计算机科学与技术学院 数据科学概论 课程实报告

|  |  |  |
| --- | --- | --- |
| 实验题目：Python控制结构和函数编程练习 | | 学号：202200130041 |
| 日期：2023.3.2 | 班级：2022级数据班 | 姓名：左景萱 |
| Email：zuojingxuan1130@mail.sdu.edu.cn | | |
| 实验目的：  本实验首先安装配置好Anaconda以及Python环境，通过代码编写，从而使同学们熟练掌握条件判断、循环语句和函数使用，并学会将其运用到实践中 | | |
| 实验软件和硬件环境：  1）操作系统：Windows11；  2）Anaconda 版本：5.3.0；  3）Python 版本：3.6.8； | | |
| 实验原理和方法：  学习python的基础语法，条件语句，循环语句与函数相关知识，并利用相关知识编写代码解决寻找第n个尼莫森数的问题。 | | |
| 实验步骤：（不要求罗列完整源代码）  1.首先利用jupyter文件中提供的示例学习了python中if elif else条件语句，接着学习了while循环和for循环，range函数。针对数组和set我接着了解了迭代器的概念，map和filter函数。接着学习了python中函数的定义和调用，同时学习了python中具有特色的higher order function。在学习函数的同时了解了函数的形参，可变参数和参数关键字。然后学习了python中的格式化输出。最后了解了变量的作用域，global和nonlocal关键字。  2.接着对于尼莫森数的问题，首先需要一个质数判断函数prime，判断输入的num是不是质数，然后需要通过一个nimosen函数去寻找第n个尼莫森数，并设置相关的循环条件。按照模板写出的代码与结果如下：   1. import time 2. def prime(num): 3. if num==2: 4. return True 5. i=2 6. while pow(i,2)<=num: 7. if num%i!=0: 8. i+=1 9. else: 10. return False 11. return True 12. def monisen(no): 13. i=0 14. j=2 15. while i<no: 16. if prime(j) and prime(pow(2,j)-1): 17. j+=1 18. i+=1 19. else: 20. j+=1 21. return pow(2,j-1)-1 22. if \_\_name\_\_ =='\_\_main\_\_': 23. for i in range(4): 24. time\_s=time.time() 25. print(monisen(int(input()))) 26. time\_e=time.time() 27. print(time\_e-time\_s)     接着考虑到质数，我写了个欧拉筛法进行处理，代码和结果如下：   1. import time 2. def monisen(no): 3. set2 = {2} 4. set1 = {2} 5. for i in range(2, 100001): 6. if i not in set1: 7. set2.add(i) 8. for j in set2: 9. set1.add(j\*i) 10. if i % j == 0: 11. break 12. j=0 13. for i in set2: 14. if (pow(2,i)-1) in set2: 15. j+=1 16. if j==no: 17. return pow(2,i)-1 18. if \_\_name\_\_ == '\_\_main\_\_': 19. for i in range(4): 20. time\_s=time.time() 21. print(monisen(int(input()))) 22. time\_e=time.time() 23. print(time\_e-time\_s)     慢的惊人。  考虑到加速问题，我又从网上了解到了taichi加速，并打算利用最原始的代码进行修改尝试（因为taichi不支持set，所以第二种方法并没有进行改进，也不想再写二分查找进行第二种方法的taichi加速了），得到结果如下：    不难发现对比原来的方法，不仅没有加速，反而更慢了。  对于这三种方法的效果差异，我进行了思考。 | | |
| 结论分析与体会：  我们不难发现对于寻找第n个尼莫森数的问题，最快的是最朴素的第一种方法，nlogn复杂度的欧拉筛反而是最慢的。这是为什么呢？因为其实对于尼莫森数而言，由于答案是指数级变化的，所以在不超过int范围内遍历所有数的时间是非常短的，因为数其实非常少，不超过16\*2个。而欧拉筛必须要对范围内的每一个数进行判断，就相当的慢了。  针对taichi加速慢于原来代码的情况，是因为taichi惊人的加速性能只能在多重循环或者大循环中体现的明显，而在批量极小的循环中，将循环转换到gpu上，cuda上运行本来就要花很多时间，所以会显得更慢一点但是对于一些百万级的循环，taichi的加速效果就相当明显了。  所以在处理特定问题的时候，并不是复杂强大的算法就一定好，需要根据数据范围进行合理的判断，选择最合适的实现方式。 | | |
| 就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟1－3道问答题：  问题：为什么四次循环输出当中第一次循环特别的慢？  答：通过询问助教可知，这与python的jit机制以及一些内存机制有关。Python作为一门解释型语言往往会特别的慢，而jit机制就可以在多次执行相同代码的情况下加速运算。当编译器发现某个方法或代码块运行特别频繁的时候，就会认为这是“热点代码”（Hot Spot Code)。然后JIT会把部分“热点代码”编译成本地机器相关的机器码，并进行优化，然后再把编译后的机器码缓存起来，以备下次使用。所以后面几次循环相较于第一次循环就少了编译的工作，所以就会更加快速。 | | |