Python程序设计

第三讲 程序设计基础

常用算法简介

张 华
WHU

常用算法简介

- 简单算法与复杂算法
- 枚举算法
- 迭代算法
- 递推算法
- 贪心算法

常用算法简介

- **算法设计水平的优劣，对于问题的求解非常重要。**
  - 实际工程中会遇到许多复杂的计算问题，有的问题若采用劣质算法求解，即便在巨型计算机上可能也要花费数个月时间，但采用优秀算法，可能在一台普通计算机上也仅需几分钟甚至数十秒就能解决。
- **可以将所有算法粗分为**
  - 简单的算法
  - 复杂的算法

简单算法

简单的算法

- ①一般有公式作为算法描述。
- ②算法通过一次性计算解决问题。
  - 例如工程学、力学、数学等学科中
    - $G=g*m1*m2/r^2$ (万有引力)
    - 电压=电阻*电流
    - 浮力=液体密度*物体排开液体的体积
    - 圆的面积=圆周率*半径*半径
    - 圆柱体体积=底面积*高
- 用“IPO”（输入-处理-输出）法描述。
  - 例如，温度转换就是一个简单算法。

复杂算法

复杂的算法

- ①一般没有终极公式作为问题解法。
- ②可通过一次次反复计算逐步得到问题的解或近似解。
- ③每个计算周期计算问题的一部分或得到一个更好的近似解。
- 用简单问题的解逐步逼近原始问题解，适合于用计算机求解。

复杂算法举例

求两个数的最大公约数

- 这是一个经典的程序设计例子，可用多种算法完成该任务。
- 方法一：直接用最大公约数的定义求解
  - 例如，求252和105的最大公约数
    - 252/105 =〉不能整除（从较小的数开始试算）
    - ？252/104 ？105/104 ×(不能整除)
    - ？252/103 ？105/103 ×(不能整除)
    - ？…… ×(不能整除)
    - ？ 252/21 ？105/21 ✓ =〉最大公约数为21

复杂算法举例

求两个数的最大公约数

方法二：“更相减损术”（来自《九章算术》）

- 两个整数的最大公约数等于其中较小的数和两数的差的最大公约数
  - 以较大的数减较小的数，接着把所得的差与较小的数比较，并以大数减小数。
  - 继续这个操作，直到所得的减数和差相等为止。
- 例如，求252和105的最大公约数
  - **252-105=147**
  - **147-105=42**
  - **105-42=63**
  - **63-42=21**
  - **42-21=21** =>最大公约数为21

复杂算法举例

求两个数的最大公约数

方法三：欧几里德算法，即“辗转相除”法

- 对所求的两个数p和q (求greatest common divisor)
  - gcd(p, q)=gcd(q, p % q)
  - 当p%q为0时，两数的最大公约数即为q
- 即
  - 将大数作为被除数，小数作为除数，若二者余数不为0，则将小数作为被除数，余数作为除数，…直到余数为0。
- 例如，求252和105的最大公约数
  - 252%105 =〉42
  - 105%42 =〉21
  - 42%21 =〉0 =>最大公约数为21

枚举算法

枚举算法

枚举算法

- 枚举法就是按问题本身的性质，一一列举出该问题所有可能的解，并在逐一列举的过程中，检验每个可能解是否真正满足问题所求。若是则采纳这个解，否则抛弃它。
- 例如，求1-1000中，能被3整除的数。
  - 算法：
    - （1）从1-1000中一一列举，这是一个循环结构
    - （2）在循环中对每个数进行检验：凡是能被3整除的数，打印输出，否则继续检验下一个数。

枚举算法

枚举算法的基本思想

- 根据问题描述和相关的知识，能为该问题确定一个大概的解空间范围。
- 枚举法就是对该解空间范围内的众多候选解按某种顺序进行逐一枚举和检验，直到找到一个或全部符合条件的解为止。

计算机程序实现枚举算法的基本方法

- 用循环结构实现一一列举的过程，
- 用选择结构实现检验的过程。

枚举算法

枚举算法的总体框架

- 枚举法一般使用循环结构来实现，其框架如下：

```
设解的个数n初始为0;
循环( 枚举每一可能解 ):
        若( 该解法满足约束 ):
                输出这个解;
                解的数量n加1;
```

枚举算法应用举例

- **问题：**
  - 有一组随机数，要找到这个数列中最大的数。
- **分析：**
  - 如果将数列中的每一个数字看成是一颗豆子的大小，可以使用下面的“捡豆子”算法：
    - 首先将第一颗豆子放入碟子中。
    - 取第二颗豆子开始检查：
      - 如果正在检查的豆子比碟子中的大，则将它捡起放入碟子中，同时丢掉原先碟子中的豆子。
      - 反之则舍去该豆子。
    - 继续取下一颗豆子，重复上一步过程，直到最后一颗豆子。
    - 最后留在碟子中的豆子就是所有的豆子中最大的一颗。

枚举算法应用举例

- 问题：判断谁是小偷
  - 有四个嫌疑人：
    - a说："我不是小偷。"
    - b说："c是小偷。"
    - c说："小偷肯定是d。"
    - d说："c冤枉人！"
  - 四人中有三人说的是真话，问到底谁是小偷？
- 分析
  - 依次假设a、b、c、d为小偷进行判断

枚举算法应用举例

求解“谁是小偷？”的算法

- 依次假设a、b、c、d为小偷，判断四人说话的真假。若在某假设下，得到的结果为三人说真话、一人说假话，该假设成立，此假设就是所求的解。
- 例如：
  - 假设a是小偷，则根据以上四人的回答可判断a、b、c都说的假话，所以这种假设不成立；
  - 同样，假设b是小偷也不成立；
  - 假设c是小偷，则根据四人的回答可判断a、b和d都说了真话，只有c一人说假话，所以假设成立。

枚举算法应用举例

求解“谁是小偷？”的算法

- 对a、b、c、d依次进行判断，显然这是一个循环的过程。为了能进行循环处理，需要对问题进行数字化：
  - 设x为假设的小偷，x依次取值1、2、3、4（表示a、b、c、d）；
  - 用s1，s2，s3，s4表示在某个假设下（即x分别为1、2、3、4时）四人说话的状态（说真话为1，说假话为0）。

枚举算法应用举例

求解“谁是小偷？”的算法

- 据此可得到以下算法：
  - 初始：设s1=s2=s3=s4=0 （“=”表示赋值）
  - 循环：对x=1,…,4 （假设4个可能的解）
    - 在此假设下，分别求四人的说话状态（s1~s4）
      - 若x!=1，s1=1（a说：“我不是小偷”。真话）；
      - 若x==3，s2=1（b说：“c是小偷”。真话）；
      - 若x==4，s3=1（c说：“小偷肯定是d。”真话）；
      - 若x!=4，s4=1（d说：“c冤枉人！”真话）。
    - 若经以上判断后，有s1+s2+s3+s4==3（即三人说真话，一人说假话），则假设成立，小偷为此时x的值对应的那个人。

枚举算法

- **应用枚举法通常有以下几个步骤：**
  - (1)建立问题的数学模型，确定问题的可能解的集合（可能的解空间）。
  - (2)确定合理的筛选条件，用来选出问题的解。
  - (3)确定搜索策略，逐一枚举可能解集合中的元素，验证是否是问题的解。

枚举算法应用举例

- **问题1：**
  - 将苹果、桔子、香蕉、梨、菠萝5种水果做水果拼盘，每个水果拼盘有3种不同的水果，且有序摆放。问可以有多少种？
- **分析与设计：**
  - 使用列表保存5种水果名。
  - 通过三重循环结构，枚举各种可能的方案
    - x、y、z的取值范围为5种水果（解空间）
    - 它们互不相等（筛选条件）
    - 摆放先后次序有区别
  - 输出所有可能的方案。

枚举算法应用举例

问题2：

- 百钱买百鸡问题：有一个人有一百块钱，打算买一百只鸡。到市场一看，大鸡三块钱一只，小鸡一块钱三只，不大不小的鸡两块钱一只。现在，请你编一程序，帮他计划一下，怎么样买法，才能刚好用一百块钱买一百只鸡？

分析与设计：

- 此题很显然是用枚举法，以三种鸡的个数为枚举对象（分别设为x，y，z），以三种鸡的总数（x+y+z）和买鸡用去的钱的总数(x*3+y*2+z/3)为判定条件，穷举各种鸡的个数。

迭代算法

迭代算法

迭代

- 是一种不断用变量的旧值递推新值的过程。
- 迭代算法是计算机解决问题的一种基本方法。
  - 它利用计算机运算速度快、适合做重复性操作的特点，
  - 从一个初始变量值出发，让计算机对一组指令（或一组步骤）进行重复执行，
  - 每次执行这组指令（或步骤）时，都从变量的原值推出它的一个新值。
  - 最终得到所求解。

迭代算法

一个最简单的迭代例子：

- 考虑求 1+2+3+……，直到和达到1000时的自然数问题。
  - 可以设一个累加变量s（初始s=0），然后通过循环将自然数1、2、3……依次加到该累加变量s中，直到s≥1000为止，这时最后一个加入的自然数就是所求之数。
- 这个例子就体现了迭代的思想，累加变量s就是迭代变量。

迭代算法

应用迭代算法的通常步骤：

(1)确定迭代变量

- 在可以用迭代算法解决的问题中，至少存在一个直接或间接地不断由旧值递推出新值的变量，这个变量就是迭代变量。
- 一般在确定迭代变量的同时，还要指定其初始值。

(2)建立迭代关系式

- 所谓迭代关系式，是指如何从变量的前一个值推出其下一个值的公式（或关系）。
- 迭代关系式的建立是解决迭代问题的关键，通常可以顺推或倒推的方法来完成。

迭代算法

应用迭代算法的通常步骤：

(3)对迭代过程进行控制

- 什么时候结束迭代过程？
  - 这是编写迭代程序必须考虑的问题。
  - 根据算法“有限性”的性质，不能让迭代过程无休止地重复执行下去。
- 迭代过程的控制通常有两种情况：
  - **a.** 所需的迭代次数是个确定的值，可以计算出来。
    - » 可以构建一个固定次数的循环来实现对迭代过程的控制。（常用**for**语句）
  - **b.** 所需的迭代次数开始时无法确定。
    - » 需要进一步分析出用来结束迭代过程的条件。
    - » 例如，当计算精度满足要求后结束。（常用**while**语句）

迭代算法应用举例

问题：验证谷角猜想

- 日本数学家谷角静夫在研究自然数时发现了一个现象：
  - 对于任意一个自然数 n，
    - 若 n 为偶数，则将其除以 2；
    - 若 n 为奇数，则将其乘以 3，然后再加 1。
  - 如此经过有限次运算后，总可以得到自然数 1。
- 要求：由键盘输入一个自然数 n，把 n 经过有限次运算后，最终变成自然数 1 的全过程打印出来。

迭代算法应用举例

分析

- 定义迭代变量为 n ， 按照谷角猜想的内容，可以得到两种情况下的迭代关系式：
  - 当 n 为偶数时， n=n/2 ；
  - 当 n 为奇数时， n=n*3+1 。
  - 这就是需要计算机重复执行的迭代过程。
- 确定结束迭代过程的条件：
  - 这个迭代过程需要重复执行多少次，才能使迭代变量 n 最终变成自然数 1 ？
  - 分析题目要求不难看出，对任意给定的一个自然数 n ， 只要经过有限次运算后，能够得到自然数 1 ，就完成了验证工作。
  - 因此，用来结束迭代过程的条件为： n==1 。

递推算法

递推算法

递推

- 与迭代相近的概念。
- 设要求问题规模为n的解，当n=1时，解已知或能方便得到，根据递推关系，能从i-1规模的解，推出i规模的解（i=2、3、……、n），这就是递推。
- 当然也可以反过来，从n、n-1、……、3、2反推到1。

递推算法应用举例

案例：植树问题

- 植树节那天，有五位同学参加了植树活动，他们完成植树的棵树都不相同。
  - 问第一位同学植了多少棵时，他指着旁边的第二位同学说比他多植了两棵；
  - 追问第二位同学，他又说比第三位同学多植了两棵；
  - ...
  - 如此，都说比另一位同学多植两棵。
  - 最后问到第五位同学时，他说自己植了10棵。
- 到底第一位同学植了多少棵树？

递推算法应用举例

植树问题分析

- 设第i位同学植树的棵树为ai，欲求a1，需从第五位同学植树的棵数a5入手，根据“多两棵”这个规律，按照一定顺序逐步进行推算：
  - (1) a5=10;
  - (2) a4=a5+2=12;
  - (3) a3=a4+2=14;
  - (4) a2=a3+2=16;
  - (5) a1=a2+2=18;
- 这就是递推的思想。

递推算法

- **递推方法是数学解题中的一种常用方法。**
  - 对一个序列来说，如果已知它的通项公式，则要求数列的第n项或前n项之和是很容易的。
  - 但许多情况下，要得到数列的通项公式很困难，然而可观察到数列相邻位置上的数据之间存在一定的关系。
  - 使用递推方法可以避开求通项公式的困难，在一次次循环中，通过已知的项推算出其后继值，直到最终找到所需的那一项。

递推算法

递推与迭代

- 递推的过程实际上就是迭代的过程，即不断用变量的旧值推出新值的过程。
- 一般递推使用数组（列表），在循环处理时利用其下标的变化实现变量的迭代，而狭义的迭代是指使用简单变量来完成这一过程。

递推算法

实施递推的步骤

- 确定递推变量
  - 递推变量可以是简单变量，也可是一维或多维组合对象（数组）。
- 建立递推关系
  - 从变量的前一些值推出其下一个值的公式或关系。这是解决递推问题的关键。
- 确定初始（边界）条件
  - 根据问题最简单的情景确定其初始数据
- 对递推过程进行控制

递推算法应用举例

问题：杨辉三角形

特点：

- 第k行有k个数；
- 每行的首尾两数均为1；
- 除首尾两数外，其余各数均为上一行肩上两数之和。

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
```

5行杨辉三角形

贪心算法

贪心算法

贪心算法（又称贪婪算法）

- 是指在问题求解时总是作出当前看是最好的选择。
- 也就是说，不从整体最优上加以考虑，它所做出的仅是在某种意义上的局部最优解。
- 贪心算法不是对所有问题都能得到整体最优解，但对范围相当广泛的许多问题它能产生整体最优解或者是整体最优解的近似解。

贪心算法

贪心算法的基本思想

- 在买东西时，售货员常常计算最少需要找多少张零钱，以便简化工作流程。
- 比如买东西需要48.5元，交给售货员100元整，按照现在的货币体系，则售货员最少需要找三张零钞：
  - 50元一张
  - 1元一张
  - 5角一张

贪心算法

贪心算法的求解策略

- 1）建立数学模型来描述问题。
- 2）把求解的问题分成若干个子问题。
- 3）对每一子问题求解，得到子问题的局部最优解。
- 4）把子问题的局部最优解合成为原来解问题的一个解。

贪心算法应用举例

问题1：最优装载

- 有n个集装箱希望能装上一艘载重量为C的轮船，其中集装箱i的重量为Wi。由于载重量的限制，这些集装箱不能全部装船。在装载体积不受限制的情况下，问最多能装载多少个集装箱。
- 载重受限，希望个数最多
- 最优装载问题可用贪心算法求最优解。
  - 其贪心选择策略是：重量轻者优先装载

贪心算法应用举例

问题2：活动安排

- 设有n个活动的集合S={1,2,…,n}，其中每个活动都要求使用同一资源，如演讲会场等，但同一时间内只有一个活动能使用这一资源。
- 怎样安排才可以举行尽可能多的活动？
- 是区间调度问题，可用贪心算法有效求解
- 先直观分析：
  - 先到先服务？
  - 短活动优先？
  - 早结束的活动优先！

贪心算法应用举例

活动安排问题分析

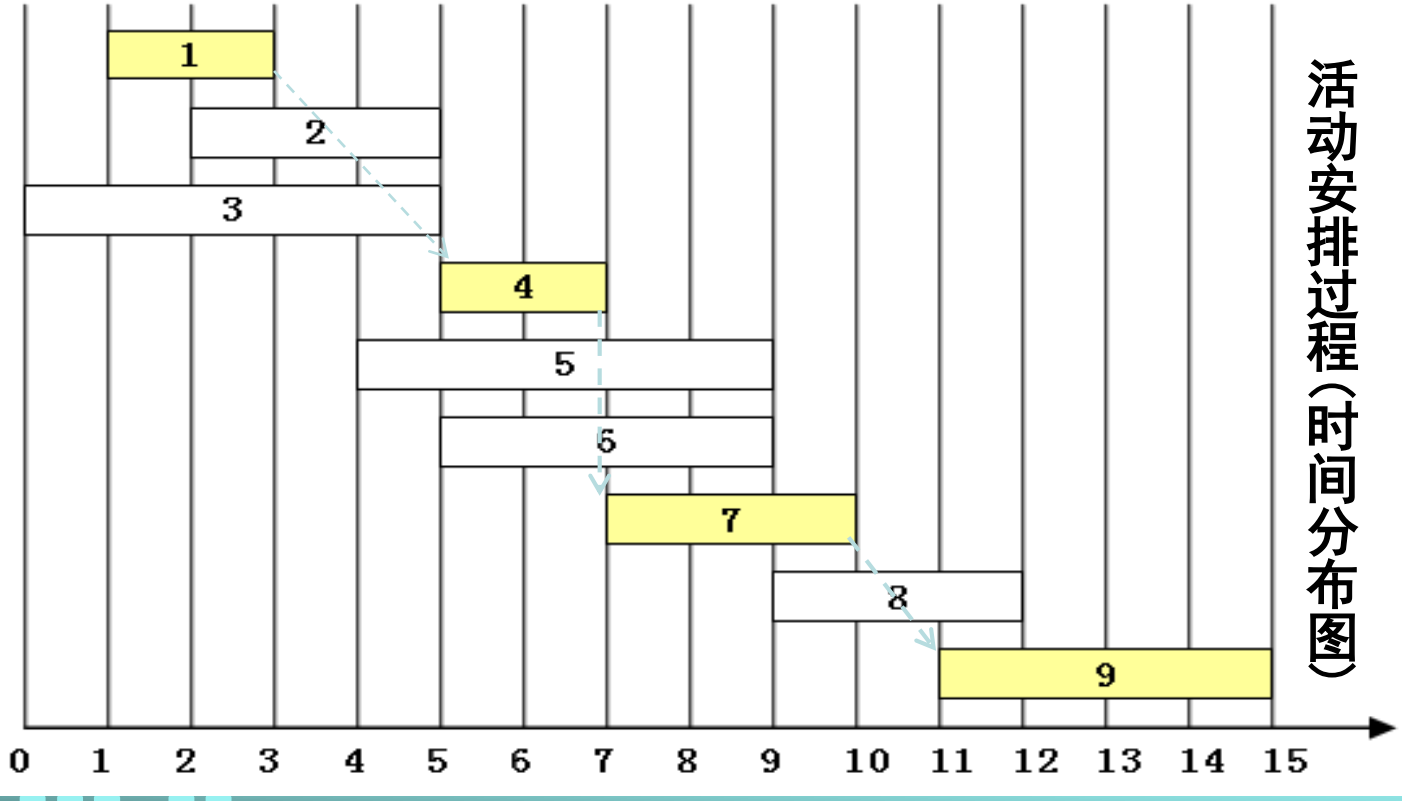
- 每个活动都有使用的起始时间bi和结束时间ei (显然bi≤ei)。对于活动i和j，若bi≥ej或bj≥ei(即一个活动结束后另一个才开始)，则活动i与活动j相容。
- 问题即为：求相容的最大活动集合。
- 示例数据（按结束时间排序）

i	1	2	3	4	5	6	7	8	9
B[i]	1	2	0	5	4	5	7	9	11
E[i]	3	5	5	7	9	9	10	12	15

贪心算法应用举例

活动安排问题分析

i	1	2	3	4	5	6	7	8	9
B[i]	1	2	0	5	4	5	7	9	11
E[i]	3	5	5	7	9	9	10	12	15



贪心算法应用举例

- **活动安排问题算法（伪代码形式）：**
  - 对所有活动按结束时间的先后排序(早结束者在前)。
  - 记录第1个活动，并使j=1。
  - 循环(i从2到n):
    - 若第i个活动的开始时间 ≥ 第j个活动的结束时间：
      - 记录该第i个活动
      - j=i
  - 输出所有记录的活动。

贪心算法

贪心算法的最优解问题

- 贪心算法要想找到最优解，需要两个条件:
  - 第一，原问题具有最优子结构。这样才能得到局部最优解。
  - 第二，原问题的最优解包含了它子问题的最优解。
- 这样才能由局部最优合成全局最优，并且局部最优解一旦获得就不会改变。

贪心算法

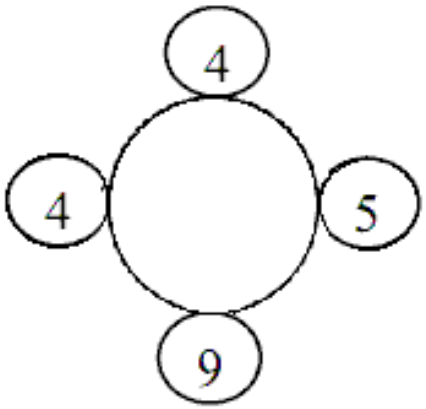
贪心算法的最优解问题

- 有时候，选择局部最优解并不能得到全局最优解。
- 例如，在前面售货员找钱的例子中，如果现有货币体系中还有8元面值、且不存在2元面值的话，则应该找零给顾客（65.5元）50元一张、8元一张、5元一张、1元二张和5角一张，共有六张零钞。但若给50元一张、5元三张和5角一张，显然只需五张零钞。

贪心算法

讨论：石子合并问题

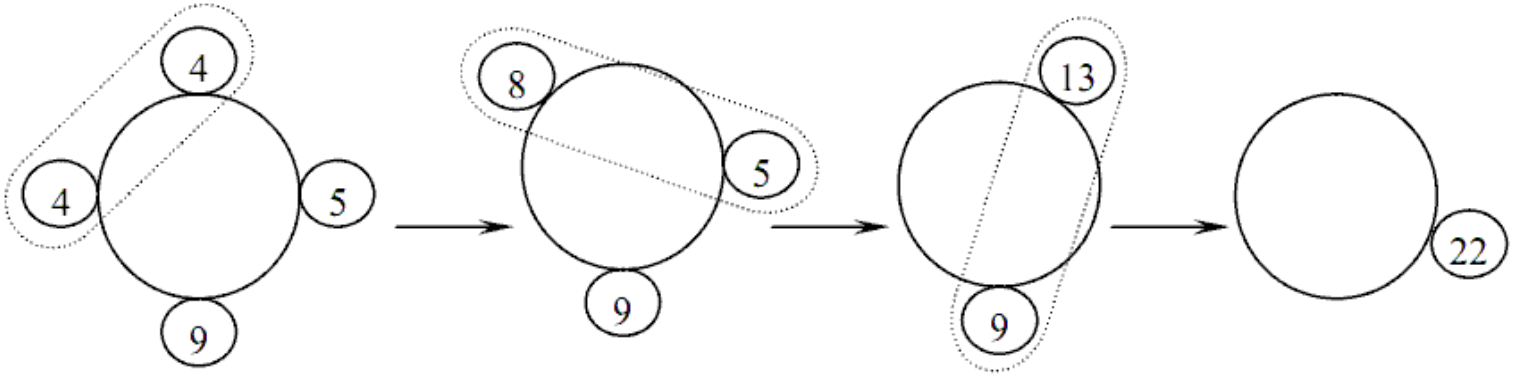
- 在一个操场的四周摆放 N 堆石子，现要将石子有次序地合并成一堆。规定每次只能选相邻的两堆合并成新的一堆，并将新的一堆的石子数记为该次合并的得分。已知每堆石子数量，请选择一种合并石子的方案，使得进行N-1次合并得分的总和最小。
- 示例数据:



贪心算法

讨论：石子合并问题

- 对于这个问题，很人都会选择贪心算法求解——即每次选相邻和最小的两堆石子合并。对于下面的图例，利用此贪心策略的确可以找到最优解。

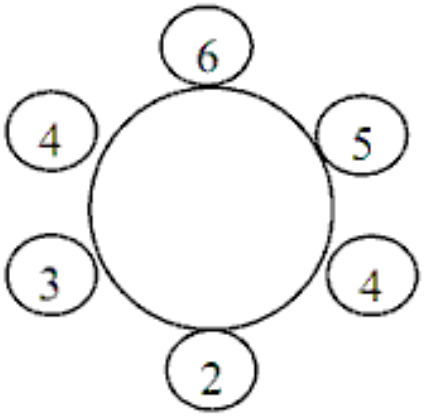


- 然而本题给的样例数据实际上是一个“陷阱”，造成了用贪心法即可解决的假象。

贪心算法

讨论：石子合并问题

- 反例



- 用贪心算法的最小值为：
  - 2＋3=5, 4+5=9, 4+5=9, 9+6=15, 15+9=24
  - 于是得分为 5＋9＋9＋15＋24＝62
- 另一种方法的最小值为：
  - 2＋4=6, 3+4=7, 5+6=11, 7+6=13, 11＋13=24
  - 于是得分为6＋7＋11＋13+24＝61

贪心算法

讨论：石子合并问题

- 可见此问题用贪心算法得到的不是最优解，贪心算法在子过程中得出的解只是局部最优，而不能保证使得全局的值最优。
- 需要用其他算法（如动态规划）来解决。