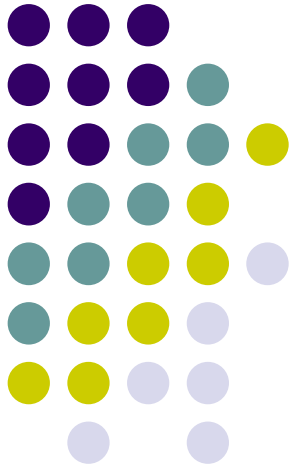
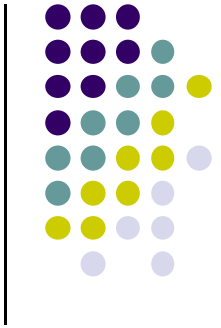
面向科学问题求解的编程实践



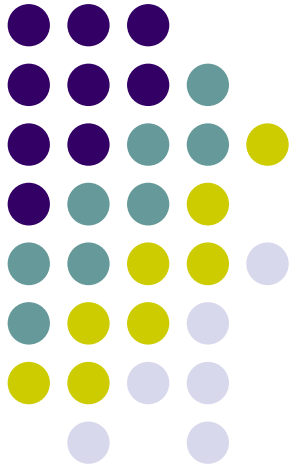
问答



- 有关上机
  - 30/20：20学时
  - 时间：周六晚上(6:30-9:30)
  - 地点：西区电三楼406
  - 内容：实践、答疑
- 有关考勤
- 有关选题
  - 推荐题目

Ponder this

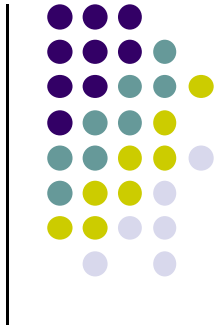
http://www.research.ibm.com/haifa/ponderthis/



Ponder this (January 2004)

- This month's puzzle was sent in by Joe Buhler. It came from a SIGCSE meeting via Eric Roberts.
- A read-only array of length n, with address from 1 to n inclusive, contains entries from the set {1, 2, ..., n-1}. By Dirichlet's Pigeon-Hole Principle there are one or more duplicated entries.
- Find a linear-time algorithm that prints a duplicated value, using only "constant extra space". (This space restriction is important; we have only a fixed number of usable read/write memory locations, each capable of storing an integer between 1 and n. The number of such locations is constant, independent of n. The original array entries can not be altered.)
- The algorithm should be easily implementable in any standard programming language.

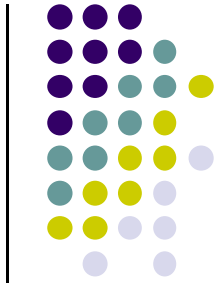
Solution



- This technique is known as Pollard's rho method; it was developed by John Pollard as a tool for integer factorization.
- Let f(x) be the location of the array at address x.

```
a := f(f(n))
b := f(n)
do while not (a=b)
a:=f(f(a))
b:=f(b)
end
/* Now a=b can be reached at either 2k or k steps from n, */
/* where k is some integer between 1 and n. */
a:=n
do while not (a=b)
a:=f(a)
b:=f(b)
end
print a
```

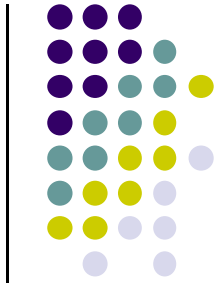
Solution



- The pattern that you get, starting from n and repeatedly applying f (doing the table lookup), is a string of some length L>0 leading to a cycle of some length M, with L+M<n. (L is nonzero because n is outside the range of f.)

- It is shaped like the Greek letter rho, hence its name. Our stopping count k is the least multiple of M greater than or equal to L, so that L \leq k < L+M < n.

Ponder this (this month)

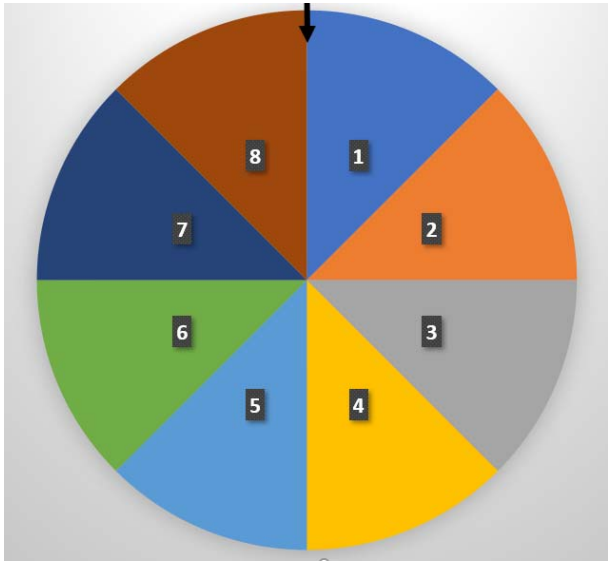


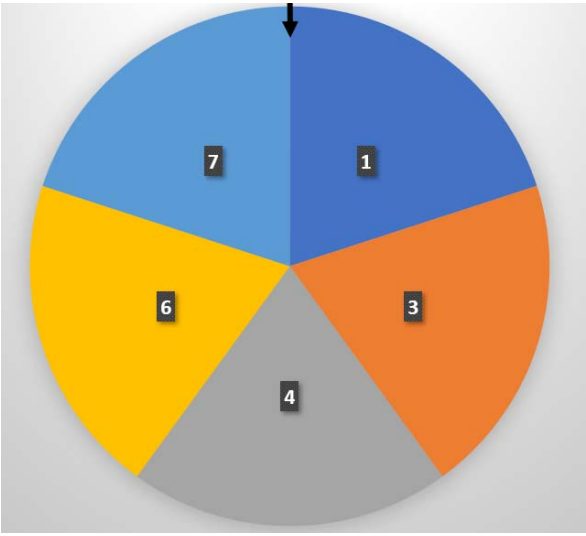
A wheel of choice is marked with the numbers 1,2,...,n, and a prize is associated with each number.

A player faced with the wheel chooses some number q and starts spinning the wheel k times. On each spin, the wheel moves forward q steps in a clockwise direction and the number / prize reached is eliminated from the wheel (i.e., the player does not get it). After k such spins, all the prizes remaining on the wheel are gifted to the player.

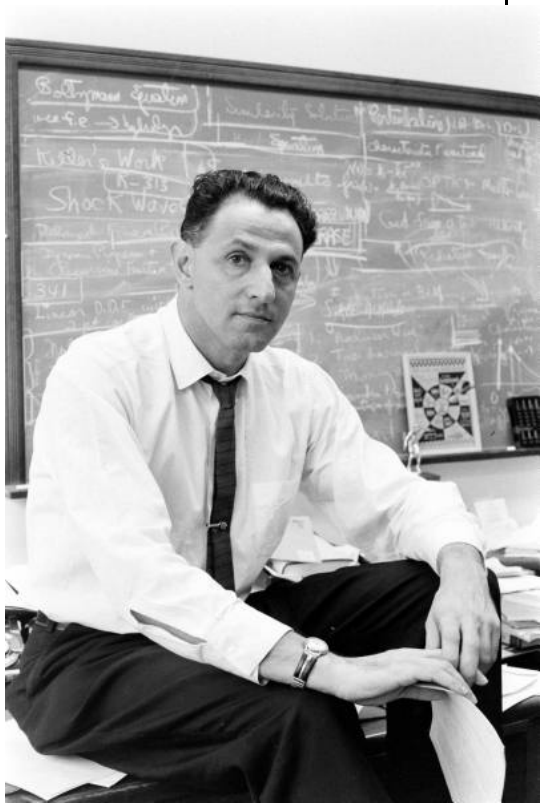
Ponder this (this month)

At the beginning, the wheel's arrow points between 1 and n. If the wheel reaches the number t, t is removed from the wheel, the size of the remaining numbers are readjusted and evenly spaced, and the arrow is moved to the point in between t's left and right neighbors. Each step moves the wheel a little less than one number forward (so the wheel comes to rest on a number and not between two). So, if the wheel is right before 1 and performs 3 steps, it ends up on 3.





- 动态规划
  - Fibonacci数列
  - 0/1背包问题
- 八皇后问题
- 随机程序



Richard Bellman
(1920—1984)

Fibonacci数列的计算问题

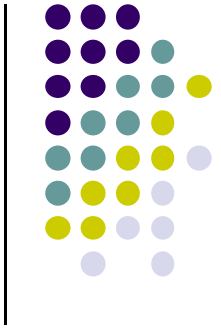
约1175-约1250

- 数列中每个数都是其前一个数和更前面一个数的和

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_n = F(n-1) + F(n-2),\ n \geq 2 \end{cases}$$

- 例如：0,1,1,2,3,5,8,13,21,34，…
- 随n增大，Fibonacci数列呈指数增长。 $F_n \approx 2^{0.694n}$
- 问题：给定任意一个n，F_n是多少？F_{100}？ F_{200}？
  - 计算机出现以前，人们只能够算出n不大的F_n

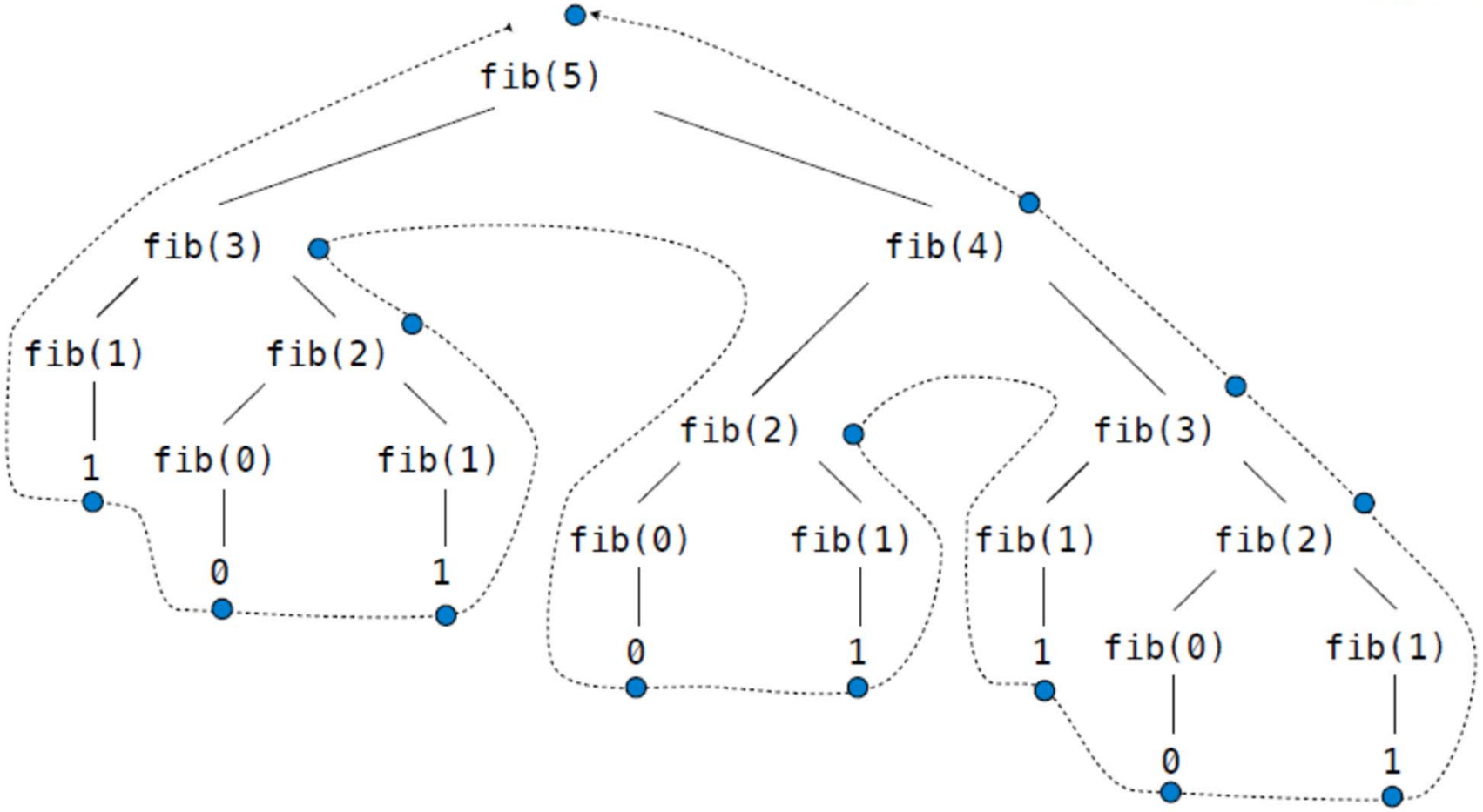
一个指数时间算法



- 算法如下

```
function fib1(n)
if n = 0:   return 0
if n = 1:   return 1
return fib1(n − 1) + fib1(n − 2)
```

- 考虑三个问题：
  - 1）正确性。正确，就是Fibonacci数列的定义
  - 2）执行时间。？？
  - 3）能否改进。？？

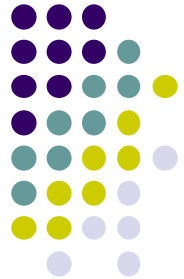

fib(5)
fib(3)
fib(4)
fib(1)
fib(2)
fib(2)
fib(3)
1
fib(0)
fib(1)
fib(0)
fib(1)
fib(1)
fib(2)
0
1
0
1
1
fib(0)
fib(1)
0
1

一个多项式时间算法

- 算法使用一个数列保存中间结果，避免重复计算
  - 当计算f[i]时，f[i-1]和f[i-2]已经获得。

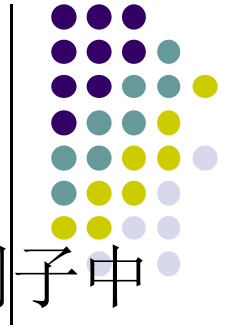
```
function fib2(n)
if n = 0 return 0
create an array f[0...n]
f[0] = 0, f[1] = 1
for i = 2...n:
    f[i] = f[i − 1] + f[i − 2]
return f[n]
```

背包问题



- 一个小偷携带一个背包入室盗窃。背包总共能够容纳W公斤的东西。小偷发现有n种物品可以盗窃，重量分别为$w_1,w_2,\ldots,w_n$，在市场上变卖分别价值$v_1,v_2,\ldots,v_n$。小偷希望能够在背包的容纳极限内，偷到尽可能值钱的物品。
  - 例如，W=10，物品的重量和价值分别如下

物品编号	重量	价值
1	6	￥30
2	3	￥14
3	4	￥16
4	2	￥9



- 背包问题有两个版本：
  - 版本一：每样物品只有一件，这种情况下上面例子中小偷应该取物品1和3，总价值￥46。
  - 版本二：每样物品有无穷多个，这种情况下，上面例子中实现最大盗窃价值的方法是取一个物品1和两个物品4，总价值￥48。

- 贪心方法可以得到一个近似解
- 枚举法可以求解最优解，时间是$O(n*2^n)$
- 使用动态规划，两个版本的背包问题都可以在O(nW)时间内求解。

课本第12章例子

	价值	重量	价值/重量
钟	175	10	17.5
油画	90	9	10
收音机	20	4	5
花瓶	50	2	25
书	10	1	10
电脑	200	20	10

贪心：快速，未必最优

枚举：最优，速度很慢

名称	值	重量
a	6	3
b	7	3
c	8	2
d	9	5

图13-3　带有价值和重量的物品表

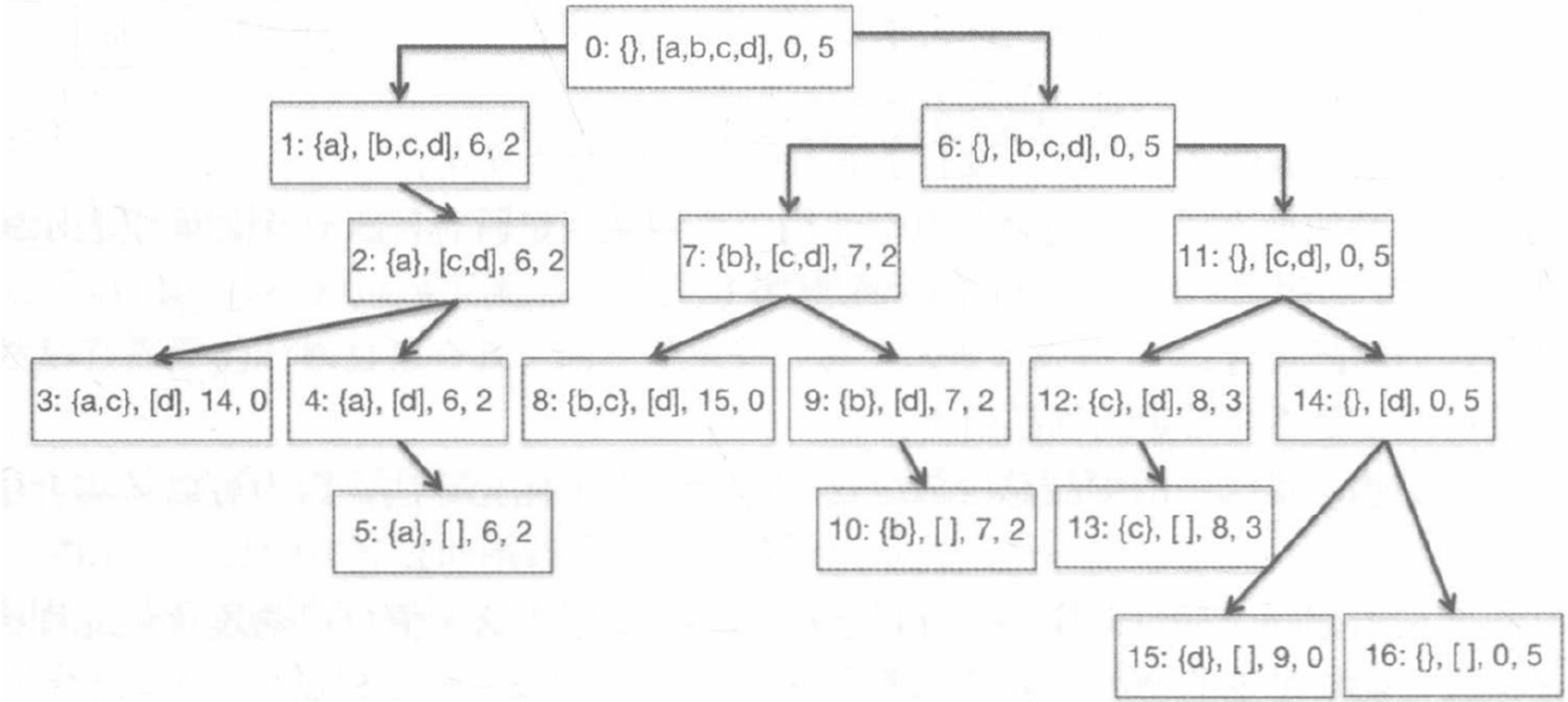
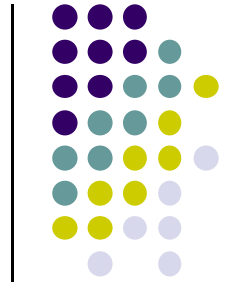


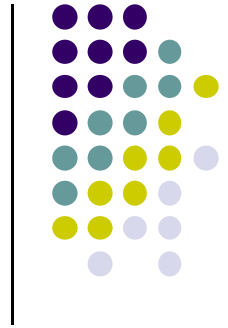
图13-4　背包问题的决策树



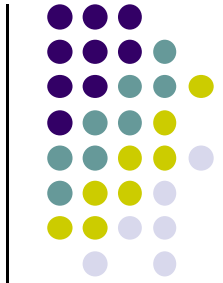
```
def smallTest():
    names = ['a', 'b', 'c', 'd']
    vals = [6, 7, 8, 9]
    weights = [3, 3, 2, 5]
    Items = []
    for i in range(len(vals)):
        Items.append(Item(names[i], vals[i], weights[i]))
    val, taken = maxVal(Items, 5)
    for item in taken:
        print(item)
    print('Total value of items taken =', val)

def buildManyItems(numItems, maxVal, maxWeight):
    items = []
    for i in range(numItems):
        items.append(Item(str(i),
                          random.randint(1, maxVal),
                          random.randint(1, maxWeight)))
    return items

def bigTest(numItems):
    items = buildManyItems(numItems, 10, 10)
    val, taken = maxVal(items, 40)
    print('Items Taken')
    for item in taken:
        print(item)
    print('Total value of items taken =', val)
```



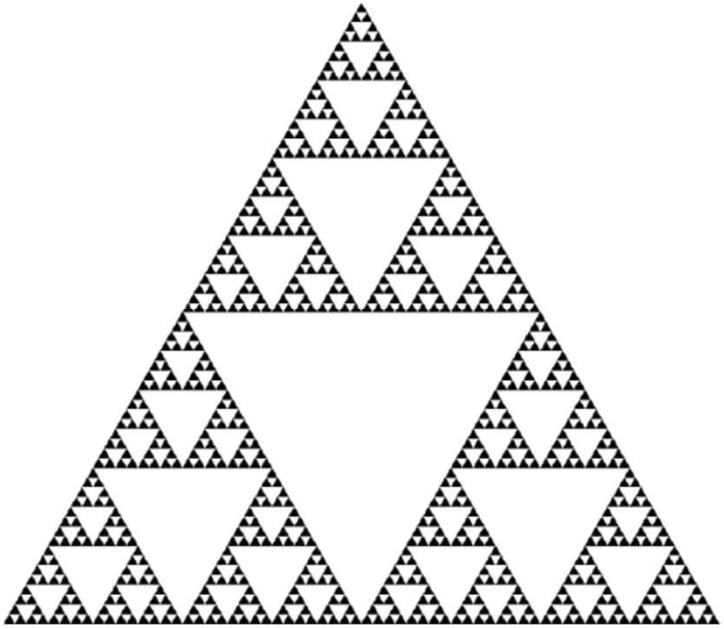
```
#Figure 13.7
def fastMaxVal(toConsider, avail, memo = {}):
    """Assumes toConsider a list of items, avail a weight
       memo supplied by recursive calls
       Returns a tuple of the total value of a solution to the
       0/1 knapsack problem and the items of that solution"""
    if (len(toConsider), avail) in memo:
        result = memo[(len(toConsider), avail)]
    elif toConsider == [] or avail == 0:
        result = (0, ())
    elif toConsider[0].getWeight() > avail:
        #Explore right branch only
        result = fastMaxVal(toConsider[1:], avail, memo)
    else:
        nextItem = toConsider[0]
        #Explore left branch
        withVal, withToTake =\
                fastMaxVal(toConsider[1:],
                           avail - nextItem.getWeight(), memo)
        withVal += nextItem.getValue()
        #Explore right branch
        withoutVal, withoutToTake = fastMaxVal(toConsider[1:],
                                               avail, memo)
        #Choose better branch
        if withVal > withoutVal:
            result = (withVal, withToTake + (nextItem,))
        else:
            result = (withoutVal, withoutToTake)
    memo[(len(toConsider), avail)] = result
    return result
```

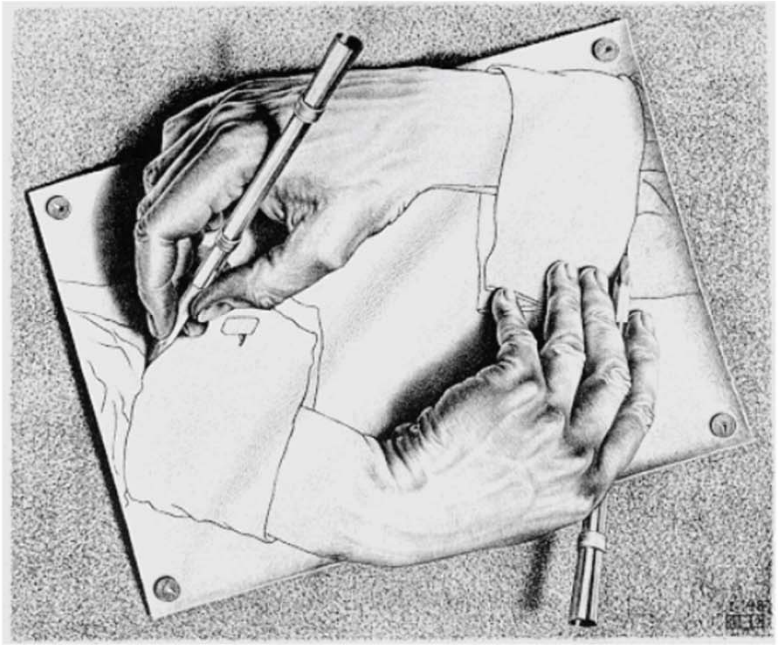


len(Items)	Number of items selected	Number of calls
4	4	31
8	6	337
16	9	1 493
32	12	3 650
64	19	8 707
128	27	18 306
256	40	36 675

图13-8　动态规划解法性能

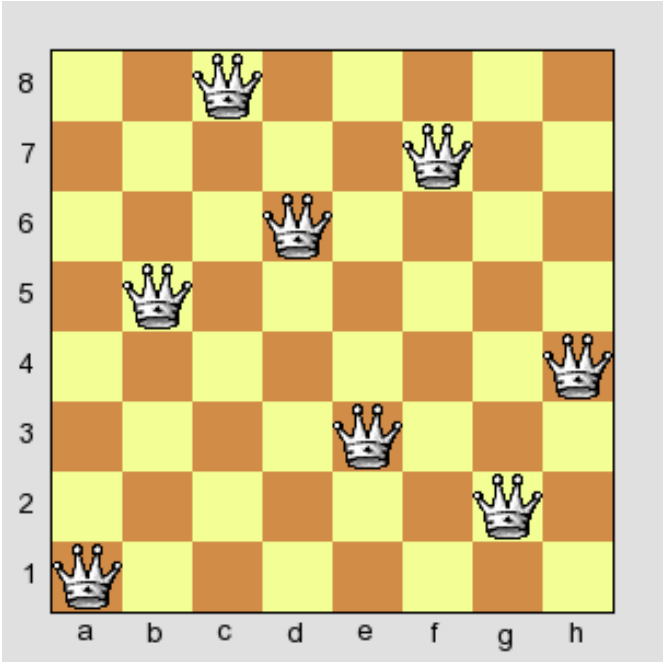
递归



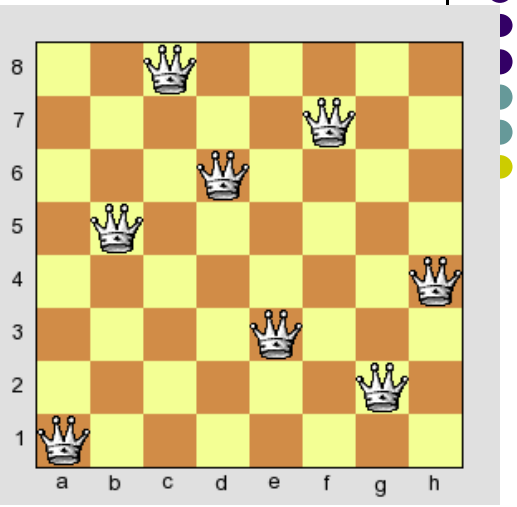


问题：8皇后问题

- 一个古老而著名的问题，是回溯算法的典型例题。
- 在8×8格的国际象棋上摆放八个皇后，使其不能互相攻击，即任意两个皇后不能处于同一行、同一列或同一斜线，问有多少种摆法。



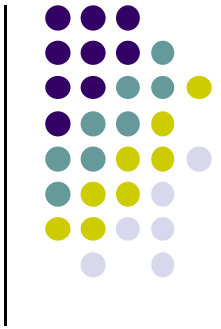
问题：8皇后问题



- 1850年，高斯认为有76种方案。

- 1854年在柏林的象棋杂志上不同作者发表了40种不同的解。
- 之后有人用图论的方法解出92种结果。

Carolus Fridericus Gauss
(1777—1855)

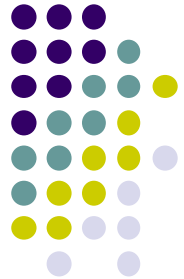


八皇后问题可用八重循环解决。N皇后问题呢?

递归能起到多重循环的作用，关键在于栈里保存了不同层次的循环控制变量 i 的值（对应于不同行），棋盘有几行，i 就有几个

循环 -> 递归

随机模拟

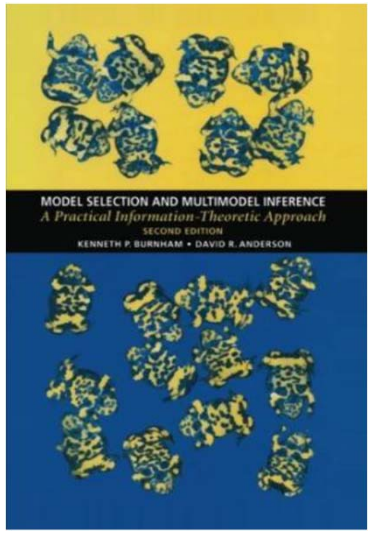


```
import random

def rollDie():
    """返回一个1~6的随机整数"""
    return random.choice([1,2,3,4,5,6])

def rollN(n):
    result = ''
    for i in range(n):
        result = result + str(rollDie())
    print(result)
```

模型



A model is a simplification or approximation of *reality and hence will not reflect all of reality.*

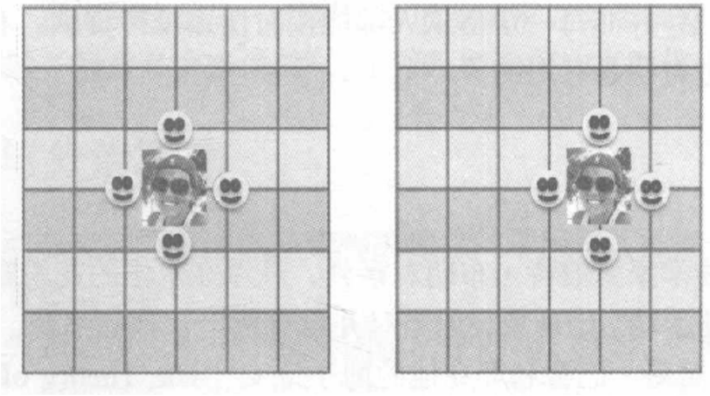
— *Model Selection and Multimodel Inference*

"All *models are wrong, but some are useful.*"

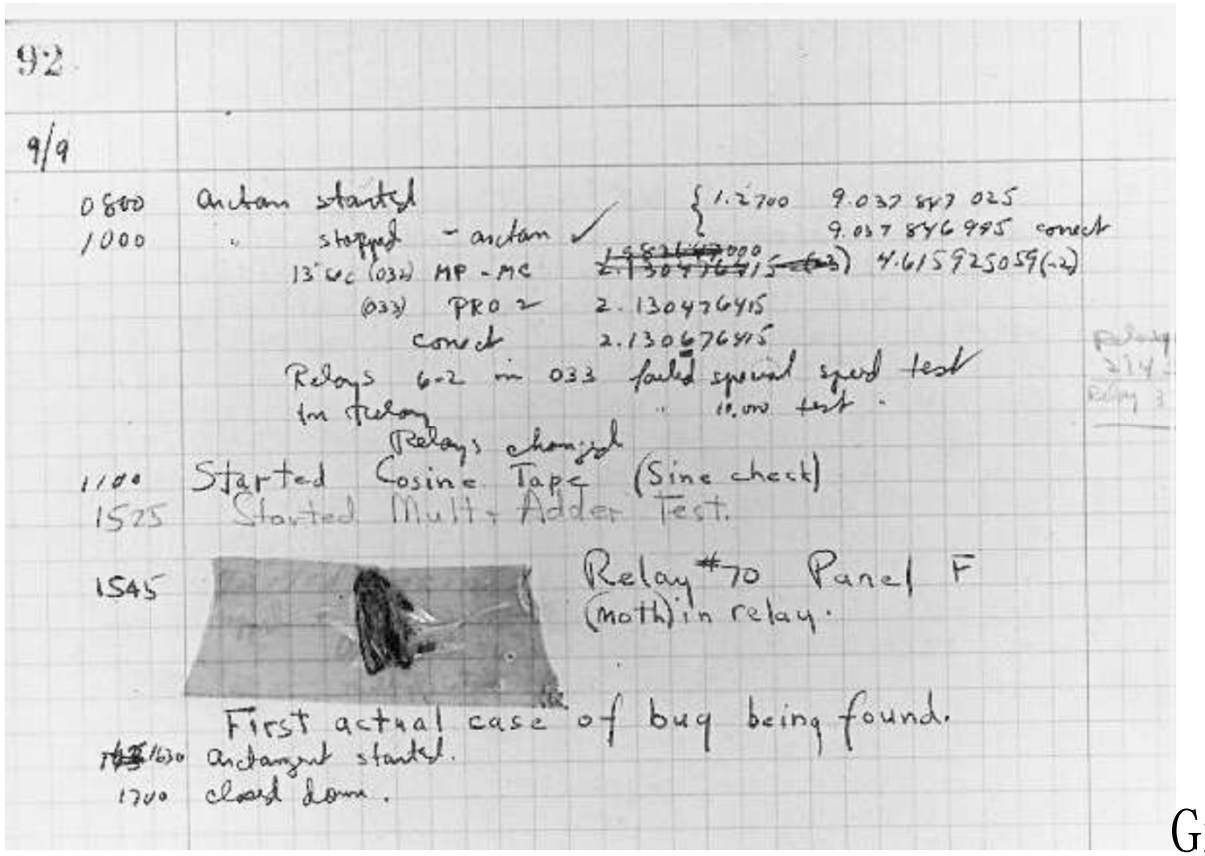
— *George Box*

随机游走

- 相关课程：概率论与数理统计、随机过程
- 求解问题：醉汉游走
- 从原点出发，每秒钟随机的向一个方向（上下左右）移动一步。1000秒后，
  - 与原点的期望距离是多少？
  - 会离原点越来越远，还是有可能一遍又一遍回到原点并始终在原点附近？



Bug!



92

9/9

0800 Arctan started
1000 " stopped - arctan ✓
13 wc (033) MP - MC
(033) PRO 2
correct
Relays 6-2 in 033 failed special speed test
In relay
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.
1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1700 closed down.

Grace Murray Hopper
（1906—1992）