



贪心算法（贪心策略）



贪心算法(Greedy Algorithm)

- 在对问题求解时，总是做出在**当前看来是最好的**选择；
- **不从整体上考虑**最优，得到的仅是某种意义上的局部最优解；
- **用局部解构造全局解**，即从问题的某一个初始解逐步逼近给定目标，以尽可能快的方式或得更好的解。



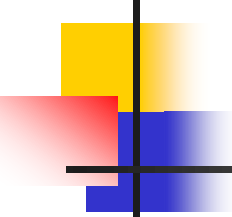
示例一求最大值

- 求一个数组的最大值。
 - 策略：
 i 从1到 n 循环，每次选择较大的（局部最优），最后
 $i==n$ 时找到的便是数组的最大值。
 - 结果
全局最优



示例—排队问题

- 在办事窗口**N**个人排队等候，每个人需要花费的时间分别为 **t_1, t_2, \dots, t_n** ，如何安排等待顺序使总的等待时间最短？
 - **策略：**
每次把时间消耗最短的人拿出来排前面
局部最优
 - **结果：**
总时间最短
全局最优



示例—选物品

- 给出 n 个物体，第 i 个重量为 w_i ，选择尽量多的物体，使得总重量不超过 C 。
 - 策略：
由于只关心物体的数量，选重的没有选轻的划算，只需要把物体重量从小到大排序，依次选轻的直到装不下为止。（每次选最轻的，局部最优）
 - 结果
全局最优



示例一钱币找零问题

- 指定币值种类以及相应的数量，用最少的数量凑齐某金额
 - **策略：**
利用贪心算法，优先选择面值大的钱币，以此类推，直到凑齐总金额。



贪心算法的一般步骤

1. 建立问题的**数学描述模型**；
2. 把问题分成若干**个子问题**；
3. 对每个子问题求解，得到**子问题的局部最优解**；
(贪心策略)
4. 把子问题的解(局部最优解)**合成**原问题的解。



示例—国王游戏问题

- 国王邀请 **n** 位大臣玩一个有奖游戏(**NOIP 2012**)
 - 国王让每个大臣在左、右手上面分别写下一个正整数，国王自己也在左、右手上各写一个正整数。然后，让这 **n** 位大臣排成一排，国王站在队伍的最前面。
 - 排好队后，所有的大臣都会获得国王奖赏的若干金币，每位大臣获得的金币数分别是：排在该大臣前面的所有人的左手上的数的乘积除以他自己右手上的数，然后向下取整得到的结果。
 - 国王不希望某一个大臣获得特别多的奖赏，所以他想请你帮他重新安排一下队伍的顺序，使得获得奖赏最多的大臣，所获奖赏尽可能的少。
 - 注意，国王的位置始终在队伍的最前面。



■ 贪心策略分析

- 考虑相邻的两个大臣，能不能调换两人的位置使得结果更优？（调整排队实际就是交换顺序）
- 设第 i 个大臣手中拿着 a_i 、 b_i ，第 $i+1$ 个大臣手中拿着 $a(i+1)$ 、 $b(i+1)$ ，设前 $i-1$ 个大臣的左手乘积为 S ；
- 交换前， i 和 $i+1$ 的奖赏分别为：
 S/b_i 、 $S*a_i/b(i+1)$
- 交换后， i 和 $i+1$ 的奖赏分别为：
 $S/b(i+1)$ 、 $S*a(i+1)/b(i)$
- 期望
 $\max(S/b(i+1), S*a(i+1)/b_i)$
 $\leq \max(S/b_i, S*a_i/b(i+1))$

- 
- 期望:

$$\max(\mathbf{S/b(i+1)}, \mathbf{S*a(i+1)/b_i}) \\ \leq \max(\mathbf{S/b_i}, \mathbf{S*ai/b(i+1)})$$

- 实际:

$$\mathbf{S/b(i+1)} \leq \mathbf{S*ai/b(i+1)}$$

如果:

$$\mathbf{S/b_i} \leq \mathbf{S*ai/b(i+1)} \text{ 且}$$

$$\mathbf{S*a(i+1)/b_i} \leq \mathbf{S*ai/b(i+1)}$$

则期望成立, 此时:

$$\mathbf{a(i+1)*b(i+1)} \leq \mathbf{ai*bi}$$



■ 贪心策略

- 如果 $a(i+1)*b(i+1) \leq a_i*b_i$ ，则交换第 i 和 $i+1$ 大臣的位置（大的排后面）
（子问题：局部最优）
- 逐一检查所有相邻的位置
结果：
所有大臣按照 a_i*b_i 从小到大进行排序
（全局最优）
- 对于任意一个不是从小到大的序列，总可以交换其中相邻的两个元素使结果更优。



贪心算法的特点

- 以当前情况为基础，根据某个优化测度作最优选择，不考虑各种可能的整体情况，省去了为找最优解要穷尽所有可能而必须耗费的大量时间。
- 自顶向下，以迭代的方式做出相继的贪心选择，每做一次贪心选择，就将所求问题简化为一个规模更小的子问题。
- 通过每一步贪心选择，可得到原问题的一个最优解。
- 虽然每一步都保证获得局部最优解，但由此产生的全局解有时不一定是最优的，因为贪心算法总是从局部出发，并没从整体考虑。



示例一背包问题

- **N**个物品，包的容量为 **C**，每个物品体积为 **W_i**，价值为 **V_i**，问背包内所装物品的最大价值？
- 贪心策略：
每次选
 - 1. **V_i** 最大的
 - 2. **V_i/W_i** 单位体积价值最高的
 - 3. **W_i** 最小的
- 能否得到最终答案？（全局最优解）

- 
- 背包容量 **C=9**，四个物品体积和价值分别为
 - **w1=6, v1=5, 5/6**
 - **w2=5, v2=4, 4/5**
 - **w3=4, v3=3, 3/4**
 - **w4=3, v4=1, 1/3**
 - 三种策略
 - 策略1：选 **1和4**，价值总和为 **6**
 - 策略2：选 **1和4**，价值总和为 **6**
 - 策略3：选 **4和3**，价值总和为 **4**
 - 实际上，选**2和3**更优，价值总和为**7**
贪心策略在此不适用



贪心算法的应用

- 针对具体问题，提出贪心策略
- 判断贪心策略是否正确
(由局部最优达到全局最优)
 - 正确，实施
 - 不正确，换
- 最优子结构性质
 - 当一个问题最优解，包含其子问题的最优解时，称此问题具有最优子结构性质。
 - 问题能够分解成子问题来解决，子问题的最优解能递推到最终问题的最优解。
 - 做贪心选择后，原问题简化为规模更小的类似子问题。



小结

- 贪心算法总是做出在当前看来是最好的选择；
- 贪心算法不是对所有问题都能得到全局最优解；
- 有很多基于贪心策略的经典算法
 - 霍夫曼编码
 - 最小生成树算法
 - 单源最短路径算法
 -