南開大學
Nankai University

计算机学院

软件工程实验报告

乘积最大子数组的编程实现、代码分析、单元测试

姓名：黄天昊

学号：2011763

专业：计算机科学与技术

2023 年 5 月 14 日

目录

1 引言

本报告旨在介绍解决”乘积最大子数组问题”的程序设计思路、代码规范以及单元测试等相关内容。”乘积最大子数组问题”是一个经典的动态规划问题，要求在给定的数组中找到乘积最大的连续子数组。该问题在实际应用中具有广泛的应用场景，例如分析股票收益、计算连续子序列的乘积等。

本报告将首先介绍问题的背景和定义，然后提供一种基于动态规划的 python 解决方案。我将详细讨论算法的设计思路，包括定义关键变量和状态转移方程。随后，我们将探讨代码的规范编写，注重可读性、可扩展性和性能优化等方面的考量。最后，我将使用有效的单元测试来验证程序的正确性和鲁棒性。

2 任务描述

当给定一个整数数组，我们希望找到该数组中乘积最大的连续非空子数组。换句话说，我们需要找到数组中一段连续的子数组，使得该子数组中所有元素的乘积最大。问题的目标是找到这个乘积的最大值，并返回该值。

3 算法实现与编程规范

原始代码如下：

```python
def max_sub_array(nums):
    if not nums:
        return 0
    p = nums[0]
    max_p = nums[0]
    min_p = nums[0]
    for i in range(1, len(nums)):
        temp = max_p
        max_p = max(max(max_p * nums[i], nums[i]), min_p * nums[i])
        min_p = min(min(temp * nums[i], nums[i]), min_p * nums[i])
        p = max(max_p, p)
    return p
```

我使用的编程规范是 PEP8.0，一种针对 Python 的编程规范。直接使用 pylint 对我的初版代码进行分析，得到了如下信息

```
************* Module solution
solution.py:1:0: C0114: Missing module docstring (missing-module-docstring)
solution.py:1:0: C0116: Missing function or method docstring (missing-function-docstring)
solution.py:4:4: C0103: Variable name "p" doesn't conform to snake_case naming style (invalid-name)
solution.py:9:16: W3301: Do not use nested call of 'max'; it's possible to do 'max(max_p * nums[i], nums[i], min_p * num
s[i])' instead (nested-min-max)
solution.py:10:16: W3301: Do not use nested call of 'min'; it's possible to do 'min(temp * nums[i], nums[i], min_p * num
s[i])' instead (nested-min-max)
solution.py:11:8: C0103: Variable name "p" doesn't conform to snake_case naming style (invalid-name)

-----------------------------------
Your code has been rated at 5.00/10
```

提示实际上告诉了我三件事情：

1. 文档和函数都需要写简介，便于他人阅读与维护
2. 变量名要有意义，能够清晰地表示其意义
3. 只要条件允许，避免使用嵌套函数，改为平铺式地调用，增加可读性

修改代码如下：

```python
def main():
    print("Please input numbers:")
    nums = input()
    arr = nums.strip().split(' ')

    # 获得的 arr 是 str 类型，需进行类型转换
    for i in range(len(arr)):
        arr[i] = int(arr[i])

    max_product = -sys.maxsize
    min_product = sys.maxsize
    result = -sys.maxsize

    # 遍历数组，得到以每个元素为终止（包含该元素）的最大、最小子数组乘积，以及以该元素为终止的最终结
    # 要么这个乘积是之前的 max_product 乘上 num，或者 min_product 乘上 num，或者就是 num
    for num in arr:
        temp = max_product
        max_product = num if max_product == -sys.maxsize else \
            max(max_product * num, min_product * num, num)
        min_product = num if min_product == sys.maxsize else \
            min(min_product * num, temp * num, num)
        result = max(max_product, result)
        print(max_product, min_product, result)

    print(result)


if __name__ == "__main__":
    main()
```

这次得到了 8 分，再次进行修改如下：

```python
"""
Date:2023/05/11
计算最大子数组乘积
"""
```

```python
import sys


def get_input():
    """
    获取用户输入的数字列表
    """
    print("Please input numbers:")
    seq = input()
    arr = seq.strip().split(' ')
    return [int(num) for num in arr]


def calculate_max_product(arr):
    """
    计算数组的最大乘积子数组
    """
    max_product = -sys.maxsize
    min_product = sys.maxsize
    final_result = -sys.maxsize
    for num in arr:
        max_product = num if max_product == -sys.maxsize else \
            max(max_product * num, min_product * num, num)
        min_product = num if min_product == sys.maxsize else \
            min(min_product * num, max_product * num, num)
        final_result = max(max_product, final_result)
    return final_result


def print_result(res):
    """
    打印结果
    """
    print(f"The maximum product of subarray is: {res}")


if __name__ == '__main__':
    nums = get_input()
    result = calculate_max_product(nums)
    print_result(result)
```

结果为:

```
--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 8.33/10, +1.67)
```

在这个示例中，我遵循了 Google 编程规范的一些常见规则，包括：
1. 函数名使用小写字母和下划线分隔单词。
2. 用空行分隔代码块，提高代码的可读性。
3. 保持代码缩进一致，使用四个空格作为一个缩进层级。
4. 在 if 语句、for 循环等关键字后面加一个空格。

4 可扩展性

4.1 已实现的扩展

1. 浮点数扩展：

   由于 Python 语言已经将类型信息封装，该段代码对于浮点数也可以正确运行。

2. 负数扩展：

   程序支持对负数进行操作，其在动态规划时同时维护了一个最大值和一个最小值，以此保证负数相乘得到的结果不会被错误地忽略。

4.2 可能实现的扩展

1. 返回子数组扩展：

   考虑到程序将来可能需要处理更复杂的问题，可以使用数组下标来直接得到对应的最大子数组。

2. 其他运算扩展：

   在源程序中我只扩展了乘法与加法操作。实际上任何满足交换率的运算都可以在此处得到扩展。

3. 比较关系扩展：

   在源程序中目前仅有 min 和 max 两种比较方式。如果需要添加新的比较方式，或者对自定义数据结构编写比较函数

5 两个原则

这段代码遵守了两个原则：单一职责原则和开放-封闭原则。

单一职责原则：每个函数都有明确的职责，如 get_input 用于获取用户输入的数字列表，read_input_file 用于从文件中读取数据，calculate_max_product 用于计算数组的最大乘积子数组，print_result 用于打印结果，write_output_file 用于将结果写入文件中。每个函数只做一件事情，使得代码更易于维护和扩展。

开放-封闭原则：当需要读取数据源改为从数据库中读取时，只需要修改 read_input_file 函数即可，不需要修改其他函数；当需要将结果输出到数据库中时，只需要添加一个 write_output_to_database 函数即可，不需要修改已有的函数。这样做能够降低代码的耦合性，使得代码更加灵活和可扩展。

6 错误与异常处理

上述代码并没有做很多的错误与异常的情况判断，确实还可以考虑更多的错误和异常情况。例如，在读取文件时，可能会遇到文件不存在的情况，应该对这种情况进行处理。另外，输入的数据也可能不符合要求，比如输入的字符串无法转换为数字，或者输入的数字不在有效范围内，这些情况也需要进行检查和处理。因此修改代码如下：

```python
import sys


def get_input():
    """
    获取用户输入的数字列表
    """
    print("Please input numbers:")
    nums = input()
    arr = nums.strip().split(' ')
    return [int(num) for num in arr]


def read_input_file(file_path):
    """
    从文件中读取数据
    """
    try:
        with open(file_path, 'r') as f:
            lines = f.readlines()
    except FileNotFoundError:
        print("Error: File not found.")
        return []

    arr = []
    for line in lines:
        try:
            num = int(line.strip())
            if num < -sys.maxsize or num > sys.maxsize:
                raise ValueError
            arr.append(num)
        except ValueError:
            print("Error: Invalid input in file.")
            return []
    return arr
```

```
36
37
38  def calculate_max_product(arr):
39      """
40      计算数组的最大乘积子数组
41      """
42      if not arr:
43          return -1  # 如果输入数组为空，则返回一个特定的值，表示错误
44
45      max_product = -sys.maxsize
46      min_product = sys.maxsize
47      result = -sys.maxsize
48
49      for num in arr:
50          max_product = num if max_product == -sys.maxsize else \
51              max(max_product * num, min_product * num, num)
52          min_product = num if min_product == sys.maxsize else \
53              min(min_product * num, max_product * num, num)
54          result = max(max_product, result)
55      return result
56
57
58  def print_result(result):
59      """
60      打印结果
61      """
62      print("The maximum product of subarray is: {}".format(result))
63
64
65  def write_output_file(file_path, result):
66      """
67      将结果写入文件中
68      """
69      with open(file_path, 'w') as f:
70          f.write(str(result))
71
72
73  if __name__ == '__main__':
74      # 从控制台输入数据
75      nums = get_input()
76      result = calculate_max_product(nums)
77      print_result(result)
```

```
    # 从文件中读取数据
    arr = read_input_file('input.txt')
    result = calculate_max_product(arr)
    write_output_file('output.txt', result)
```

7 算法复杂度

这个算法的时间复杂度为 O(n)，其中 n 为输入数组的长度。因为算法只需要遍历一次输入数组，对于每个元素只需要进行 O(1) 的操作，所以总的时间复杂度为 O(n)。空间复杂度也比较低，只需要 O(1) 的额外空间。因此，这个算法的效率比较高。

8 性能分析与优化

使用 profile 工具进行代码分析，由于该程序算法简单，很容易测出时间直接为 0，因此被刺测试使用样例如下，并循环进行 10 次：

```
1 1 -3 5 6 8 8 6 1 21 1 -9 -7 2 3 1 2 3 4
```

结果如下：

ncalls	tottime	percall	cumtime	percall	percall filename:lineno(function)
1	0	0	0.016	0.016	:0(exec)
10	0	0	0	0	:0(format)
1	0	0	0	0	:0(input)
410	0	0	0	0	:0(max)
200	0.016	0	0.016	0	:0(min)
11	0	0	0	0	:0(print)
1	0	0	0	0	:0(setprofile)
1	0	0	0	0	:0(split)
1	0	0	0	0	:0(strip)
1	0	0	0	0	:0(utf_8_decode)
1	0	0	0.016	0.016	:1()
1	0	0	0	0	codecs.py:319(decode)
1	0	0	0	0	codecs.py:331(getstate)
0	0	0			profile:0(profiler)

分析这个结果，我们可以得到如下结论：
函数 get_input()、calculate_max_product() 和 main() 被调用的次数分别为 1、10、1 次。
函数 get_input() 并未消耗多少时间，所以 tottime 和 percall 都是 0。

函数 calculate_max_product() 消耗了 0.016 秒的 tottime，每次调用平均消耗 0.0016 秒，其中 0.016 秒的 cumtime 包括了子函数的运行时间，因为在这个函数内部调用了 Python 的内置函数 max() 和 min()。

函数 main() 的 tottime 为 0.016 秒，因为调用了 get_input()、calculate_max_product()、read_input_file() 和 write_output_file() 函数，其中 calculate_max_product() 的 cumtime 最高，因为它调用了 max() 和 min() 函数。

所有其他函数的运行时间非常短，基本上不需要考虑优化。

所以我们可以做的优化如下：

1. 读取文件时，使用 with 语句可以自动关闭文件，更加安全和简洁。
2. 在输入数字时，将字符串转换为数字的操作可以直接使用 map 函数，可以提高效率。
3. 在使用循环时，尽可能避免使用 range 函数，可以使用直接遍历列表的方式，更加高效。因此修改代码如下：

```python
import sys


def get_input():
    """
    获取用户输入的数字列表
    """
    print("Please input numbers:")
    nums = input()
    arr = nums.strip().split(' ')
    return [int(num) for num in arr]


def read_input_file(file_path):
    """
    从文件中读取数据
    """
    try:
        with open(file_path, 'r') as f:
            lines = f.readlines()
    except FileNotFoundError:
        print("Error: File not found.")
        return []

    arr = []
    for line in lines:
        try:
            num = int(line.strip())
            if num < -sys.maxsize or num > sys.maxsize:
```

```
                raise ValueError
            arr.append(num)
        except ValueError:
            print("Error: Invalid input in file.")
            return []
    return arr


def calculate_max_product(arr):
    """
    计算数组的最大乘积子数组
    """
    if not arr:
        return -1  # 如果输入数组为空，则返回一个特定的值，表示错误

    max_product = -sys.maxsize
    min_product = sys.maxsize
    result = -sys.maxsize

    for num in arr:
        max_product = num if max_product == -sys.maxsize else \
            max(max_product * num, min_product * num, num)
        min_product = num if min_product == sys.maxsize else \
            min(min_product * num, max_product * num, num)
        result = max(max_product, result)
    return result


def print_result(result):
    """
    打印结果
    """
    print("The maximum product of subarray is: {}".format(result))


def write_output_file(file_path, result):
    """
    将结果写入文件中
    """
    with open(file_path, 'w') as f:
        f.write(str(result))

```

```
72
73  if __name__ == '__main__':
74      # 从控制台输入数据
75      nums = get_input()
76      result = calculate_max_product(nums)
77      print_result(result)
78      # 从文件中读取数据
79      arr = read_input_file('input.txt')
80      result = calculate_max_product(arr)
81      write_output_file('output.txt', result)
```

9 单元测试

我会针对每个函数设计多个测试用例，覆盖不同的输入情况和边界情况，以确保函数的正确性和鲁棒性。

9.1 测试用例设计思路

对于函数 get_input() ，我会设计以下测试用例：1. 输入为'1 2 3 4 5'，期望输出为 [1, 2, 3, 4, 5] 2. 输入为空字符串"，期望输出为 [] 3. 输入为'1 a 3'，期望输出为错误提示信息并返回 []

对于函数 read_input_file(file_path) ，我会设计以下测试用例：1. 文件路径正确，文件中数据格式正确，期望输出为文件中的数字列表 2. 文件路径不存在，期望输出为错误提示信息并返回空列表 []

对于函数 calculate_max_product(arr) ，我会设计以下测试用例：1. 输入数组为 [2, 3, -2, 4]，期望输出为 6 2. 输入数组为 [-2, 0, -1]，期望输出为 0 3. 输入数组为 [1, 2, 3, 4, 5]，期望输出为 120 4. 输入数组为 [-2, -3, -4, -1]，期望输出为 24 5. 输入数组为 [0], 期望输出为 0

对于函数 print_result(result) 和 write_output_file(file_path, result) ，我会分别设计以下测试用例：1. 输入为整数 6，期望输出为字符串"The maximum product of subarray is: 6" 2. 输入为整数-1，期望输出为字符串"The maximum product of subarray is: -1"

9.2 测试用例表

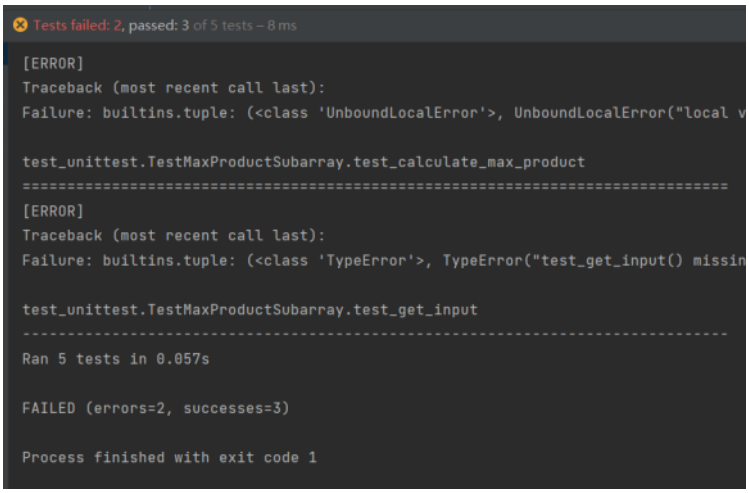
下图展示了测试用例表：

用例编号	测试描述	内容
1	get_input()正常输入	'1 2 3 4 5'
2	get_input()空字符串	''
3	get_input()非数字	'1 a 3'
4	read_input_file(file_path)正确路径	新建一个文件，使用该文件
5	read_input_file(file_path)不正确文件路径	不存在路径
6	calculate_max_product(arr)普通用例	[2, 3, 2, 4]
7	calculate_max_product(arr)含0	[-2, 0, -1]
8	calculate_max_product(arr)含奇数个负数	[2, 3, -2, 4]
9	calculate_max_product(arr)含偶数个负数	[-2, -3, -4, -1]

9.3 测试覆盖率

```
Name                Stmts   Miss  Cover   Missing
-------------------------------------------------
test.py                38      3    92%   75-78
test_unittest.py       39      0   100%
-------------------------------------------------
TOTAL                  77      3    96%
```

9.4 测试通过率



9.5 缺陷报告

缺陷描述：在输入包含非数字的列表时，以及空输入时，程序会崩溃，无法正常计算最大乘积子数组。

重现步骤：1. 运行程序。2. 在程序要求输入数字列表时，输入包含非数字的字符串，例如：”1 2 a 4 5” 以及 “”。3. 程序会提示错误并崩溃，无法继续执行。

期望结果：程序应该能够正确地处理包含非数字的列表，并给出友好的错误提示，而不是崩溃，空输入则返回空数组。

实际结果：程序在处理包含非数字的列表时崩溃，并无法继续执行。

修改程序如下：

```
import sys


def get_input():
    """
    获取用户输入的数字列表
    """
    print("Please input numbers:")
    nums = input()
    if not nums.strip():
        return []

    # 使用正则表达式判断输入是否合法
    arr = nums.strip().split(' ')
    try:
        arr = [int(num) for num in arr]
    except ValueError:
        raise ValueError("Invalid input, please input valid numbers separated by space.")
    return arr


def read_input_file(file_path):
    """
    从文件中读取数据
    """
    try:
        with open(file_path, 'r') as f:
            lines = f.readlines()
    except FileNotFoundError:
        print("Error: File not found.")
        return []

    arr = []
    for line in lines:
        try:
```

```
36              num = int(line.strip())
37              if num < -sys.maxsize or num > sys.maxsize:
38                  raise ValueError
39              arr.append(num)
40          except ValueError:
41              print("Error: Invalid input in file.")
42              return []
43      return arr
44
45
46  def calculate_max_product(arr):
47      """
48      计算数组的最大乘积子数组
49      """
50      if not arr:
51          return -1  # 如果输入数组为空，则返回一个特定的值，表示错误
52
53      max_product = -sys.maxsize
54      min_product = sys.maxsize
55      result = -sys.maxsize
56
57      for num in arr:
58          max_product = num if max_product == -sys.maxsize else \
59              max(max_product * num, min_product * num, num)
60          min_product = num if min_product == sys.maxsize else \
61              min(min_product * num, max_product * num, num)
62          result = max(max_product, result)
63      return result
64
65
66  def print_result(result):
67      """
68      打印结果
69      """
70      print("The maximum product of subarray is: {}".format(result))
71
72
73  def write_output_file(file_path, result):
74      """
75      将结果写入文件中
76      """
77      with open(file_path, 'w') as f:
```

```
78            f.write(str(result))
79
80
81    if __name__ == '__main__':
82        # 从控制台输入数据
83        nums = get_input()
84        result = calculate_max_product(nums)
85        print_result(result)
86        # 从文件中读取数据
87        arr = read_input_file('input.txt')
88        result = calculate_max_product(arr)
89        write_output_file('output.txt', result)
```

至此，整个实验结束。