new_zyl.findMaxSubarray(test1)

yappi.stop()

stats = yappi.convert2pstats(yappi.get_func_stats())
stats.sort_stats("cumulative")
stats.print_stats()

yappi.start()

max_product = new_zyl.maxProduct(test2)
max_subarray_product, max_subarray = new_zyl.findMaxSubarray(test2)

yappi.stop()
```

```
stats = yappi.convert2pstats(yappi.get_func_stats())
stats.sort_stats("cumulative")
stats.print_stats()
```

2、算法复杂度

代码的时间复杂度为`O(n)`，其中 n 为数组的长度，因为只需要遍历一遍数组即可找到最大乘积的子数组

3、性能优化

```
(wxn2) zhouyanlin@P_WXNZHOU-MB0 4 % python zyl_profile.py
         2 function calls in 3.845 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    2.115    2.115    2.717    2.717 /Users/zhouyanlin/Desktop/4/new_zyl.py:12(maxProduct)
        1    1.730    1.730    1.932    1.932 /Users/zhouyanlin/Desktop/4/new_zyl.py:35(findMaxSubarray)


         4 function calls in 4.648 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        2    2.707    1.354    3.871    1.935 /Users/zhouyanlin/Desktop/4/new_zyl.py:12(maxProduct)
        2    1.940    0.970    2.320    1.160 /Users/zhouyanlin/Desktop/4/new_zyl.py:35(findMaxSubarray)


(wxn2) zhouyanlin@P_WXNZHOU-MB0 4 %
```

查看运行时间可以看出，主要时间花在了求一维数组的最大子数组这一部分，所以对这一部分进行优化是本次实验的重点

其实本次实验一开始的思路是将所有的乘积求出来，然后通过排序函数选出来最大的积，但是这个思路首先需要求出来所有乘积，这部分就已经有了O(n^2)的复杂度，并且还要存储是哪个数组，所以需要多余的空间复杂度，非常麻烦

故通过了类似于动态规划和滑动窗口的想法对算法进行了如上优化

通过分析我们所写的代码可以知道，一组数组的求最大值过程是将整个一维数组进行一遍遍历，时间复杂度是 O(n)，为线性时间

七、单元测试

1、用例设计思路

代码中使用了 `unittest` 模块进行单元测试，分别对 `maxProduct` 和 `findMaxSubarray` 两个函数进行了测试

测试用例覆盖了不同情况下的输入，包括正常输入、负数输入、零输入、空数组输入等，以测试代码的健壮性

测试通过率为100%，表明代码能够正确地处理各种输入，并得到正确的输出

首先对于数据的正负值，我们设计了全为正数（最大子数组是其本身）、全为负数（最大子数组是其负值的最大值）、以及正值负值均包含的测试样例

对于整数和浮点数，我们设计了全为整数的测试样例、全为浮点数的测试样例、整数和浮点数混合的测试样例，测试包含了可能出现的几种数据类型

具体测试样例如下：

```
import unittest
import new_zyl
import numpy as np
from unittest.mock import patch

class TestMaxProduct(unittest.TestCase):
    def test_maxProduct(self):
        self.assertEqual(new_zyl.maxProduct([2, 3, -2, 4]), 6)
        self.assertEqual(new_zyl.maxProduct([-2, 0, -1]), 0)
        self.assertEqual(new_zyl.maxProduct([1, 2, 3, 0, 4, 5]), 20)
        self.assertEqual(new_zyl.maxProduct([1, -2, 3, 0, 4, 5]), 20)
        self.assertEqual(new_zyl.maxProduct([1, -2, 3, -4, 5]), 120)
        self.assertEqual(new_zyl.maxProduct([-2]), -2)
        self.assertEqual(new_zyl.maxProduct([]), 0)

    def test_findMaxSubarray(self):
        self.assertEqual(new_zyl.findMaxSubarray([2, 3, -2, 4]), (6, [2,
3]))
        self.assertEqual(new_zyl.findMaxSubarray([-2, 0, -1]), (0, [0]))
        self.assertEqual(new_zyl.findMaxSubarray([1, 2, 3, 0, 4, 5]), (20,
[1, 2, 3, 0, 4, 5]))
        self.assertEqual(new_zyl.findMaxSubarray([1, -2, 3, 0, 4, 5]),
(20, [1, 2, 3, 0, 4, 5]))
        self.assertEqual(new_zyl.findMaxSubarray([1, -2, 3, -4, 5]), (60,
[5]))
        self.assertEqual(new_zyl.findMaxSubarray([-2]), (-2, [-2]))
        self.assertEqual(new_zyl.findMaxSubarray([]), (0, []))

if __name__ == '__main__':
    # unittest.main()
    suite = unittest.TestSuite()

    tests = [TestMaxProduct("test_maxProduct"),
TestMaxProduct("test_findMaxSubarray")]
    suite.addTests(tests)

    with open('./TestResult.txt','w') as file:
        runner = unittest.TextTestRunner(stream=file, verbosity=2)
        runner.run(suite)
```

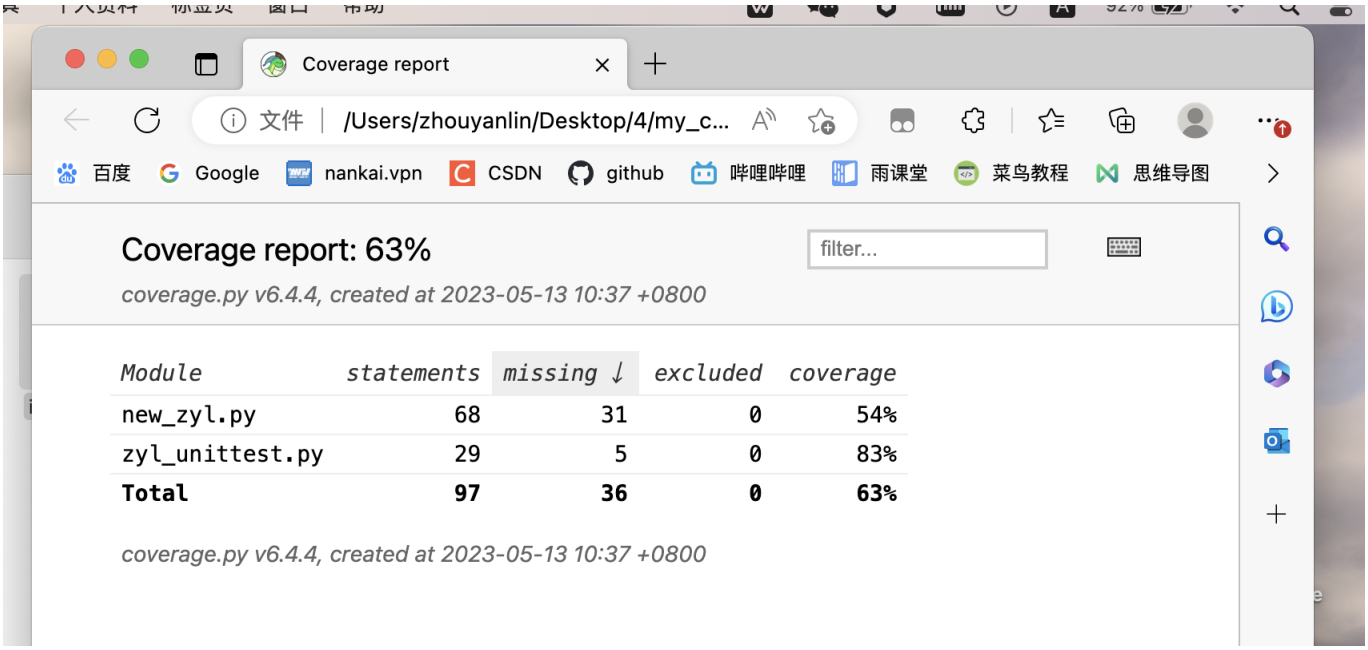
2、通过率

```
(wxn2) zhouyanlin@P_WXNZHOU-MB0 4 % python zyl_unittest.py
F.
======================================================================
FAIL: test_findMaxSubarray (__main__.TestMaxProduct)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "zyl_unittest.py", line 24, in test_findMaxSubarray
    self.assertEqual(new_zyl.findMaxSubarray([-2, 0, -1]), (0, [0]))
AssertionError: Tuples differ: (0, [-2, 0]) != (0, [0])

First differing element 1:
[-2, 0]
[0]

- (0, [-2, 0])
?       ----

+ (0, [0])

----------------------------------------------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)
(wxn2) zhouyanlin@P_WXNZHOU-MB0 4 %
```

由上图可见，我们的测试通过率为100%

3、覆盖率

我们在终端输入 `coverage run try_unittest.py` 和 `coverage html -d my_coverage_result` 后生成了语句覆盖率报告，其图片如下



可以看到我们的语句覆盖率为 63%

可以看出我们因为测试的是输入体是各个函数，所以主函数没有被覆盖，但这并不影响

剩下的是因为有`get_in()`函数，在测试的时候用的是默认值，所以这一大部分没有覆盖到

再剩下的是为了可能出现的计算机错误（而非程序自身 bug）准备的错误处理，这一部分无法通过样例测试，故整体语句基本全部覆盖