# 第1节 数据结构与算法基础

## 1 算法的引入

### 学习目标

1. 了解算法的含义; 2. 了解算法的特征

### 1.1 实例

例1. 用纸和笔计算 的值，精确到小数点后四位（根号10 与计算的结果的差要小于等于0.0001 ）。(例:计算面积为10的方形之边长)

方法一思路：

1. 第一步：先找到两个数其中一个数A的平方小于10,另一个数B的平方大于10，比如找得A=3, B=4

2. 第二步：计算A和B的平均值C,并计算C的平方 (检查是否达到精度，没有达到精度，则执行第三步，否则终止程序)

3. 第三步: 比较C的平方与10的大小关系，如果大于10,将B的值用C替换；如果C的平方小于10,则将A的值用C替换。

4. 重复第二步。

方法一代码：

1. **def** sr(a):
2. x0 = 1.0
3. x2 = 10.0
4. i = 0
5. **while** 1:
6. x1 = (x2+x0)/2
7. **print**("第", i+1, "步：")
8. **if** x1\*\*2 < 10:
9. x0 = x1
10. **print**("x0=",x0)
11. **print**("x0 \* x0 = ", x0\*\*2)
12. **if** (x0\*\*2 -a)\*(x0\*\*2 -a) <0.00000001:
13. **break**
14. i = i + 1
15. **else**:
16. x2 = x1
17. **print**("x2=",x2)
18. **print**("x2 \* x2 = ", x2\*\*2)
19. **if** (x2\*\*2 -a)\*(x2\*\*2 -a) <0.00000001:
20. **break**
21. i = i + 1
22. **return** x1
23. **print**("sr: ",sr(10))

方法二思路:

1. 第一步: 选择一个试探值

2. 第二步: 计算

3. 第三步: 求的平方，, 判断是否达到所要精度

4. 第四步: 重设试探值

5. 第五步: 重复第二步

方法二代码：

1. **def** sqrt1(a):
2. x = 3
3. i = 0
4. **while** 1:
5. x = (x+ a/x)/2
6. **print**("第", i+1, "步：")
7. **print**("%f^2=%f" % (x, x\*\*2))
8. **if** (x\*\*2 - a)\*(x\*\*2 - a) < 0.00000001:
9. **break**
10. i = i + 1
11. **return** x
12. **print**("sqrt1:", sqrt1(10))

更好的代码:

1. **def** sqrt2(a):
2. x = 3
3. **while** (x\*\*2 - a)\*(x\*\*2 - a) > 0.00000001:
4. x = (x + a/x)/2
5. **return** x
6. **print**(sqrt(10))

我们可以打印出其计算的步骤:

1. **def** sqrt2(a):
2. x = 3
3. i = 0
4. **while** (x\*\*2 - a)\*(x\*\*2 - a) > 0.00000001:
5. x = (x + a/x)/2
6. **print**("第", i+1, "步：")
7. **print**("%f^2=%f" % (x, x\*\*2))
8. i = i + 1
9. **return** x
10. **print**(sqrt2(10))

分析：

1. 按方法一计算10的平方根，需14次循环达到所要的精度;

2. 按方法二来计算时，需2次循环，达到所要的精度。

总结：

1. 解决问题的方法有多种;

2. 为达到同样的目的，不同方法所需要步骤的数目不一样;

3. 为了减少步骤的数目，选择解决问题方法非常重要。

### 1.2 什么是算法？

解题方案的准确而完整的描述，是一系列解决问题的清晰指令。 简单地说，算法就是解决问题的思路。

注意：

1. 算法是解决问题的思路。

2. 算法与编程语言无关，它独立于编程语言.

### 1.3 算法的特征

1. 有穷性：算法必须能在执行有限步骤之后终止。

2. 确切性：算法的每一步骤必须有确切的定义。

3. 输入: 一个算法有0个或多个输入，以刻画运算对象的初始情况。

4. 输出: 一个算法有一个或多个输出，给出对输入数据加工后的结果。没有输出的算法毫无意义。

5. 有效性: 算法中执行的每一个步骤都是可以被分解为基本的可执行的操作步，即每个步骤都可以在有限时间内完成。

### 1.4 小结

1. 从上述例子可见算法的重要性

2. 结合上述思路理解算法的特征

## 2 时间复杂度

### 学习目标

1. 掌握常见时间复杂度的大小关系

2. 了解基本操作数量的估算

### 2.1 如何衡量一个算法的优劣？

上面两种方法都可以计算的值。为了得到相同精度，它们所需要执行的计算次数是不一样的。

问: 那一个方法更好？

不同计算机执行同一算法所用的时间可能不同，但是如果它们执行同一算法，它们执行的基本操作的数量是一样的。

所以，我们可用程序执行的基本操作的数量来衡量一个算法的优劣。

可以引入时间复杂度的概念。

### 2.2 什么是时间复杂度？(掌握)

算法的时间复杂度指执行该算法所需要的基本操作的数量.

分析：　基本操作包括：

算术运算：加减乘除等运算；

逻辑运算：或、且、非等运算；

数据传输：输入、输出、赋值等运算;

关系运算：大于、小于、等于、不等于等运算

例2：计算上面两种方法计算10的平方根精确到小数点后４位所需要的基本操作的数量。

方法一代码：

1. #方法一
2. a = 10
3. x0 = 1.0
4. x2 = 10.0
5. i = 0
7. **while** 1:
8. x1 = (x2+x0)/2
9. **print**("第", i+1, "步：")
10. **if** x1\*\*2 < 10:
11. x0 = x1
12. **print**("x0=",x0)
13. **print**("x0 \* x0 = ", x0\*\*2)
14. **if** (x0\*\*2 -a)\*(x0\*\*2 -a) <0.00000001:
15. **break**
16. i = i + 1
17. **else**:
18. x2 = x1
19. **print**("x2=",x2)
20. **print**("x2 \* x2 = ", x2\*\*2)
21. **if** (x2\*\*2 -a)\*(x2\*\*2 -a) <0.00000001:
22. **break**
23. i = i + 1

赋初值：3次

循环内的操作数量：7次

循环次数：15次

基本操作数量: T = 7\*15 +3 = 108 次

方法二代码：

1. a = 10.0
2. x0 = 3
4. i = 0
5. **while** 1:
6. x1 = (x0+ a/x0)/2
7. **print**("第", i+1, "步：")
8. **print**("%f^2=%f" % (x1, x1\*\*2))
9. **if** (x1\*\*2 - a)\*(x1\*\*2 - a) < 0.00000001:
10. **break**
11. x0 =x1
12. i = i + 1

赋初值：2次

循环内的操作数量：5次

循环次数：2次

基本操作数量:T = 2\*5+2 = 12 次

**例3**. 如果正整数a ,b, c满足条件 , 且 , 求a, b, c的所有可能组合.

解: 我们先写出程序，再计算时间复杂度。

方法一：

1. #方法一
2. **for** a **in** range(1001):
3. **for** b **in** range(1001):
4. **for** c **in** range(1001):
5. **if** a + b + c == 1000 **and** a\*\*2 + b\*\*2 == c\*\*2:
6. **print** "a,b,c:", a, b, c

基本操作的数量： 1000 \* 1000 \* 1000 \*10 = n \* n \* n \*10 = 10n^3

方法二

1. #方法二
2. **for** a **in** range(1001):
3. **for** b **in** range(1001):
4. c = 1000 - a - b
5. **if** a\*\*2 + b\*\*2 == c\*\*2:
6. **print** "a,b,c:", a, b, c

基本操作的数量：1000 \* 1000 \*9

### 2.3 如何表示时间复杂度？

(1) 问题：如上例中，如果遇到的条件是

,

或者是

,

或者对任意的整数,

,

我们该如何计算出基本操作的数量？

方法一基本操作数量:

2000: 2000\*2000\*2000\*10

3000: 3000\*3000\*3000\*10

任意正整数:　 (的三次函数)

方法二基本操作数量:

2000: 2000\*2000\*9

3000: 3000\*3000\*9

任意正整数:　　(的二次函数)

所以，我们可以把基本操作数量写成一个待解决问题的数量规模的一个函数：

这就是时间复杂度。

(2)可以计算出算法一和算法二的时间复杂度：

算法一:

算法二:

(3)数量级的概念

纳微毫厘分十百千兆吉太

(4) 大O记法

实践中关心的是时间复杂度的数量级. 例2中,方法一的时间复杂度可以记为:

方法二的时间复杂度可以记为：

**习题1**. 写一个Python程序, 计算从1到100的所有奇数之和.

**习题2**. 计算下列算法执行得最频繁的操作的数量. 并写出该算法的时间复杂度(用大O记法表示, 已知) :

1. # -\*- coding: utf-8 -\*-
2. n = 1000
3. x = 0
5. **for** i **in** range(n):
6. **for** j **in** range(i):
7. **for** k **in** range(j):
8. x = x + 1
10. **print**("x ={}".format(x))

### 2.4 为什么需要时间复杂度？

1. 时间复杂度可用来判断程序执行的效率。

2. 用时间复杂度可衡量算法的优劣，从而有助于我们写出优质代码。

### 2.5 常见时间复杂度和大小关系

常见的时间复杂度 举例

常数阶:

线性阶:

平方阶: )

对数阶:

阶 :

立方阶:

指数阶:

常见时间复杂度的大小关系 (掌握)

) < < < < < ) <

习题2. 计算下列算法的时间复杂度.

1. i = 2
3. **while** i < n:
4. i = i\*\*2

### 2.6 小结

1. 时间复杂度的含义.(理解)

2. 如何评价一个算法的优劣(了解)

3. 常见时间复杂度的大小关系(掌握)

## 3 数据结构的引入

### 学习目标

1. 掌握数据结构的概念

2. 了解数据结构与算法的时间复杂度之间存在关系

### 3.1 引入

用Python保存一个班级学生的信息，包括学生的姓名，年龄，出生地.

#### 3.1.1 以什么类型保存数据？

1. 列表嵌套元组

>>>li = [("Zhang",22,"Chengdu"),("Li",20,"Chongqing"),("Qian",20,"Nanjing")]

1. 列表嵌套字典

>>>li1 = [{"name":"Zhang","age": 22, "hometown":"Chengdu"}, {"name":"Li","age": 20, "hometown":"Chongqing"},{"name":"Qian","age": 20, "hometown":"Nanjing"}]

1. 字典嵌套字典

>>>li2 = {"Zhang":{"age":22,"hometown":"Chendu"},"Li":{"age":22,"hometown":"Chendu"},"Qian":{"age":20,"hometown":"Chendu"}}

#### 3.1.2 如何查找某学生的信息

查找并打印出姓名为”Qian”的同学的信息.

列表嵌套元组：

**课堂习题**: 对li, 查找并打印出姓名为”Qian”的同学的信息.

1. # -\*- coding: utf-8 -\*-
2. """
3. 找出并打印某个名字的同学的信息.
4. """
5. li = [("Zhang",22,"Chengdu"),("Li",20,"Chongqing"),("Qian",20,"Nanjing")]
6. count = 0
7. **for** i **in** range(len(li)):
8. **if** li[i][0] == "Qian":
9. **print**(li[i])
10. count = count + 1
11. **break**
12. **else**:
13. count = count + 1
15. **print**("搜索的次数= ",count)

列表嵌套字典：

**课堂习题**: 对li1, 查找并打印出姓名为”Qian”的同学的信息.

>>> li1 = [{"name":"Zhang","age": 22, "hometown":"Chengdu"}, {"name":"Li","age": 20, "hometown":"Chongqing"},{"name":"Qian","age": 20, "hometown":"Nanjing"}]

>>> li1[0]["name"]

'Zhang'

>>> li1[2]["age"]

20

>>>

字典嵌套字典：

**课堂习题**: 对li2, 查找并打印出姓名为”Qian”的同学的信息.

>>> li2 = {"Zhang":{"age":22,"hometown":"Chendu"},"Li":{"age":20,"hometown":"Chendu"},"Qian":{"age":20,"hometown":"Nanjing"}}

>>> li2["Zhang"]["age"]

20

>>> li2["Li"]["hometown"]

'Chendu'

>>>

例. 查找姓名为”Qian”的同学的记录. 写出程序来估算时间复杂度.

1. # -\*- coding: utf-8 -\*-
2. """
3. 找出并打印某个名字的同学的信息.
4. """
5. # 1列表嵌套元组. 假设你知道姓名在元组中的位置
6. li = [("Zhang",22,"Chengdu"),("Li",20,"Chongqing"),("Qian",20,"Nanjing")]
7. count = 0
8. **for** i **in** range(len(li)):
9. **if** li[i][0] == "Qian":
10. **print**(li[i])
11. count = count + 1
12. **break**
13. **else**:
14. count = count + 1
16. **print**("列表嵌套字典, 搜索的次数= ",count)
18. # 1’列表嵌套元组. 假设你不知道姓名在元组中的位置
19. li = [("Zhang",22,"Chengdu"),("Li",20,"Chongqing"),("Qian",20,"Nanjing")]
20. count = 0
21. **for** i **in** range(len(li)):
22. **for** j **in** range(len(li[i])):
23. **if** li[i][j] == "Qian":
24. **print**(li[i])
25. count = count + 1
26. **break**
27. **else**:
28. count = count + 1
30. **print**("列表嵌套字典, 搜索的次数= ",count)
32. # 2列表嵌套字典.
33. li1 = [{"name":"Zhang", "age": 22, "hometown":"Chengdu"},
34. {"name":"Li", "age": 20, "hometown":"Chongqing"},
35. {"name":"Qian", "age": 20, "hometown":"Nanjing"}]
36. count = 0
37. **for** i **in** range(len(li)):
38. **if** li1[i]["name"] == "Qian":
39. **print**(li1[i])
40. count = count + 1
41. **break**
42. **else**:
43. count = count + 1
45. **print**("列表嵌套字典. 搜索的次数= ",count)
47. # 3字典嵌套字典.
48. li2 = {"Zhang":{"age":22,"hometown":"Chendu"},
49. "Li":{"age":22,"hometown":"Chendu"},
50. "Qian":{"age":20,"hometown":"Chendu"}}
51. count = 0
52. **if** li2["Qian"] **is** **not** None:
53. **print**(li2["Qian"])
54. count = count + 1
55. **else**:
56. count = count + 1
58. **print**("列表嵌套字典. 搜索的次数= ",count)

总结：算法的时间复杂度与用什么类型保存数据有密切关系。

### 3.2 什么是数据结构？

数据结构是对基本数据类型的封装。数据结构确定一组数据如何保存. 例如上例子中，列表和字典都分别是一种数据结构。

分析：

1. 基本数据类型有: int, float, str等

2. 不同的数据结构，导致算法的时间复杂度不同.

3. Python数据结构举例：列表，元组，字典，集合.

注意：

列表，元组，字典等常用序列都不是基本数据类型。

### 3.3　列表中操作的时间复杂度

现在我们看对列表的不同操作的时间复杂度.

#### 3.3.1 实例

例4. 构造新列表的如下三个操作：向列表尾添加元素、列表相加和插入元素

li.append()

li + li2

li.insert()

哪一个的效率更高？代码如下：

1. **import** timeit
2. **from** timeit **import** Timer
4. **def** t1():
5. li = []
6. **for** i **in** range(1000):
7. li.append(i)
9. **def** t2():
10. li =[]
11. **for** i **in** range(1000):
12. li = li + [i]
13. **def** t3():
14. li = []
15. **for** i **in** range(1000):
16. li.insert(0,i)
18. timer1 =Timer("t1()", "from \_\_main\_\_ import t1")
19. **print** (timer1.timeit(100))
21. timer2 =Timer("t2()", "from \_\_main\_\_ import t2")
22. **print** (timer2.timeit(100))
24. timer3 =Timer("t3()", "from \_\_main\_\_ import t3")
25. **print** (timer3.timeit(100))

运行结果：

0.0151388645

0.2677149772

1.0190680027

分析：

1. 同一台计算机上运行，我们用操作所耗费的时间来衡量操作的效率
2. 列表的三个操作，insert方法耗时最多，append方法耗时最少
3. 操作列表时，一般选择append方法,而不原则列表“+“操作

思考：

为什么列表的不同操作效率差别巨大？

这是由列表的存储方式决定的.

#### 3.3.2 列表常见操作的时间复杂度

index O(1) eg: li = [4，1,2,3]; li[2]

append O(1)

insert(i) O(N), N为列表长度

pop() O(1)

pop(i) O(N)

contains O(N) eg: li=[1,2,3]; 2 in li

iteration O(N)

get slice [x:y] O(k)，k=y-x

set slice O(N+k)

del slice O(N)

sort() O(NlogN)

reverse O(N)

#### 3.3.3 字典常见内置操作的时间复杂度

index O(1)

contains O(1) 例: dict ={"name":"Zhang","age":23}; "age" in dict

iteration O(N)

get item O(1)

set item O(1)

del item O(1)

##### 3.3.4小结：

1. 列表和字典的内置方法是对基本操作步骤的封装,而非基本操作步骤

2. 列表(字典)的方法各有不同的操作步骤数量，因此具有不同的时间复杂度

3. 不同的方法的效率有差别

4. 列表和字典不属于基本数据类型

### 3.4 算法与数据结构有什么关系？

- 算法关注解决问题的思路,不关心处理怎样的数据.

- 数据结构关注待解决问题中的数据该如何保存.

- 程序=算法＋数据结构

## 4 习题

**习题1**. 写一个Python程序, 计算从1到100的所有奇数之和.

参考解:

1. # -\*- coding: utf-8 -\*-
2. sum = 0
3. **for** i **in** range(101):
4. **if** i % 2 == 1:
5. sum = sum + i
6. **print**(sum)

**习题2**. 计算下列算法执行得最频繁的操作的数量. 由此算出该算法的时间复杂度=\_\_\_\_\_\_\_\_. (用大O记法表示, 已知)

1. # -\*- coding: utf-8 -\*-
2. n = 100
3. x = 0
5. **for** i **in** range(n):
6. **for** j **in** range(i):
7. **for** k **in** range(j):
8. x = x + 1
10. **print**("x ={}".format(x))

解:

**习题3**. 计算下列算法的时间复杂度. (用大O记法表示)

1. n = int(input("Please input an integer:\n"))
2. i = 2
4. **while** i < n:
5. i = i\*\*2
6. **print**(i)

**习题4**. 输入一个数,判断该数是否为素数. 输出结果.

参考解一(能用):

1. # -\*- coding: utf-8 -\*-
2. x = int(input("Please input an integer:\n"))
3. count = 0
5. **for** i **in** range(1,x+1):
6. **if** **not** x % (i) == 0:
7. count = count + 1
8. **print**("x=",x, "i=",i, "x%i=", x%i)
9. **print**(count)
10. **if** count + 2 == x:
11. **print**("{} is a prime number.".format(x))
12. **else**:
13. **print**("{} is a not prime number.".format(x))

参考解二(更好):

1. **import** math
2. n = int(input("Please input an integer:\n"))
4. x = int(math.sqrt(n))
5. i, w = 2, 0
6. **for** i **in** range(2, x+1):
7. **if** n % i == 0:
8. w = 1
9. **if** w == 1:
10. **print**('{} is not a prime number.'.format(n))
11. **else**:
12. **print**('{} is a prime number.'.format(n))

检查效率: 输入值以499为例.

1. # -\*- coding: utf-8 -\*-
2. """
3. To check if an integer is a prime number.
4. """
5. **from** timeit **import** Timer
7. **def** isprime():
8. #x = int(input("Please input an integer:\n"))
9. x = 499
10. count = 0
12. **for** i **in** range(1,x+1):
13. **if** **not** x % (i) == 0:
14. count = count + 1
15. **if** count + 2 == x:
16. **print**("{} is a prime number.".format(x))
17. **else**:
18. **print**("{} is a not prime number.".format(x))

21. **def** isprime2():
22. **import** math
23. #n = int(input("Please input an integer:\n"))
24. n = 499
25. x = int(math.sqrt(n))
26. i, w = 2, 0
27. **for** i **in** range(2, x+1):
28. **if** n % i == 0:
29. w = 1
30. **if** w == 1:
31. **print**('{} is not a prime number.'.format(n))
32. **else**:
33. **print**('{} is a prime number.'.format(n))
35. timer1 = Timer("isprime()", "from \_\_main\_\_ import isprime")
36. **print**("isprime():",timer1.timeit(20))
38. timer2 = Timer("isprime2()", "from \_\_main\_\_ import isprime2")
39. **print**("isprimer2():",timer2.timeit(20))

结果为:

**499 is a prime number.**

**…**

**isprime(): 0.0017782476203649367**

**499 is a prime number.**

**…**

**isprimer2(): 0.0005416197392236199**

## 5 参考资料

[1]<http://www.planetb.ca/syntax-highlight-word>

[2]Office中插入公式: **Alt + =**

[3]Spyder中运行程序: F5