南開大學
Nankai University

计算机学院

软件工程实验报告

软件编码实现、分析和测试

姓名：徐文斌

学号：2010234

专业：计算机科学与技术

2023 年 5 月 14 日

目录

1 问题描述及分析

1.1 问题描述

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

1.2 问题分析

解决该问题的一个较为朴素的思路是暴力枚举数组的每一个子数组，计算每个子数组的乘积，从而得到乘积最大的子数组，时间复杂度为 $O(n^3)$。

考虑效率更高的算法，我们考虑数组的第 i 个元素 $nums[i]$，如果其是一个正数，我们希望以它前一个位置结尾的某个段的积也是个正数，并且希望该段的乘积尽可能地大；如果其是一个负数，那么我们希望以它前一个位置结尾的某个段的积也是个负数，这样就可以负负得正，并且我们希望该段的乘积尽可能小。显然，我们可以使用动态规划算法来解决该问题，用 $f_{max}(i)$ 来表示以第 i 个元素结尾的乘积最大的子数组的乘积，用 $f_{min}(i)$ 表示以第 i 个元素结尾的乘积最小的子数组的乘积。我们可以得到动态规划转移方程如下：

$$f_{max}(i) = max\{f_{max}(i-1) \times nums[i], f_{min}(i-1) \times nums[i], nums[i]\} \tag{1}$$

$$f_{min}(i) = min\{f_{max}(i-1) \times nums[i], f_{min}(i-1) \times nums[i], nums[i]\} \tag{2}$$

$f_{max}(i)$ 中的最大值即为最终答案。

2 代码实现

2.1 MaxProductSubArray 类

2.1.1 成员变量

- self.array: 用于存储待处理的数组
- self.legal: 用于存储数组是否合法

代码实现如下：

```
1  class MaxProductSubArray:
2      """
3      用于计算数组中最大乘积子数组
4      """
5      def __init__(self):
6          """
7          初始化函数
8          array为数组，legal表示数组是否合法
9          """
10         self.array = []
11         self.legal = True
```

2.1.2 成员函数 set_array

根据传入的参数选择从标准输入/文件/参数中读取数组至 self.array 中,最后调用 self.array_is_legal 函数对数组合法性进行检查。如果传入参数 source 为 None，表示从标准输入读取数组，调用 ___read_array_from_stdin 函数;如果 source 为字符串,表示从文件读取数组,调用 ___read_array_from_file 函数；否则从 source 中读取数组。代码实现如下：

```
def set_array(self, source=None):
    """
    根据参数初始化self.array数组
    :param source:
        可以传入一个列表或字符串或为空，如果传入字符串，会从对应的文件中读取数组；
    如果为空，则从标准输入读取
    :return: None
    """
    input_array = source
    # 在从文件读取的时候不对数组进行类型检查
    if source is None:
        input_array = self.__read_array_from_stdin()
    elif isinstance(source, str):
        input_array = self.__read_array_from_file(source)
    self.array = input_array
    # 最终调用array_is_legal统一进行数组检查
    self.legal = self.array_is_legal()
```

2.1.3 成员函数 ___read_array_from_stdin

该函数从标准输入读取数组，并返回读取到的数组，在读取过程中使用 literal_eval 进行初步的合法性判断。代码实现如下：

```
def __read_array_from_stdin(self):
    """
    从标准输入中读取数组
    :return: 读取到的数组
    """
    res_list = self.array
    try:
        res_list = literal_eval(input("请输入数组，输入格式参照[1, 2 ,3, 4]: "))
    except (SyntaxError, ValueError):
        print("输入格式不合法")
        res_list = None
    return res_list
```

2.1.4 成员函数 ___read_array_from_file

该函数从文件中读取数组，并返回读取到的数组，在读取过程中使用 literal_eval 进行初步的合法性判断。代码实现如下：

```
def __read_array_from_file(self, file_path):
    """
    从文件中读取数组
    :param file_path: 文件路径
    :return: 返回读取到的数组
    """
    res_list = self.array
    with open(file=file_path, mode='r', encoding='UTF-8') as file:
        try:
            res_list = literal_eval(file.read())
        except(SyntaxError, ValueError):
            print("文件读取失败!")
            res_list = None
    return res_list
```

2.1.5 成员函数 array_is_legal

用于判断 self.array 是否是一个可以处理的合法数组，目前仅支持一维数组。实现代码如下：

```
def array_is_legal(self):
    """
    检查self.array是否合法，不为空，且元素是int/float
    :return: True表示合法，False表示不合法
    """
    res = True
    if isinstance(self.array, list):
        if not self.array:
            res = False
            print("输入数组为空")
        else:
            try:
                tmp = np.array(self.array)
            except ValueError:
                res = False
                print("数组格式错误")
                return res
            if len(tmp.shape) == 1:
                for ele in self.array:
                    try:
                        assert isinstance(ele, (int, float))
                    except AssertionError:
                        res = False
                        print("数组元素类型非法!")
                        return res
            else:
                res = False
                print("暂不支持多维数组")
    else:
        res = False
```

```
31          print("输入数组不是以列表形式")
32      return res
```

2.1.6 成员函数 solve

用户可以调用该函数来对由 set_array 设置的数组进行求解，该函数首先判断 self.legal 是否为 True，若为 True，则调用相应的求解函数，否则报错。实现代码如下：

```
def solve(self):
    """
    计算数组最大乘积子数组，用户接口
    :return: 最大乘积以及对应的子数组内容
    """
    if not self.legal:
        print("错误，无法计算")
        return None
    res = self.__solve()
    return res
```

2.1.7 成员函数 ___solve

考虑到后期计算多维数组的可能性，这里定义 ___solve 函数，该函数根据数组的维度来调用不同的函数，返回计算结果，这里我们没有添加对于多维数组的支持，处理多维数组时简单地打印提示。实现代码如下：

```
def __solve(self):
    """
    根据数组的维度来进行对应的计算
    :return: 最大乘积以及对应的子数组
    """
    res = None
    tmp = np.array(self.array)
    if len(tmp.shape) == 1:
        res = self.__on_solve(self.array)
    else:
        print("多维数组暂未支持")
    return res
```

2.1.8 成员函数 ___on_solve

该函数是我们在问题分析部分提出的 $O(N)$ 算法的具体实现，其可以对一维数组进行处理。实现代码如下：

```
def __on_solve(self, nums):
    """
    计算一维数组最大乘积子数组O(N)算法
    :param nums: 待处理的一维数组
```

```
    :return: 最大乘积以及对应的子数组内容
    """
    if not self.legal:
        return None
    res_val = min(nums)
    nums_len = len(nums)
    f_max = [0] * nums_len
    f_min = [0] * nums_len
    f_max[0] = nums[0]
    f_min[0] = nums[0]
    res_list = []
    for i in range(1, nums_len):
        f_max[i] = max(f_max[i - 1] * nums[i], f_min[i - 1] * nums[i], nums[i])
        f_min[i] = min(f_max[i - 1] * nums[i], f_min[i - 1] * nums[i], nums[i])
        if f_max[i] > res_val:
            res_val = f_max[i]
    for i in range(nums_len):
        if f_max[i] == res_val:
            tmp_val = nums[i]
            res_list.append(nums[i])
            for j in range(i - 1, -1, -1):
                if tmp_val == res_val and nums[j] != 1:
                    break
                tmp_val = tmp_val * nums[j]
                res_list.append(nums[j])
            break
    res_list.reverse()
    return res_val, res_list
```

3 编程规范

采用 PEP8 规范，参考网址为：https://legacy.python.org/dev/peps/pep-0008/。使用 pylint 对代码规范进行检查，创建 try_pylint.py 文件，代码如下：

```
import pylint.lint

pylint_opt = ['-ry', './max_product_sub_array.py']
pylint.lint.Run(pylint_opt)
```

运行该文件，可以看到我们的代码完全符合 PEP8 的规范要求。

```
Report
======
111 statements analysed.

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

+---------+-------+-----------+-----------+------------+---------+
|type     |number |old number |difference |%documented |%badname |
+=========+=======+===========+===========+============+=========+
|module   |1      |1          |=          |100.00      |0.00     |
+---------+-------+-----------+-----------+------------+---------+
|class    |1      |1          |=          |100.00      |0.00     |
+---------+-------+-----------+-----------+------------+---------+
|method   |8      |8          |=          |100.00      |0.00     |
+---------+-------+-----------+-----------+------------+---------+
|function |2      |2          |=          |100.00      |0.00     |
+---------+-------+-----------+-----------+------------+---------+



External dependencies
---------------------
::

    numpy (max_product_sub_array)



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

+----------+-------+------+---------+-----------+
|type      |number |%     |previous |difference |
+==========+=======+======+=========+===========+
|code      |117    |61.58 |117      |=          |
+----------+-------+------+---------+-----------+
|docstring |55     |28.95 |55       |=          |
+----------+-------+------+---------+-----------+
|comment   |3      |1.58  |3        |=          |
+----------+-------+------+---------+-----------+
|empty     |15     |7.89  |15       |=          |
+----------+-------+------+---------+-----------+
```

```
Duplication
-----------

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



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

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




--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

4 程序可扩展性

4.1 数据扩展

数据方面，我们主要考虑了数据来源的扩展、数据规模的扩展以及数据形式的扩展。

4.1.1 数据来源

考虑我们如何得到数据，我们可以从用户输入、文件、网络、数据库等来源处得到我们需要处理的数组，因此这里我们将得到数据的这一行为封装成 set_array 成员函数，该函数根据传入参数的不同调用不同的函数，从不同的源头处获得数据。如代码实现部分所展示，我们目前实现了从用户输入、文件、函数参数处获取待处理的数组，之后如果想对获取数组这一行为进行扩展，我们只需要实现新的相应的数据读取函数，并将该函数添加到 set_array 函数的一个新的分支中即可。如下所示为添加从 web 读取数组功能的样例实现：

```
def set_array(self, source=None):
    input_array = source
    if source is None:
        input_array = self.__read_array_from_stdin()
    elif isinstance(source, str):
        input_array = self.__read_array_from_file(source)
    # 如果要添加从web读取数组这一功能，只需实现新的数据读取函数
    # 并将其添加到set_array的新分支即可，如下所示
    elif isUrl(source):
        input_array = self.__read_array_from_web(source)
    self.array = input_array
    self.legal = self.array_is_legal()
```

4.1.2 数据规模

这里为了支持对任意长度的数组以及多维数组的处理，我们选择使用 list 类型来存储数组，使用 list 类型在理论上可以对数组的维度和数组长度进行无限延伸，从而支持不同数据规模的处理。

4.1.3 数据形式

数据形式方面，分为数组元素的类型以及数组的维度。类型方面，由于使用 list 类型，可以实现浮点数和整数的等多种类型的混合数组。数组维度方面，虽然我们目前只实现了一维数组的处理，但是我们也考虑了对于多维数组的扩展。如在成员函数 array_is_legal和成员函数 ___solve 中预留了多维数组的处理分支，当我们需要扩展至多维数组时，只需在对应的分支处添加实现即可。

4.2 算法实现方式扩展

考虑到在未来，我们还有可能对最大乘积子数组算法的具体实现方式进行更改，我们定义成员函数 ___solve提供统一的抽象，当我们修改或者添加新的算法实现方式的时候，我们只需要在 ___solve 函数中调用新的处理函数即可，而不需要更改其他的函数的代码。

4.3 结果呈现扩展

除了返回数组的最大乘积子数组的乘积的值之外，我还实现了得到该最大乘积子数组的具体内容，用户调用 solve 函数便可以得到子数组内容以及乘积。未来可以考虑将结果可视化地呈现给用户。

5 两个原则

5.1 单一职责原则 (SRP)

我对 MaxProductSubArray 类的成员函数的设计符合单一职责原则。如 set_array 函数只负责读取数组；array_is_legal 函数只负责判断数组是否合法；___solve 函数只负责数组求解。且对于不同数组读取方式以及不同的问题求解算法也编写到了不同的函数中，从而保证对某一个功能的修改只会导致单一函数的代码发生改动，符合单一职责原则。

5.2 开放-封闭原则 (OCP)

MaxProductSubArray 类的成员函数 ___on_solve 的作用是求出一维数组的最大子数组乘积，这一部分内容是不必进行更改的，对于更高维度的数组的求解可以通过调用该函数来实现，是可以扩展的，满足开放-封闭原则。

6 错误处理

首先我们应该规定要处理的数组的格式，我们规定数组应该以列表的形式输入，数组元素为 int 或 float 类型。同时目前只支持一维数组的求解，对于多维数组应当给出报错提示。

首先，我们在 ___read_array_from_stdin 和 ___read_array_from_file 这两个成员函数中会通过 literal_eval 函数来初步判断读入的数据是否是 python 中的合法的数据类型，如果判断失败则会打印提示。如 2.1.3节所示。

但是凭借 literal_eval 只能在一定程度上对输入数组进行检测，我们还需要调用 array_is_legal 函数来进一步判断输入数组的合法性。在该函数中，我们首先判断数组是否为 list 格式，如果不是则提示错误。然后判断数组是否为空，若为空同样提示错误。最后，判断数组的维度，如果维度大于一，由于程序还没有实现对高维数组的处理，这里也提示错误；如果数组维度为一，则遍历数组每个元素判断元素类型是否为 int/float，若不是，则提示错误。最终我们根据 array_is_legal 的返回值来赋值 self.legal。当 self.legal 为 False 时，调用类的 solve 函数将会提示错误。具体请参考2.1.5和2.1.6小节。

7 性能分析

算法复杂度的分析我们在文章开头处已经给出，这里不再赘述。下面我们使用 profile 对程序性能进行分析。添加性能分析函数 profile_test 如下所示：

```
1 def profile_test(scale):
2     """
3     性能测试函数
4     :param scale: 测试数组规模
5     :return: None
6     """
```

```
    test_array = [np.random.randint(-10, 10) for i in range(scale)]
    mpsa = MaxProductSubArray()
    mpsa.set_array(test_array)
    mpsa.solve()
```

接下来，新建 try_profile.py 文件，内容如下：

```
import profile
import max_product_sub_array as mpsa

profile.run("mpsa.profile_test(1000000)")
```

运行 try_profile.py 文件，结果如下图：

```
F:\PyCharm_Project>python try_profile.py
         4000256 function calls in 4.172 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      237    0.000    0.000    0.000    0.000 :0(append)
        2    0.031    0.016    0.031    0.016 :0(array)
        1    0.000    0.000    4.156    4.156 :0(exec)
  1000002    0.312    0.000    0.312    0.000 :0(isinstance)
        3    0.000    0.000    0.000    0.000 :0(len)
   999999    0.562    0.000    0.562    0.000 :0(max)
  1000000    0.656    0.000    0.656    0.000 :0(min)
  1000000    0.797    0.000    0.797    0.000 :0(randint)
        1    0.000    0.000    0.000    0.000 :0(reverse)
        1    0.016    0.016    0.016    0.016 :0(setprofile)
        1    0.000    0.000    4.156    4.156 <string>:1(<module>)
        1    0.000    0.000    0.641    0.641 max_product_sub_array.py:126(set_array)
        1    0.000    0.000    2.297    2.297 max_product_sub_array.py:143(solve)
        1    0.000    0.000    4.156    4.156 max_product_sub_array.py:155(profile_test)
        1    0.000    0.000    0.000    0.000 max_product_sub_array.py:16(__init__)
        1    0.422    0.422    1.219    1.219 max_product_sub_array.py:161(<listcomp>)
        1    1.062    1.062    2.281    2.281 max_product_sub_array.py:24(__on_solve)
        1    0.016    0.016    2.297    2.297 max_product_sub_array.py:85(__solve)
        1    0.297    0.297    0.641    0.641 max_product_sub_array.py:98(array_is_legal)
        1    0.000    0.000    4.172    4.172 profile:0(mpsa.profile_test(1000000))
        0    0.000             0.000          profile:0(profiler)
```

可以看到运行时间集中在 ___on_solve 函数，即我们计算一维数组最大乘积子数组的函数。我们应该集中优化该函数，具体的优化思路在文章开头已经给出，即使用时间复杂度为 $O(N)$ 的动态规划算法替代原来的时间复杂度为 $O(N^3)$ 的暴力算法。进一步优化的话可能可以考虑使用一些并行算法来加速该函数的运行。除此之外，我们可以使用滚动数组来优化动态规划算法的内存占用。

8 单元测试

对我们的代码进行黑盒测试。设计样例考虑如下：

- 首先对于数据的正负值，我们设计了全为正数、全为负数以及正数和负数混合的测试样例；
- 对于数据类型，我们设计了全为整数、全为浮点数以及整数和浮点数混合的测试样例；
- 对于数组长度，我们设计了数组长度为 1 的极端数据情况；
- 设计了对从标准输入读入数组功能和从文件读入数组功能的测试样例；

- 同时我们也进行了错误输入的测试，如输入的不是列表、输入格式错误、数组元素不是 int/float、数组为空、输入的数组维度大于 1 等。

具体测试样例如下：

```python
import unittest
import max_product_sub_array as mpsa
from unittest.mock import patch


class TestMPSA(unittest.TestCase):
    """
    单元测试类
    """

    def test_common_case(self):
        # 测试数组长度为1
        self.assertEqual(mpsa.unit_test([2])[0], 2)
        self.assertEqual(mpsa.unit_test([-2])[0], -2)
        # 测试全为正整数
        self.assertEqual(mpsa.unit_test([2, 3, 4])[0], 24)
        # 测试全为负整数
        self.assertEqual(mpsa.unit_test([-2, -3, -4])[0], 12)
        # 测试正整数负整数混合
        self.assertEqual(mpsa.unit_test([2, 3, -2, 4])[0], 6)
        # 测试浮点数
        self.assertEqual(mpsa.unit_test([2.0, 3.0, -2.0, 4.0])[0], 6)
        # 测试浮点数整数混合
        self.assertEqual(mpsa.unit_test([2.0, 3.0, -2, 4])[0], 6)

    @patch('builtins.input')
    def test_read_from_stdin(self, mock_input):
        """
        测试标准输入
        """
        mock_input.return_value = '[2]'
        self.assertEqual(mpsa.unit_test()[0], 2)
        mock_input.return_value = '[-2]'
        self.assertEqual(mpsa.unit_test()[0], -2)
        mock_input.return_value = '[2, 3, 4]'
        self.assertEqual(mpsa.unit_test()[0], 24)
        mock_input.return_value = '[-2, -3, -4]'
        self.assertEqual(mpsa.unit_test()[0], 12)
        mock_input.return_value = '[2, 3, -2, 4]'
        self.assertEqual(mpsa.unit_test()[0], 6)
        mock_input.return_value = '[2.0, 3.0, -2.0, 4.0]'
        self.assertEqual(mpsa.unit_test()[0], 6)
        mock_input.return_value = '[2.0, 3.0, -2, 4]'
        self.assertEqual(mpsa.unit_test()[0], 6)

```

```
46      def test_read_from_file(self):
47          """
48          测试从文件读入数组
49          样例和test_read_from_stdin一样
50          """
51          self.assertEqual(mpsa.unit_test("test_cases/case1.txt")[0], 2)
52          self.assertEqual(mpsa.unit_test("test_cases/case2.txt")[0], -2)
53          self.assertEqual(mpsa.unit_test("test_cases/case3.txt")[0], 24)
54          self.assertEqual(mpsa.unit_test("test_cases/case4.txt")[0], 12)
55          self.assertEqual(mpsa.unit_test("test_cases/case5.txt")[0], 6)
56          self.assertEqual(mpsa.unit_test("test_cases/case6.txt")[0], 6)
57          self.assertEqual(mpsa.unit_test("test_cases/case7.txt")[0], 6)
58
59      @patch('builtins.input')
60      def test_error(self, mock_input):
61          """
62          测试错误检测
63          """
64          # 输入数组为空
65          self.assertEqual(mpsa.unit_test([]), None)
66          # 输入不是列表
67          self.assertEqual(mpsa.unit_test((1, 2, 3)), None)
68          # 输入格式错误
69          mock_input.return_value = "[1, 2, 3"
70          self.assertEqual(mpsa.unit_test(), None)
71          self.assertEqual(mpsa.unit_test("test_cases/error_case.txt"), None)
72          # 数组元素不是int/float
73          self.assertEqual(mpsa.unit_test(['a', "b", (1, 2)]), None)
74          self.assertEqual(mpsa.unit_test(['a', "b", 'c']), None)
75          # 数组维度大于一
76          self.assertEqual(mpsa.unit_test([[1, 2, 3], [4, 5, 6]]), None)
77
78
79  if __name__ == "__main__":
80      suite = unittest.TestSuite()
81      tests = [TestMPSA("test_common_case"), TestMPSA("test_read_from_stdin"),
82               TestMPSA("test_read_from_file"), TestMPSA("test_error")]
83      suite.addTests(tests)
84      runner = unittest.TextTestRunner()
85      runner.run(suite)
```

```
Ran 4 tests in 0.005s

OK

Process finished with exit code 0
```

由上图可见，我们的测试通过率为 100%，在终端执行如下命令：

```
coverage run try_unittest.py
coverage html -d my_coverage_result
```

生成语句覆盖率报告，如下所示：

Coverage report: 93%

coverage.py v7.2.2, created at 2023-05-14 11:45 +0800

Module	statements	missing	excluded	coverage
max_product_sub_array.py	112	11	0	90%
try_unittest.py	52	0	0	100%
Total	**164**	**11**	**0**	**93%**

coverage.py v7.2.2, created at 2023-05-14 11:45 +0800

代码中未被覆盖的语句主要为用于性能测试的 profile_test 函数以及 main 函数，这两个函数并不需要被测试，在单元测试时也没有被调用，因此未被覆盖。其余部分的代码大致均已被覆盖，只有几条为了后续扩展预留的代码没有被覆盖。