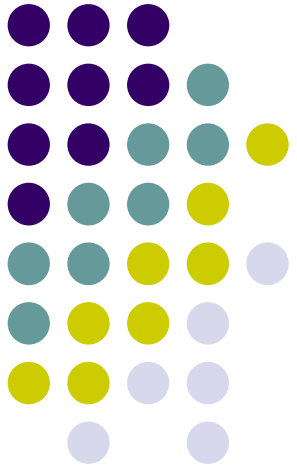
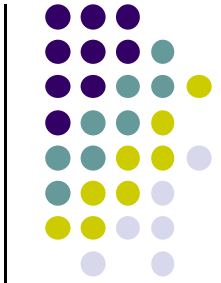
面向科学问题求解的编程实践

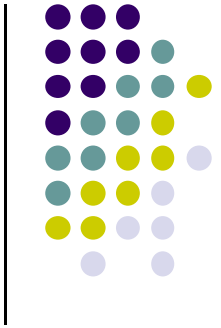


问题回答



- 上机时间
- 软件安装问题
  - 第11章的画图工具安装 -> 助教辅导
- 报告选题
  - 选题5月12日(周三）初步确定（BB系统提交）
  - 难度问题

最优化问题

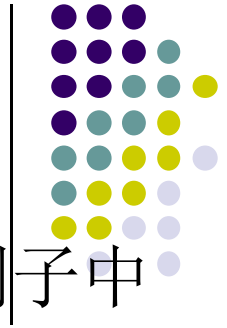


- 通常包括两个部分
  - 目标函数
  - 约束条件集合
- 主要内容
  - 现实问题可以抽象为最优化问题并计算求解
  - 新问题可以转换为已能求解的问题
  - 背包问题和图的最优化问题
  - 穷举比较简单，但可能不可行
  - 贪心方法很实用，虽然可能不一定是得到最优解

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

物品的类

```
class Item(object):
    def __init__(self, n, v, w):
        self.name = n
        self.value = v
        self.weight = w
    def getName(self):
        return self.name
    def getValue(self):
        return self.value
    def getWeight(self):
        return self.weight
    def __str__(self):
        result = '<' + self.name + ', ' + str(self.value)\
                 + ', ' + str(self.weight) + '>'
        return result
def value(item):
    return item.getValue()

def weightInverse(item):
    return 1.0/item.getWeight()

def density(item):
    return item.getValue()/item.getWeight()
```

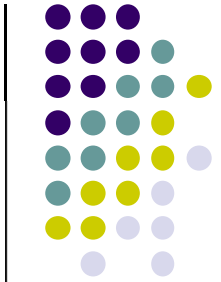
贪心算法的实现

```
def greedy(items, maxWeight, keyFunction):
    """假设Items是列表，maxWeight >= 0
       keyFunctions将物品元素映射为数值"""
    itemsCopy = sorted(items, key=keyFunction, reverse = True)
    result = []
    totalValue, totalWeight = 0.0, 0.0
    for i in range(len(itemsCopy)):
        if (totalWeight + itemsCopy[i].getWeight()) <= maxWeight:
            result.append(itemsCopy[i])
            totalWeight += itemsCopy[i].getWeight()
            totalValue += itemsCopy[i].getValue()
    return (result, totalValue)
```

```
def buildItems():
    names = ['clock','painting','radio','vase','book','computer']
    values = [175,90,20,50,10,200]
    weights = [10,9,4,2,1,20]
    Items = []
    for i in range(len(values)):
        Items.append(Item(names[i], values[i], weights[i]))
    return Items

def testGreedy(items, maxWeight, keyFunction):
    taken, val = greedy(items, maxWeight, keyFunction)
    print('Total value of items taken is', val)
    for item in taken:
        print(' ', item)

def testGreedys(maxWeight = 20):
    items = buildItems()
    print('Use greedy by value to fill knapsack of size', maxWeight)
    testGreedy(items, maxWeight, value)
    print('\nUse greedy by weight to fill knapsack of size',
          maxWeight)
    testGreedy(items, maxWeight, weightInverse)
    print('\nUse greedy by density to fill knapsack of size',
          maxWeight)
    testGreedy(items, maxWeight, density)
```



```
Use greedy by value to fill knapsack of size 20
Total value of items taken is 200
    <computer, 200, 20>

Use greedy by weight to fill knapsack of size 20
Total value of items taken is 170
    <book, 10, 1>
    <vase, 50, 2>
    <radio, 20, 4>
    <painting, 90, 9>

Use greedy by density to fill knapsack of size 20
Total value of items taken is 255
    <vase, 50, 2>
    <clock, 175, 10>
    <book, 10, 1>
    <radio, 20, 4>
```

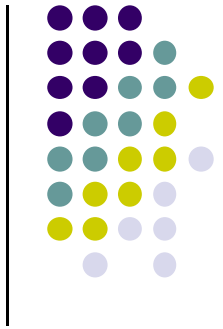
枚举法求最优解

```
def chooseBest(pset, maxWeight, getVal, getWeight):
    bestVal = 0.0
    bestSet = None
    for items in pset:
        itemsVal = 0.0
        itemsWeight = 0.0
        for item in items:
            itemsVal += getVal(item)
            itemsWeight += getWeight(item)
        if itemsWeight <= maxWeight and itemsVal > bestVal:
            bestVal = itemsVal
            bestSet = items
    return (bestSet, bestVal)

def testBest(maxWeight = 20):
    items = buildItems()
    pset = genPowerset(items)
    taken, val = chooseBest(pset, maxWeight, Item.getValue,
                            Item.getWeight)
    print('Total value of items taken is', val)
    for item in taken:
        print(item)
```

- 背包问题版本一：每样物品无穷多件
  - 动态规划核心：如何将问题分解为更小的子问题
  - 定义：K(w)=如果背包的容量为w，能够取得的价值
    - 原问题表达为求解K(W)
  - 如果背包问题的最优解包含至少一件物品i，则将一件物品i拿掉，剩下的物品必然构成问题$K(w-w_i)$的最优解。然而，我们并不知道最优解包含哪些物品，所以有如下关系

$$K(w) = \max_{i:w_i \le w}\{K(w-w_i)+v_i\}$$



- 从问题分解，得出算法如下

$K(0) = 0$
for $w = 1$ to W:
    $K(w) = \max\{K(w - w_i) + v_i : w_i \leq w\}$
return $K(W)$

- 算法分析：算法依次计算K(1)，K(2)，…，K(W)。每次计算K(w)，需要考虑所有n种物品，耗时O(n)，所以整个算法耗时O(nW)

- 背包问题版本二：每样物品仅一件
  - 原来的问题分解已经不适用了。比如，如果从问题K(w)的解中拿掉一件物品n，则在考虑$K(w-w_n)$时我们必须排除物品n，然而算法中解决$K(w-w_n)$时仍然考虑所有的物品。
  - 引入第二个变量j，$0 \le j \le n$
  - 定义：
    K(w,j)=背包容量为w，且仅考虑物品1,2,…,j时能够取得的最大价值。
    - 原问题表达为求解K(W,n)

- 考虑问题K(w,j)，如果问题的解中包含物品j，则拿掉它，剩下的物品构成问题K(w-w_j,j-1)的最优解；如果不包含物品j，则可以直接考虑K(w,j-1)。因此

$$K(w, j) = \max\{K(w - w_j, j-1) + v_j, K(w, j-1)\}$$

  - 注意：如果w_j>w，第一种情况直接不考虑
- 算法如下：

Initialize all $K(0, j) = 0$ and all $K(w, 0) = 0$
for $j = 1$ to n:
    for $w = 1$ to W:
        if $w_j > w$: $K(w, j) = K(w, j-1)$
        else: $K(w, j) = \max\{K(w, j-1), K(w - w_j, j-1) + v_j\}$
return $K(W, n)$

- 这个算法的执行过程类似于填写一个(W+1)×(n+1)的表格。在确定每个表格项K(w,j)时，需要O(1)时间。算法耗时O(nW)。

图的最优化问题

- 哥尼斯堡七桥问题（1735年）

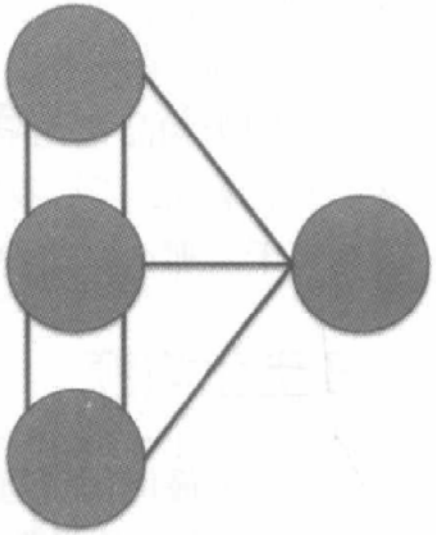
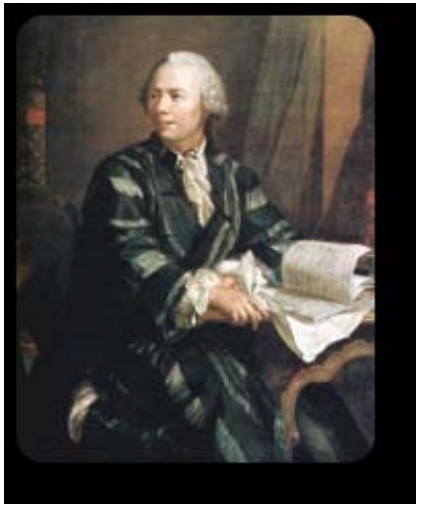


图12-6　哥尼斯堡的七座桥（左）及欧拉的简化地图（右）



Leonhard Euler
（1707—1783）

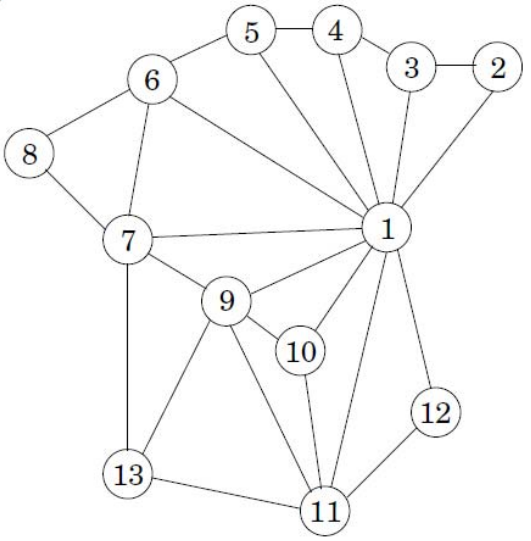
图的应用

- 很多问题都可以使用图来描述和刻画
  - 地图染色问题：在地图上将不同的国家涂上不同的颜色，使得相邻的国家被涂上不同的颜色
  - 期末考试安排：大学教务处安排每门课期末考试的时间，要求如果有同学选两门课，则这两门课不能同时进行考试

- 使用“图”来描述问题
  - 地图染色
    - 忽略不相干的信息，比如边界的形状
    - 将国家描述为图的节点，两个国家之间如果有边界，则相应节点之间有边连接。
    - 巴西-节点1
    - 阿根廷-节点11
  - 安排考试
    - 每门课是一个节点
    - 如果有同学选两门课，对应两个点有边连接

(a)

(b)



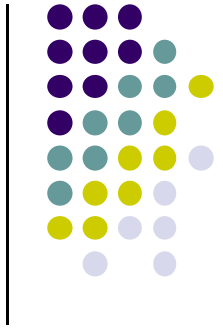
- 图的表示
  - 包含点的集合V和边的集合E
  - $V=\{v_1,v_2,\cdots\}$，如果$|V|=n$，图可以用一个邻接矩阵表示，

$$a_{ij}=\begin{cases}1 & \text{存在从}v_i\text{到}v_j\text{的边}\\ 0 & \text{不存在边}\end{cases}$$

  - 如果图的边有方向（称为有向图），则$a_{ij}\neq a_{ji}$；如果无方向（称为无向图），则$a_{ij}=a_{ji}$，矩阵是对称矩阵。
  - 在计算机中存储图，如果采用邻接矩阵，占用空间$O(n^2)$，这种方法在矩阵密度不高的情况下比较浪费空间
  - 仅仅存储边，比如在每个节点上存储和该节点连接的边，占用空间$O(|E|)$。称为邻接表

- 如何确定图的存储方式
  - 取决于|V|和|E|之间的关系。
  - 当|E|<|V|-1，图中将出现孤立的点
  - 边数最少的连通的图是树，有|E|=|V|-1
  - |E|最多为$|V|^2$-|V|
  - 当|E|接近|V|时，称图是稀疏的，此时采用邻接表的方式比较节省空间。
  - 例如，如果WWW上每个网页是一个节点，每个超链接是一个边，将会有超过80亿个节点，采用邻接矩阵存储需要几百万T(10^{12})的空间。如果采用邻接表（几百亿超链接），一两个T就够了。

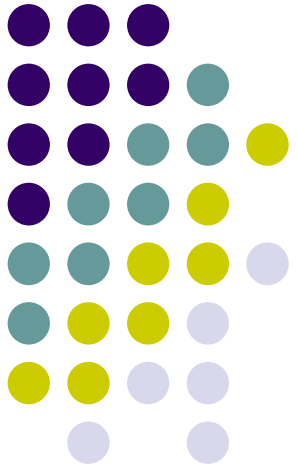
一些图论问题



- 最短路径
- 最短加权路径
- 最大团
- 最小割

Ponder this

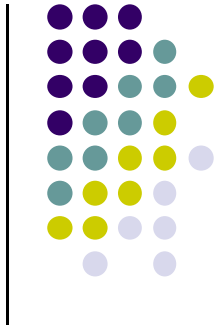
http://www.research.ibm.com/haifa/ponderthis/



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.

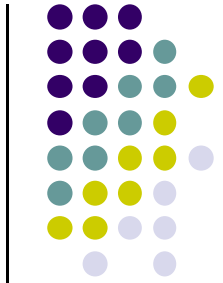
Solution



- 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
```

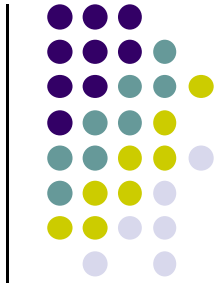
Solution



- 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.