编程作业实验报告

- 姓名：陈睿颖
- 学号：2013544
- 专业：计算机科学与技术

1. 实验内容

题目:乘积最大子数组的编程实现、代码分析、单元测试

输入一个整数数组nums，需要找出数组中乘积最大的非空连续子数组，并输出该子数组所对应的乘积。(子数组是指数组的连续子序列)。

示例1:输入:nums=[2,3,-2,4]，输出:6，解释子数组[2,3]有最大乘积6

然后在此编程实现的基础上对其进行以下任务:用pylint进行代码分析、用profile进行性能分析、用unittest进行单元测试。

2. 代码设计

思路如下：

由于要找乘积最大的子数组，我们可以维护两个变量 `max_product` 和 `min_product`，分别表示以当前元素结尾的最大和最小的子数组乘积。

对于当前元素，如果是正数，则乘上前一个元素的最大子数组乘积 `max_product`，否则乘上前一个元素的最小子数组乘积 `min_product`。然后再更新 `max_product` 和 `min_product`。

由此可以进行代码的实现：

```python
from typing import List

def maxProduct(nums: List[int]) -> int:
    """
    给定一个整数数组，找出数组中乘积最大的非空连续子数组，并输出该子数组所对应的乘积。

    Args:
        nums: 整数数组

    Returns:
        最大子数组乘积

    Raises:
        ValueError: 数组为空或长度为0，或者数组中所有元素都为0
    """

    # 数组为空或长度为0，直接返回0
    if not nums:
        raise ValueError("数组不能为空")

    # 数组中所有元素都为0，最大子数组乘积也为0
    if all(num == 0 for num in nums):
        return 0

    max_product = min_product = res = nums[0]

    for i in range(1, len(nums)):
        if nums[i] >= 0:
            max_product = max(nums[i], max_product * nums[i])
            min_product = min(nums[i], min_product * nums[i])
        else:
            tmp = max_product
            max_product = max(nums[i], min_product * nums[i])
            min_product = min(nums[i], tmp * nums[i])

        res = max(res, max_product)

```

```
    return res

if __name__ == '__main__':
    print(maxProduct([2, 3, -2, 4]))

```

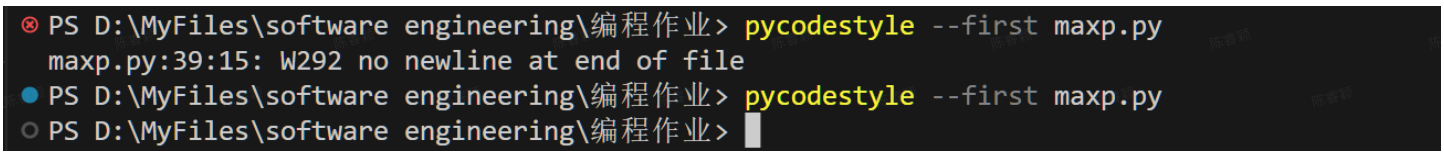
3. 思考题

3.1 是否遵守编程规范，参考的哪个规范，如何检查是否遵守编程规范的?

遵循了PEP 8编码规范，其中包括：

- 函数名使用小写字母和下划线的组合，例如 `max_product`。
- 使用双引号表示字符串，例如 `"数组不能为空"`。
- 使用四个空格缩进。
- 使用空格将运算符和变量隔开，例如 `max_product = min_product = res = nums[0]`。
- 在参数列表和函数返回值之间使用文档字符串进行说明。
- 在代码中适当加入注释，对代码进行解释和说明。

如图所示，使用pycodestyle工具进行检查，第一次运行时报错文件结尾没有空行，经修改后不再报错，说明符合PEP 8编程规范。



用pylint工具进行代码分析

结果如下：

```
************* Module maxp
maxp.py:4:0: C0103: Function name "maxProduct" doesn't conform to snake_case nam

-------------------------------------------------------------------
Your code has been rated at 9.44/10 (previous run: 8.89/10, +0.56)

```

代码被Pylint分析后，得到了9.44分（满分10分），其中一个警告是函数名 `maxProduct` 不符合Python中的蛇形命名规范（snake_case），因为函数名中每个单词应该使用下划线分隔。建议将函数名改为 `max_product` ，这样符合编码规范。更改后的代码如下：

```python
from typing import List

def cal_max_product(nums: List[int]) -> int:
    """
    给定一个整数数组，找出数组中乘积最大的非空连续子数组，并输出该子数组所对应的乘积。

    Args:
        nums: 整数数组

    Returns:
        最大子数组乘积

    Raises:
        ValueError: 数组为空或长度为0，或者数组中所有元素都为0
    """

    # 数组为空或长度为0，直接返回0
    if not nums:
        raise ValueError("数组不能为空")

    # 数组中所有元素都为0，最大子数组乘积也为0
    if all(num == 0 for num in nums):
        return 0

    max_product = min_product = res = nums[0]

    for i in range(1, len(nums)):
        if nums[i] >= 0:
            max_product = max(nums[i], max_product * nums[i])
            min_product = min(nums[i], min_product * nums[i])
        else:
            tmp = max_product
            max_product = max(nums[i], min_product * nums[i])
            min_product = min(nums[i], tmp * nums[i])

        res = max(res, max_product)

    return res

if __name__ == '__main__':
    print(cal_max_product([2, 3, -2, 4]))

```

经pylint分析后得到满分。

3.2 是否考虑算法的可扩展性 (已扩展了什么功能、未来还可以如何扩展，注意不要求按照实验课ppt上举例的扩展方法逐一实现)

作出下面的扩展：给定一个整数数组，找出数组中乘积最大的非空连续子数组，并输出该子数组所对应的乘积和其起始位置、结束位置。即在基础功能上实现了数组的索引输出。代码设计如下：

```python
def cal_max_product_with_index(nums: List[int]) -> Tuple[int, int, int]:
    """
    给定一个整数数组，找出数组中乘积最大的非空连续子数组，并输出该子数组所对应的乘积和其起始
    :param nums: 整数数组
    :return: 数组中乘积最大的非空连续子数组的乘积、其起始位置、结束位置
    """
    if not nums:
        return 0, 0, 0

    max_product = min_product = res = nums[0]
    start = end = res_start = 0

    for i in range(1, len(nums)):
        if nums[i] >= 0:
            if nums[i] == 0:
                max_product = min_product = 1
                start = end = i
            else:
                max_product = max(nums[i], max_product * nums[i])
                min_product = min(nums[i], min_product * nums[i])
                end = i
        else:
            tmp = max_product
            max_product = max(nums[i], min_product * nums[i])
            min_product = min(nums[i], tmp * nums[i])
            end = i

        if max_product > res:
            res = max_product
            res_start = start

        if max_product == 0:
            start = end = i + 1
            max_product = min_product = 1
```

```
35
36      return res, res_start, end
37
```

在基础部分的代码中加入两个变量 `start` 和 `end`，分别表示最大子数组的起始位置和结束位置。在更新 `max_product` 时，同时更新 `start` 和 `end`。即可实现扩展功能。

考虑到该函数的输出不唯一，可能有多个子数组乘积最大，所以可能有多种结果，未来可以实现将所有的结果进行返回。这里暂时只返回了一种。

3.3 是否遵守了两个原则，代码的哪个部分是后续扩展功能也可以重复使用、不会修改的基础功能

两个原则

- 单一职责原则 (Single Responsibility Principle,SRP) 指出：一个模块（类）应该只有一个导致它变化的原因，一个模块应该完全对某个功能负责。
- 另一个重要的软件设计原则是开放- 封闭原则(Open-Close Principle，OCP)：软件实体应该是可以扩展的，同时是不可修改的。
    - 允许修改(Open for modification)当应用的需求发生改变时，我们可以对模块进行扩展，从而改变模块的功能
    - 不允许修改(Closed for modification)。对模块行为进行扩展时，不必改变模块的本身。

单一职责原则要求一个模块（类）应该只有一个导致它变化的原因，一个模块应该完全对某个功能负责。这段代码中，有两个函数，`cal_max_product` 和 `cal_max_product_with_index`，每个函数只负责一个明确定义的任务，即计算一个整数数组中的最大子数组乘积和其起始位置、结束位置，这两个函数也不相互依赖，因此符合单一职责原则。

开放-封闭原则要求软件实体应该是可以扩展的，同时是不可修改的。该代码中，两个函数的功能都是可以被扩展的，如果需要计算最大子数组的其他属性（如长度），可以通过扩展函数 `cal_max_product_with_index` 来实现，而不需要修改 `cal_max_product` 函数。因此，该代码符合开放-封闭原则。

3.4 考虑了哪些错误与异常处理

在函数入口处对输入的数组进行判断，如果数组为空或长度为0，则直接返回0；另外，还需要对数组中的元素进行判断，如果数组中所有元素都为0，则最大子数组乘积也为0：

```
    # 数组为空或长度为0，直接返回0
    if not nums:
        raise ValueError("数组不能为空")

    # 数组中所有元素都为0，最大子数组乘积也为0
    if all(num == 0 for num in nums):
        return 0
```

3.5 算法复杂度是多少?如何对代码性能进行分析?分析的结果如何?你是如何进行优化的?

算法复杂度

算法的时间复杂度为O(n)，其中n是数组nums的长度。算法对于数组进行了一次遍历，因此时间复杂度为O(n)。空间复杂度为O(1)，只需要常数级别的额外空间用于记录变量。

使用profile进行性能分析

执行下面的命令：

```
python3 -m cProfile -o profileresult maxp.py
python3 -c "import pstats; p=pstats.Stats('profileresult');
p.sort_stats('time').print_stats()" >profile.txt
```

即可得到性能分析的结果：

```
Mon May 15 16:39:08 2023    profileresult

         138 function calls in 0.001 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.print}
       16    0.000    0.000    0.000    0.000 C:\Program Files\WindowsApps\Pytho
        1    0.000    0.000    0.001    0.001 maxp.py:1(<module>)
       16    0.000    0.000    0.000    0.000 C:\Program Files\WindowsApps\Pytho
        4    0.000    0.000    0.000    0.000 C:\Program Files\WindowsApps\Pytho
        2    0.000    0.000    0.000    0.000 C:\Program Files\WindowsApps\Pytho
```

```
        1    0.000    0.000    0.000    0.000 maxp.py:4(cal_max_product)
        1    0.000    0.000    0.000    0.000 C:\Program Files\WindowsApps\Pytho
        2    0.000    0.000    0.000    0.000 C:\Program Files\WindowsApps\Pytho
        2    0.000    0.000    0.000    0.000 C:\Program Files\WindowsApps\Pytho
        1    0.000    0.000    0.000    0.000 C:\Program Files\WindowsApps\Pytho
        3    0.000    0.000    0.000    0.000 C:\Program Files\WindowsApps\Pytho
       28    0.000    0.000    0.000    0.000 {built-in method builtins.isinstan
        4    0.000    0.000    0.000    0.000 C:\Program Files\WindowsApps\Pytho
       16    0.000    0.000    0.000    0.000 {method 'startswith' of 'str' obje
        2    0.000    0.000    0.000    0.000 C:\Program Files\WindowsApps\Pytho
        1    0.000    0.000    0.001    0.001 {built-in method builtins.exec}
        2    0.000    0.000    0.000    0.000 C:\Program Files\WindowsApps\Pytho
        1    0.000    0.000    0.000    0.000 C:\Program Files\WindowsApps\Pytho
        8    0.000    0.000    0.000    0.000 {method 'endswith' of 'str' object
        6    0.000    0.000    0.000    0.000 C:\Program Files\WindowsApps\Pytho
        4    0.000    0.000    0.000    0.000 C:\Program Files\WindowsApps\Pytho
        6    0.000    0.000    0.000    0.000 {built-in method builtins.max}
        2    0.000    0.000    0.000    0.000 maxp.py:23(<genexpr>)
        3    0.000    0.000    0.000    0.000 {built-in method builtins.min}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.all}
        3    0.000    0.000    0.000    0.000 {built-in method builtins.len}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Prof
```

上面的结果包括记录函数执行时间，按照执行时间排序，其中包括每个函数被调用的次数和它们各自的总运行时间。

总运行时间为0.001秒，即1毫秒。在这1毫秒的时间内，代码执行了138次函数调用。下面是函数调用的排名：

- 第一名是print函数，被调用了1次，总运行时间为0秒。因此，`print`函数对程序整体运行时间没有影响。
- 第二名是`typing.py`文件中的`__setattr__`函数，被调用了16次，总运行时间为0秒。这是Python自带的一个库，用于类型注解。因此，它也不会对程序运行时间产生影响。
- 第三名是代码中的`cal_max_product`函数，被调用了1次，总运行时间为0秒。这是程序的主要功能函数。
- 剩下的函数都被调用了2到28次，总运行时间都为0秒。

从分析结果可以看出，程序的瓶颈主要在`cal_max_product`函数中。因此，如果需要进一步优化程序，就应该重点关注该函数。

3.6 单元测试的用例设计思路、覆盖率 (选择一种覆盖指标)、测试通过率分别是多少?

根据函数 `cal_max_product` 的逻辑:

- 输入:一个整数数组
- 输出:一个整数,代表乘积最大的非空连续子数组的乘积
- 行为:找到乘积最大的非空连续子数组
- 异常:如果输入为空数组或所有元素都为0,抛出ValueError异常

因此,可以设计以下测试用例:

1. 输入数组为空数组
    - 输入:[]
    - 预期输出:抛出ValueError异常
2. 输入数组中有多个连续的0
    - 输入:[1, 2, 0, 3, 0, 4]
    - 预期输出:4。
3. 输入数组所有元素都为0
    - 输入:[0, 0, 0]
    - 预期输出:0
4. 输入数组中只有正数
    - 输入:[3, 4, 1 ,6]
    - 预期输出:72。
5. 输入数组中只有负数
    - 输入:[-2, -3, -5]
    - 预期输出:15。
6. 输入数组中只有一个正数
    - 输入:[3]
    - 预期输出:3。
7. 输入数组中只有一个负数
    - 输入:[-2]
    - 预期输出:-2。

8. 输入数组有正数和负数，且乘积最大的非空连续子数组包含正数和负数
    - 输入：[2, 3, -2, 4, -1]
    - 预期输出：6 * (-2) * 4 * (-1) = 48
9. 输入数组有正数和负数，且乘积最大的非空连续子数组只包含正数
    - 输入：[1, 2, 3, -1, 4]
    - 预期输出：1 * 2 * 3 = 6
10. 输入数组有正数和负数，且乘积最大的非空连续子数组只包含负数
    - 输入：[-2, -3, -1, -4]
    - 预期输出：（-2） * （-3） * （-1） * （-4） = 24
11. 输入数组有正数和负数，且乘积最大的非空连续子数组只包含一个数
    - 输入：[1, -1, 2, -2, 3, -3]
    - 预期输出：36

整理如下：

	A	B
1	**输入**	**预期输出**
2	[]	抛出ValueError异常
3	[1, 2, 0, 3, 0, 4]	4
4	[0, 0, 0]	0
5	[3, 4, 1 ,6]	72
6	[-2, -3, -5]	15
7	[3]	3
8	[-2]	-2
9	[2, 3, -2, 4, -1]	48
10	[1, 2, 3, -1, 4]	6
11	[-2, -3, -1, -4]	24
12	[1, -1, 2, -2, 3, -3]	36

增加的测试代码如下：

```
import unittest

class Tests(unittest.TestCase):
    # 创建test_为前缀的方法
    def test_1(self):
```

```
6           self.assertEqual(maxProduct([]), ValueError)
7
8       def test_2(self):
9           self.assertEqual(maxProduct([1, 2, 0, 3, 0, 4]), 4)
10
11      def test_3(self):
12          self.assertEqual(maxProduct([0, 0, 0]), 0)
13
14      def test_4(self):
15          self.assertEqual(maxProduct([3, 4, 1, 6]), 72)
16
17      def test_5(self):
18          self.assertEqual(maxProduct([-2, -3, -5]), 15)
19
20      def test_6(self):
21          self.assertEqual(maxProduct([3]), 3)
22
23      def test_7(self):
24          self.assertEqual(maxProduct([-2]), -2)
25
26      def test_8(self):
27          self.assertEqual(maxProduct([2, 3, -2, 4, -1]), 48)
28
29      def test_9(self):
30          self.assertEqual(maxProduct([1, 2, 3, -1, 4]), 6)
31
32      def test_10(self):
33          self.assertEqual(maxProduct([-2, -3, -1, -4]), 24)
34
35      def test_11(self):
36          self.assertEqual(maxProduct([1, -1, 2, -2, 3, -3]), 36)
37
38  #unittest.main()作为主函数入口
39  if __name__ == '__main__':
40      print(maxProduct([2, 3, -2, 4]))
41      unittest.main()
```

运行结果如图：

```
PS D:\MyFiles\software engineering\编程作业> python3 .\test.py
6
...........
----------------------------------------------------------------------
Ran 11 tests in 0.001s

OK
```

测试通过率为100%。