不同的查找算法在猜数字游戏中的效率

徐晨雨

摘要: 本文分别模拟了四种算法进行“猜数字”游戏时的情况，即顺序查找、二分法查找、黄金分割法查找、随机查找。试图通过大量的数据来对比四种算法的效率，并得知此游戏的规定，即只能猜六次是否公平。

关键词： “猜数字”，查找算法，计算机模拟

此文来源于数算上机的第一个代码：随机生成一个 1-99 之间的整数，输入猜测的数字，并返回这两个数字间的相对大小，共有六次猜测机会。起初，我随意猜测数字，发现这种方法效率很低，之后又尝试了二分法，虽然效率提高了，但有几次猜对是在第六次猜测时二选一。通过简单的数学计算可知，该题目条件下用二分法猜数字至多需要猜 7 次，即满足 2n ＞ c （c 为所有可产生的数字总数）的 n 次。

随之而来的还有另外一个问题：是否存在另外一种算法，可以使得在六次之内必然能够猜出(即查找出)所给数字？此问题为典型的查找算法问题，查找算法共有四类：顺序查找，二分查找，分块查找和哈希表查找。由于问题的特殊性，即我们并不知道所要查找的数字是什么，故哈希表查找不适用于此问题，另外，也可以有另外的方法来解决此问题，其一为随机产生数字，其二也可以通过黄金分割法。

以上可行的五种算法中，分块查找最为特殊，在演算过程中的条件要做出调整，其余四种方式的算法相似。

下面我将通过实际程序的模拟，对比查找次数的期望来确定以上四种算法在求解猜数字问题过程中的效率。为了统计的方便，每次求出三十次模拟后尝试次数的期望，重复一百次，再求出总期望以及这一百组数据的方差，该方差能够在一定程度上代表尝试次数的稳定性。

一、顺序查找：

由于随机数的均匀性和对称性，取 1 为原点和取 99 为原点情况相同，不妨只研究从一开始的情况。

由于产生某一随机数概率

P(n)= 1 / 99

（n 为 1-99 的正整数）故查找次数最多为 99 次，期望为

(1+2+3+…+99)/99=50

附图 1 为程序源代码。

模拟结果见附图 2。

根据模拟结果，可得期望为 50.11，方差为 28.09。在误差允许范围内，此结果与理论计算结果相同。

二、二分法：

由于此问题涉及数字都是整数，所以我们规定二分法所得整数全部向下取整，这也与我在实际操作过程中的做法一致。此问题的理论分析并不困难，在此不再赘述。

附图 3 为程序源代码。

附图 4 为计算机模拟结果。

根据模拟结果，期望为 5.80，方差为 0.040。

三、黄金分割法

黄金分割法与二分法的思想类似，都

属于分数法。都是以每次淘汰一定比例的区间来确定随机数值的，二分法每次可以固定淘汰二分之一区间，而对于黄金分割法，每次可能淘汰 0.618 个区间，也可能淘汰 0.382 个区间。由于产生随机数的区间十分均匀，通过简单的乘法可知黄金分割法的效率略低于二分法，下面我们通过计算机模拟来对比其效率高低。

附图 5 为程序源代码。

附图 6 为计算机模拟结果。从中可得期望为 5.89，方差为 0.055。可见黄金分割法的方差与二分法处于同一数量级，且期望与二分法接近，故黄金分割法也是一种有效的方法。

四、随机数法

此方法灵感来源于数算课上的排序算法：洗牌—检查顺序，重复此步骤直至其顺序达到要求为止。这个排序算法的效率极低，最坏的情况是永远也无法得到结果。但是，对于我们的主题而言，由于每次都可以减小区间，这个循环是可以终止的。显然，最多需要 99 次就可以找到随机数值。可以预见，此算法的效率并不高，但这不失为一种有趣的算法。

附图 7 为此程序源代码。

附图 8 为计算机模拟结果。期望为 9.38，方差为 0.55。此方差比二分法和黄金分割法大一个数量级，可见其稳定性较差。

总结：通过以上的数值模拟，我们发现四种方法中二分法的效率最高，且根据附表 Sheet 2 中的具体尝试次数，我们可以得出每种方法所需尝试的最大值分别为 99、7、99、9。故无论是从多次猜测的角度亦或是单次猜测的角度，二分法都是最佳选择。在猜测次数足够多的情况下，二分法比黄金分割法略好一些。另外，由于二分法与黄金分割法中得到的期望值都小于六，所以这个游戏是有利于玩家的。

参考文献

1.http://baike.baidu.com/link?url=3LO2eNsbETOhlhoCO1g7ekVf2W2NEbVxo_GQkBXV4XJUcPwPU9dIaytqt5tmMtuYrh3kugpK17EyjLFSyaHQQnZzU5RZfyML9C3Lly_msK3Y_PMQfdho7zxCGlxGPXOQ，查找算法 _ 百度百科。

2. http://gis4g.pku.edu.cn/sessdsa-practice-2/。

3. sessdsa2017-01.pdf，陈斌，北京大学地球与空间科学学院。

附表：不同的查找算法在猜数字游戏中的效率—附表

代码：顺序查找 .py

二分法查找 .py

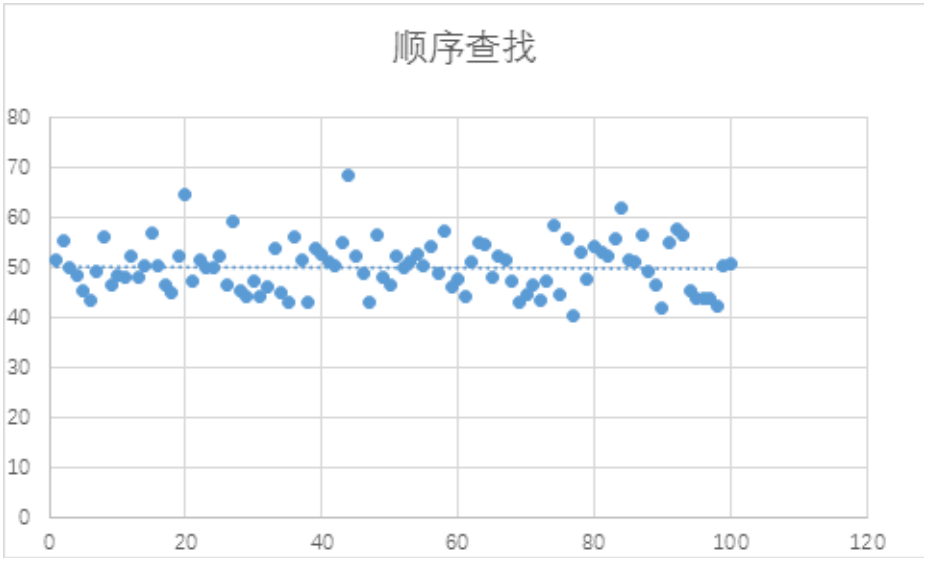
黄金分割法查找 .py

随机查找 .py

```
import random

def onetry():
    def compare():
        secret = random.randint(1,100)
        tries = 0
        guess = 1
        while tries >= 0:
            if guess==secret:
                tries += 1
                count_list.append(tries)
                break
            elif guess<secret:
                guess += 1
                tries += 1
    count_list=[]
    Sum = 0
    for i in range(1,31):
        compare()
    for i in range(0,30):
        Sum += count_list[i]
    print('%.2f'%(Sum/30,))
for j in range(0,100):
    onetry()
```

附图 1

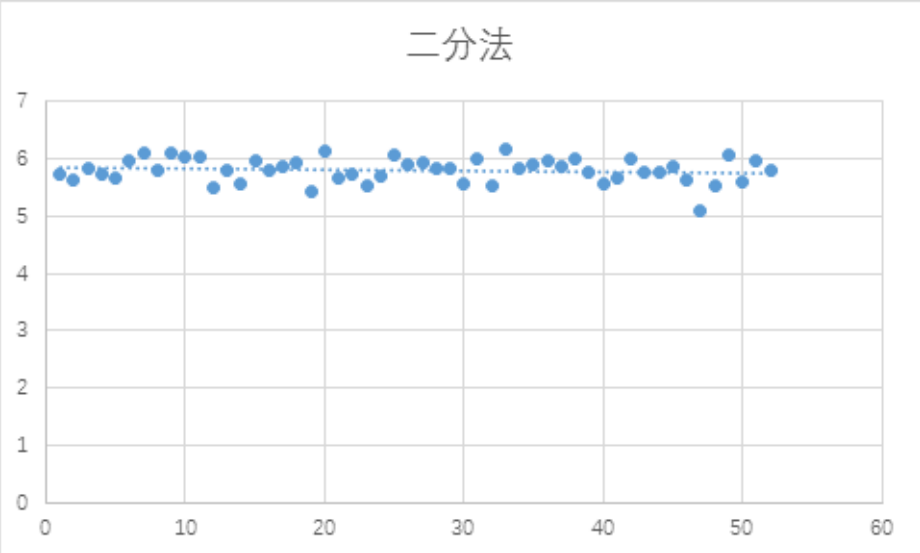


附图 2

```
import random

def onetry():
    def compare():
        secret = random.randint(1,100)
        tries = 0
        guess = 50
        bottom = 1
        top = 100
        while tries >= 0:
            if guess == secret:
                tries += 1
                count_list.append(tries)
                break
            elif guess > secret:
                top = guess
                guess = bottom + (top - bottom) // 2
                tries += 1
            elif guess < secret:
                bottom = guess
                guess = bottom + (top - bottom) // 2
                tries += 1
    count_list=[]
    Sum = 0
    for i in range(1,31):
        compare()
    for i in range(0,30):
        Sum += count_list[i]
    print('%.2f'%(Sum/30,))
onetry()
```

附图 3

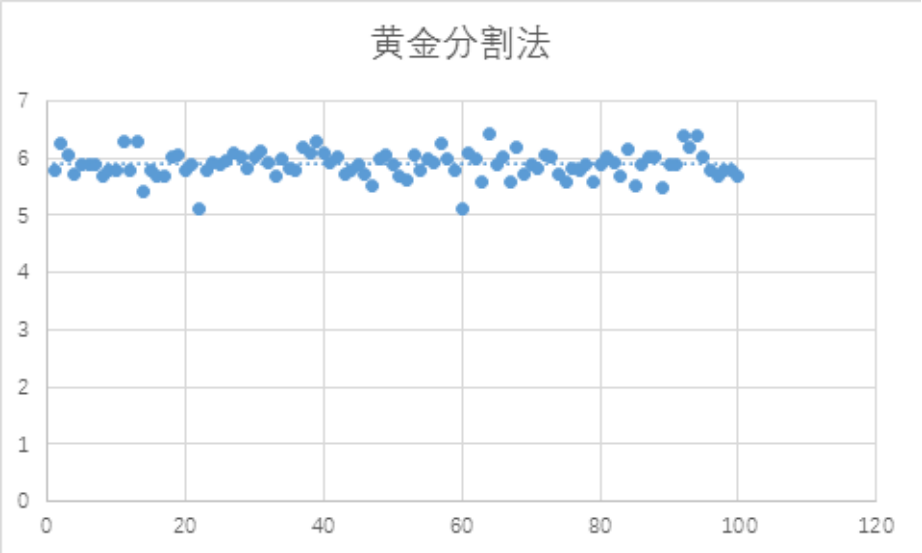


附图 4

```
import random

def onetry:
    def compare():
        secret = random.randint(1,100)
        tries = 0
        guess = 62
        bottom = 1
        top = 100
        while tries >= 0:
            if guess == secret:
                tries += 1
                count_list.append(tries)
                break
            elif guess > secret:
                top = guess
                guess = int ( bottom + (top - bottom) * 0.618 )
                tries += 1
            elif guess < secret:
                bottom = guess
                guess = int ( bottom + (top - bottom) * 0.618 )
                tries += 1
    count_list=[]
    summ = 0
    for i in range(1,31):
        compare()
    for i in range(0,30):
        summ += count_list[i]
    print('%.2f'%(summ/30,))
onetry()
```

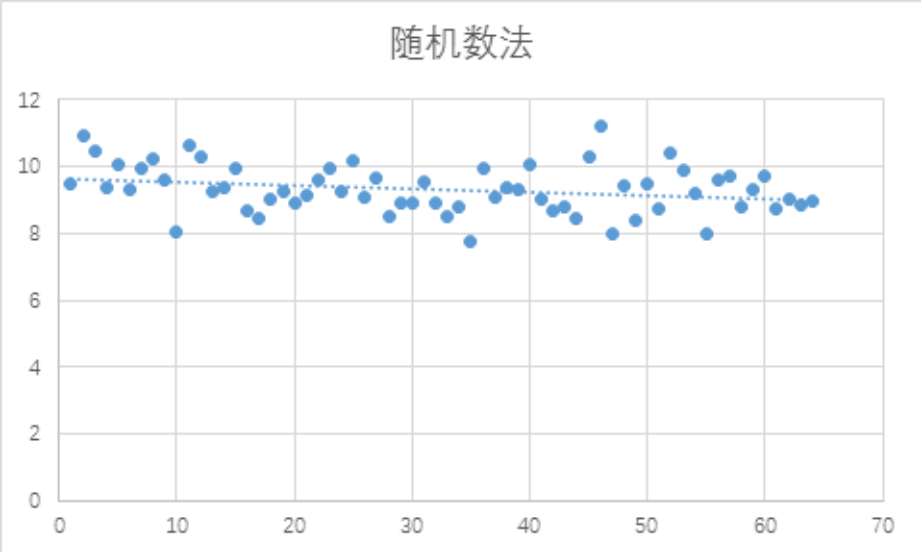
附图 5



附图 6

```
import random
def onetry():
    def compare():
        secret = random.randint(1,100)
        guess = random.randint(1,100)
        tries = 0
        bottom = 1
        top = 100
        while tries >= 0:
            if guess == secret:
                tries += 1
                count_list.append(tries)
                break
            elif guess > secret:
                top = guess
                guess = random.randint(bottom,top)
                tries += 1
            elif guess < secret:
                bottom = guess
                guess = random.randint(bottom,top)
                tries += 1
    Sum = 0
    count_list = []
    for i in range (1,31):
        compare()
    for i in range (0,30):
        Sum += count_list[i]
    print('%.2f'%(Sum/30,))
for j in range(0,100):
    onetry()
```

附图 7



附图 8