第三章贪心算法

3.1贪心算法的设计思想

- 贪心法通常用来解决具有最大或最小值的优化问题。
- 从某一个初始状态出发，根据当前局部的而不是全局的最优决策，以满足约束方程为条件，以使得目标函数的值增加最快或最慢为准则，选择一个能够最快达到要求的输入元素，以便尽快地构成问题的可行解。

- 贪心算法通过一系列的局部选择来得到一个问题的解。所作的每一个选择都是当前状态下“最优”的选择。
  - 要依照某种策略。策略“只顾眼前，不管将来”，称之为“贪心策略”。
- 贪心算法没有固定的算法框架，算法设计的关键是贪心策略的选择。
- 贪心算法能否得到最优解？
- 如何判断“当前最优”？

问题分析

- 例3.1 币种统计问题
- 某单位给每个职工发工资(精确到元)。为了保证不要临时兑换零钱，且取款的张数最少，取工资前要统计出所有职工的工资所需各种币值(100,50,20,10,5,2,1元共七种)的张数。请编程完成。

	GZ\币值	100	50	20	10	5	2	1
周	252	2	1				1	
张	2686	26	1	1	1	1		1
	……							
总计	2938	28	2	1	1	1	1	1

- 数据结构设计
  - 将七种币值存储在数组B。这样，七种币值就可表示为B[i],i=1,2,3,4,5,6,7。为了能实现贪心策略，七种币应该从大面额的币种到小面额的币种依次存储。
  - 记录最终币值所需数量：设置一个有7个元素的累加器数组S。
- 贪心策略：
  - 对每个人的工资，用“贪心”的思想，先尽量多地取大面额的币种，由大面额到小面额币种逐渐统计。

例3.1-不同情况

某国的币种是这样的,共9种:100,70,50,20,10,7,5,2,1。

这种情况下，刚才的贪心算法是否能够求得最优解？

- 在这样的币值种类下,再用贪心算法就行不通了,比如某人工资是140,按贪心算法140=100*(1张)+20*(2张)共需要3张,而事实上,只要取2张70面额的是最佳结果,这类问题可以考虑用动态规划算法来解决。

为什么？

- 因为，70破坏了“取最优”的贪心策略的正确性。

又为什么？

- 什么样的问题可以使用贪心算法策略，并获得最优解？

例3.2 活动安排问题

- n个活动E={1,2,..,n}，都要求使用同一公共资源（如演讲会场等）。且在同一时间仅一个活动可使用该资源。

  $i \in [s_i, f_i)$, s_i为起始时间，f_i为结束时间。$s_i < f_i$

  活动i和j相容：$s_i >= f_j$ 或 $s_j >= f_i$

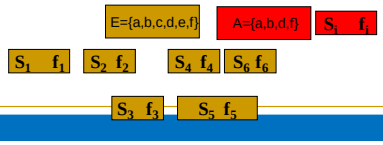
- 活动安排问题:求最大的相容活动子集合。

  ---尽可能多的活动兼容使用公共资源。

例3.2 活动安排问题-算法分析

1. 按结束时间递增排序：$f_1 <= f_2 <= .. <= f_n$　　--$O(n\log n)$
2. 从第1个活动开始，按顺序放进集合A。放入活动 I 当且仅当与集合 A 中已有元素相容。

   --与集合 A 中最后元素 j 比较：若 $s_i >= f_j$ 则加入，否则不加入 $f_j = \max_{k \in A}(f_k)$ ---集合 A 中的最大结束时间



例3.2 活动安排问题

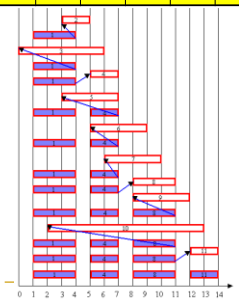
- 规则：选择具有最早结束时间的相容活动加入，使剩余的可安排时间最大，以安排尽可能多的活动。
- 由于输入的活动以其完成时间的递增排列，所以算法GreedySelector每次总是选择具有最早完成时间的相容活动加入集合A中。直观上，按这种方法选择相容活动为未安排活动留下尽可能多的时间。
- 也就是说，该算法的贪心选择的意义是使剩余的可安排时间段极大化，以便安排尽可能多的相容活动。

例3.2 活动安排问题算例

例：设待安排的11个活动的开始时间和结束时间按结束时间的递增排列如下：

i	1	2	3	4	5	6	7	8	9	10	11
S[i]	1	3	0	5	3	5	6	8	8	2	12
f[i]	4	5	6	7	8	9	10	11	12	13	14

i	1	2	3	4	5	6	7	8	9	10	11
S[i]	1	3	0	5	3	5	6	8	8	2	12
f[i]	4	5	6	7	8	9	10	11	12	13	14



例3. 2 活动安排问题

算法GreedySelector的计算过程如左图所示。图中每行相应于算法的一次迭代。阴影长条表示的活动是已选入集合A的活动，而空白长条表示的活动是当前正在检查相容性的活动。

例3.2 活动安排问题

若被检查的活动 i 的开始时间S_i小于最近选择的活动 j的结束时间f_i，则不选择活动i，否则选择活动i加入集合A中。

贪心算法并不总能求得问题的整体最优解。但对于活动安排问题，贪心算法GreedySelector却总能求得的整体最优解，即它最终所确定的相容活动集合A的规模最大。这个结论可以证明。

- 贪心算法GreedySelector—总能得到整体最优解，即A规模最大。
- 证明：

n个活动E={1,2,..,n}，按结束时间递增排序：f_1<=f_2 <= .. <= f_n。

1. 总存在一个最优解以贪心选择开始即包含活动1。

E按结束时间递增排序，活动1具有最早结束时间。

设A为最优解，A⊆E, A也按结束时间递增排序。

设A中第一活动为k，k=1时，显然成立。

k>1时，设B=A-{k}∪{1}，由于f_1 <=f_k且A中活动互为相容，则B中活动也互为相容。而B与A中活动个数相同，且A为最优解，则B为最优解。

2. 选择活动1以后，问题变为子问题E':与活动1相容的活动安排问题。

设A为包含活动1的最优解，则A'=A-{1}为E'的一个最优解。假如存在E'的一个解B'，|B'|>|A'|,则|B' ∪{1}|>A,与A为最优解矛盾。

由1,2 可知GreedySelector—总能得到整体最优解。

贪心法解题总结

- 首先选取一个度量标准（贪心准则），以这个标准把这n个输入排序，并按排序顺序一次输入一个量。然后把这个新的输入与当前已构成的在这种度量意义下的部分解加在一起，考察新增这个输入后是否能够产生一个可行解，如果不能则不把这个输入加入到这部分已经存在的可行解中（贪心选择）。

- 在一般情况下，贪心算法由一个迭代的循环组成，在每一轮的循环中，通过少量的局部的计算，去寻求一个局部的最优解。
- 每一步的工作都增加了部分解的规模，每一步的选择都极大的增长了它所希望实现的目标函数。所以它产生的算法特别有效。在很多实例中，它产生的局部解可以转化成全局最优解。
- 难点也就在于证明所设计的算法就是真正解这个问题的最优算法。

贪心算法的重要性质（1）

- 贪心选择性质

所谓贪心选择性质，是指所求解问题的全局最优解，可以通过一系列局部最优解的选择来达到。每进行一次选择，就得到一个局部解，并将求解的问题简化为一个规模更小的类似子问题。

贪心算法的重要性质（2）

- 最优子结构

所谓最优子结构，是一个问题的最优解中包含它的子问题的最优解。

例如在货币兑换问题中

- 银行出纳员支付一定数量的现金，他手中有各种面值的货币，要求他用最少的货币张数来支付现金。
- 假设手中有n张面值为p_i的货币，$1 \le i \le n$,用集合$P=\{p_1, p_2, \ldots, p_n\}$表示这些货币。
- 如果出纳员需要支付的现金为A，则他必须从P中选取一个最小的子集S,使得：

$$p_i \in S, \quad \text{且} \quad \sum p_i = A$$

问题求解：

如果用向量$X=\{x_1, x_2, \ldots, x_n\}$表示$S$ 中所选取的货币，使得：

$$x_i = \begin{cases} 1 & p_i \in S \\ 0 & p_i \notin S \end{cases}$$

那么，出纳员支付的现金必须满足：

$$\sum_{i=1}^{n} x_i p_i = A \qquad \text{（约束条件）}$$

且使得：

$$d = \min \sum_{i=1}^{n} x_i \qquad \text{（目标函数）}$$

满足约束条件的解：可行解，满足目标函数的解：最优解。

出纳员手中的货币：$P=\{p_1, p_2, \ldots, p_{60}\}$，10张10元，10张5元，10张1元，10张5角，10张2角，10张1角。$X=\{x_1, x_2, \ldots, x_{60}\}$。

支付：57元8角。

贪心选择性质：

在当前状态下，选择p_1=10 可以达到这个目的。于是，在第一步，所选择的货币集合是$S_1=\{p_1\}$，得到一个局部解。问题可以简化为在$P_1=\{p_2, \ldots, p_{60}\}$中挑选货币。支付：47元8角。

- **最优子结构**

所谓最优子结构，是一个问题的最优解中包含它的子问题的最优解。

上述问题的最优解是：

$S_n=\{p_1, p_2, p_3, p_4, p_5, p_{11}, p_{21}, p_{22}, p_{31}, p_{41}, p_{51}\}$

子问题的最优解是：

$S_{n-1}=\{p_2, p_3, p_4, p_5, p_{11}, p_{21}, p_{22}, p_{31}, p_{41}, p_{51}\}$，显然：

$$S_{n-1} \subset S_n \quad \text{并且} \quad S_{n-1} \cup \{p_1\} = S_n$$

3.2 贪心问题实例

- 例3.3 键盘输入一个高精度的正整数N，去掉其中任意S个数字后剩下的数字按原左右次序将组成一个新的正整数。编程对给定的N和S，寻找一种方案使得剩下的数字组成的新数最小。
- 输出应包括所去掉的数字的位置和组成的新的正整数(N不超过100位)。
- 数据结构设计：和上一节对高精度正整数的处理一样，将输入的高精度数存储为字符串格式。根据输出要求设置数组，在删除数字时记录其位置。

例3.3-问题分析

- 通过“枚举归纳”设计算法，实例(s=3)：
  - $n1$=“1 2 4 3 5 8 6 3”
- 贪心策略
  - 在位数固定的前提下，让高位的数字尽量小其值就较小；
  - 删除高位较大的数字；
  - 相邻两位比较，若高位比低位大则删除高位。
- n1=“1 2 4 3 5 8 6 3”
  - 4比3大 删除 “1 2 3 5 8 6 3”
  - 8比6大 删除 “1 2 3 5 6 3”
  - 6比3大 删除 “1 2 3 5 3”

- 再一个实例：
  - n2="2 3 1 1 8 3"
  - 3比1大 删除 "2 1 1 8 3"
  - 2比1大 删除 " 1 1 8 3"
  - 8比3大 删除 " 1 1 3"
- 由实例1可以看出，相邻数字只需要从前向后比较；而从实例2中可以看出当第*i*位与第*i*+1位比较，若删除第*i*位后，必须向前考虑第*i*-1位与第*i*+1位进行比较，才能保证结果的正确性。

例3.3-问题分析

- n3="1 2 3 4 5 6 7" s=3
  - 由这个实例看出，经过对n3相邻比较一个数字都没有删除，这就要考虑将后三位进行删除；
  - 当然还有可能，在相邻比较的过程中删除的位数小于s时，也要进行相似的操作。
- n4="1 2 0 0 8 3" s=3
  - 2比0大 删除 "1 0 0 8 3"
  - 1比0大 删除 " 0 0 8 3"
  - 8比3大 删除 " 0 0 3"
- 由这个实例子又能看出，当删除掉一些数字后，结果的高位有可能出现数字"0"，直接输出这个数据不合理，要将结果中高位的数字"0"删除掉，再输出。特别地还要考虑若结果串是"0000"时，不能将全部"0"都删除，而要保留一个"0"最后输出。

例3.3-算法设计

- 根据以上实例分析，算法主要由四部分组成：初始化、相邻数字比较(必要时删除)、处理比较过程中删除不够s位的情况和结果输出。
- 其中删除字符的实现方法很多，如：
  - 1)物理进行字符删除，就是用后面的字符覆盖已删除的字符，字符串长度改变。这样可能会有比较多字符移动操作，算法效率不高。
  - 2) 可以利用数组记录字符的存在状态，元素值为"1"表示对应数字存在，元素值为"0"表示对应数字已删除。这样避免了字符的移动，字符串长度不会改变，可以省略专门记录删除数字的位置。但这样做前后数字的比较过程和最后的输出过程相对复杂一些。

- 3)同样还是利用数组，记录未删除字符的下标，粗略的过程如下：
- n="1 2 4 3 5 8 3 3" s=3 1 2 3 4 5 6 7 8
  - 4比3大 删除 "1 2 4 3 5 8 3 3" 1 2 4 5 6 7 8
  - 8比3大 删除 "1 2 4 3 5 8 3 3" 1 2 4 5 7 8
  - 5比3大 删除 "1 2 4 3 5 8 3 3" 1 2 4 7 8
- 这时数组好象是数据库中的索引文件。此方式同样存在操作比较复杂的问题。

例3.4 数列极差问题

在黑板上写*N*个正整数排成一个数列，进行如下操作：每一次擦去其中的两个数*a*和*b*，然后在数列中加入一个数*a*×*b*+1，如此下去直至黑板上剩下一个数，在*所有按这种操作方式*最后得到的数中，最大的记作max，最小的记作min，则该数列的极差定义为*M*=max-min。

通过实例来认识题目中描述的计算过程。

对三个具体数据3，5，7讨论，可能有以下三种结果：

3，5，7 -> (3*5+1)=16，7-> (3*5+1)*7+1=113,

3，5，7 -> (3*7+1)=22，5-> (3*7+1)*5+1=111,

3，5，7 -> (5*7+1)=36，3 ->(5*7+1)*3+1=109

结论：先运算小数据得到的是最大值，先运算大数据得到的是最小值。

下面以三个数为例证明此题用贪心策略求解的合理性，不妨假设：$a<b=a+k1<c=a+k1+k2$，$k1$，$k2>0$，则有以下几种组合计算结果：

1) $(a*b+1)*c+1=a*a*a+(2k1+k2)a*a+(k1(k1+k2)+1)*a+k1+k2+1$
2) $(a*c+1)*b+1=a*a*a+(2k1+k2)a*a+(k1(k1+k2)+1)*a+k1+1$
3) $(b*c+1)*a+1=a*a*a+(2k1+k2)a*a+(k1(k1+k2)+1)*a+1$

显然此问题适合用贪心策略，不过在求最大值时，要先选择较小的数操作。求最小值时，要先选择较大的数操作。这是一道两次运用贪心策略解决的问题。

例3.4-算法设计

1）不断从现有的数据中，选取最大和最小的两个数，计算后的结果继续参与运算，直到剩余一个数算法结束。

2）选取最大和最小的两个数较高效的算法是用二分法完成，这里仅用简单的逐个比较方法来求解。注意到由于找到的两个数将不再参与其后的运算，其中一个用它们的计算结果代替，另一个用当前的最后一个数据覆盖即可。所以不但要选取最大和最小，还必须记录它们的位置，以便将其覆盖。

3）求max、min过程必须独立，即求max和min都必须从原始数据开始，否则不能找到真正的max和min。

例3.4-算法分析

算法的主要操作就是比较查找和计算，都是线性的，因此算法的时间复杂度为$O(n)$。由于计算最大结果和计算最小结果需要独立进行，所以算法的空间复杂度为$O(2n)$。

例3.5

设计一个算法，把一个真分数表示为埃及分数之和的形式。所谓埃及分数，是指分子为1的分数。如 $\frac{7}{8}=\frac{1}{2}+\frac{1}{3}+\frac{1}{24}$

基本思想：逐步选择分数所包含的最大埃及分数，这些埃及分数之和就是问题的一个解。

如：7/8>1/2，
7/8-1/2>1/3，
7/8-1/2-1/3=1/24。

过程如下：
1）找最小的n（最大的埃及分数1/n），使分数f >1/n；
2）输出1/n；
3）计算f=f-1/n；
4）若此时的f是埃及分数，输出 f，算法结束，否则返回1）。

例3.5-问题模型

◆记真分数$F=A/B$；对B/A进行整除运算，商为D，余数为$0<K<A$，它们导出关系如下：

$B=A*D+K$，$B/A=D+K/A<D+1$，$A/B>1/(D+1)$，记$C=D+1$。

◆这样分数F所包含的“最大”埃及分数就是$1/C$。

进一步计算：$A/B-1/C=(A*C-B)/B*C$

◆也就是说继续要解决的是有关分子为$A=A*C-B$，分母为$B=B*C$的问题。

例3.5-算法设计

由以上数学模型，算法过程如下：

1）设某个真分数的分子为A（≠1），分母为B；
2）把B除以A的商的整数部分加1后的值作为埃及分数的一个分母C；
3）输出$1/C$；
4）将A乘以C减去B作为新的A；
5）将B乘以C作为新的B；
6）如果A大于1且能整除B，则最后一个分母为B/A；
7）如果$A=1$，则最后一个分母为B；否则转步骤2）。

实例：7/8=1/2+1/3+1/24的解题步骤：

同样用变量*A*表示分子，变量*B*表示分母；

C=8/7+1=2　　//说明7/8>1/2，

打印1/2

A=7*2-8=6,*B*=*B***C*=16 //计算7/8-1/2=(7*2-8)/(7*2)=6/16=*A*/*B*

C=16/6+1=3　　//说明16/6>1/3，

打印1/3

A=6*3-16=2,*B*=*B***C*=16*3=48

　　//计算6/16-1/3=(6*3-16)/(16*3)=2/48=*A*/*B*

A>1但*B*/*A*为整数24，打印1/24 结束。

例3.6-取数游戏-有2个人轮流取2*n*个数中的*n*个数，取数之和大者为胜。请编写算法，让先取数者胜，模拟取数过程。

例3.6-问题分析

这个游戏一般假设取数者只能看到2 n个数中两边的数, 用贪心算法的情况：

若一组数据为：6,16,27,6,12,9,2,11,6,5。用贪心策略每次两人都取两边的数中较大的一个数, 先取者胜。

以A先取为例，取数结果：

A 6,27,12,5,11=61 胜

B 16,6,9,6,2=39

例3.6-问题分析

若选另一组数据：16,27,7,12,9,2,11,6。仍用贪心算法，先取者A败。

取数结果：

A 16,7,9,11=43

B 27,12,6,2=47　胜

若只能看到两边的数据，则此题无论先取还是后取都无必胜的策略。这时一般的策略是用近似贪心算法。

但若取数者能看到全部2*n*个数，则此问题可有一些简单的方法，有的虽不能保证所取数的和是最大，却是一个先取者必胜的策略。

例3.6-数学模型

数学模型：N个数排成一行，从左到右编号，依次为1，2，…，N，因为N为偶数，又因为我们先取数，计算机后取数，所以一开始我们既可取到一个奇编号数，又可取到一个偶编号数。

若我们第一次取奇编号数，则计算机只能取到偶编号数；

若我们第一次取偶编号数，则计算机只能取到奇编号数；

所以，只要比较奇编号数之和与偶编号数之和谁大，以决定最开始我们是取奇编号数还是偶编号数即可。（若同样大，我们第一次可任意取数，因为当两和相同时，先取者胜。）

例3.6-算法设计

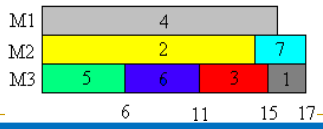
算法设计：只需分别计算一组数的奇数位和偶数位的数据之和，然后先取数者可以确定必胜的取数方式。

以下面一排数为例：1 2 3 10 5 6 7 8 9 4

奇编号数之和为25（=1+3+5+7+9），小于偶编号数之和为30（=2+10+6+8+4）。我们第一次取4，以后，计算机取哪边数我们就取哪边数（如果计算机取1，我们就取2；如果计算机取9，我们就取8）。这样可保证我们自始自终取到偶编号数，而计算机自始自终取到奇编号数。

例3.7-多机调度问题

- 设有n个独立的作业{1,2,…, n }，由m台相同的机器进行加工处理。
- 作业i所需的处理时间为t_i。现约定，每个作业均可在任何一台机器上加工处理，但未完工前不允许中断处理。作业不能拆分成更小的子作业。
- 多机调度问题要求给出一种作业调度方案，使所给的n个作业在尽可能短的时间内由m台机器加工处理完成。

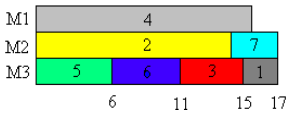


例3.7-问题分析

- 这个问题是“NP完全问题”，到目前为止还没有“有效”的解法。对于这一类问题,用贪心选择策略有时可以设计出较好的近似算法。
- 采用最长处理时间作业优先的贪心选择策略可以设计出解多机调度问题的较好的近似算法。
  - 按此策略，当m<=n时，只要将机器i的[0, t_i]时间区间分配给作业i即可。
  - 当m > n 时，首先将n个作业依其所需的处理时间从大到小排序。然后依此顺序将作业分配给空闲的处理机。

例3.7-算法实现

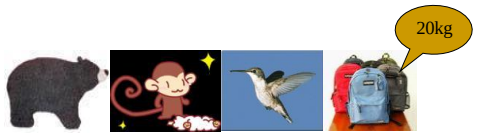
- 1 先将n个作业从小到大排序，存入数组a；
- 2 将排好序的后m个作业按照从小到大存入数组b；
- 3 循环 n-m次，将 a中剩余的最大项(从a[n-m]到a[1])与b中最小项相加；
- 在计算过程中输出分配情况，b中最大元素为加工时间。



- 例如，设7个独立作业{1,2,3,4,5,6,7}由3台机器M1，M2和M3加工处理。各作业所需的处理时间分别为{2,14,4,16,6,5,3}。
- 计算结果：所需的加工时间为17。

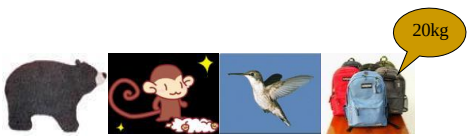
例3.8-背包问题：森林里进行一场装背包比赛：

- 参加者：猴子 黑熊 啄木鸟
- 每个比赛者一个背包，背包的载重量为20公斤。
- 给定N个物品，每个物品有一定的重量和价值。



森林里进行一场装背包比赛

- 规则：
  物品可切一部分放入；
  背包里装的物品的总重量不超过背包的载重量；
  背包里装的物品的价值最高者获胜。

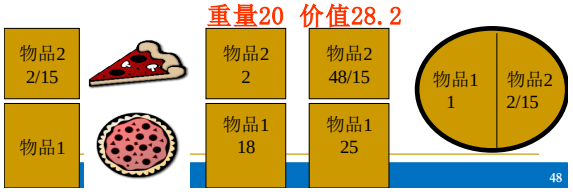


黑瞎子掰棒子的策略

黑　熊

- 黑瞎子掰棒子的策略：价值高的优先放入。
- 物品n=3，背包的载重量C=20
- 各个物品的价值(v_1,v_2,v_3)=(25,24,15)
- 各个物品的重量(w_1,w_2,w_3)=(18,15,10)。

重量20　价值28. 2



猴子耍小聪明策略

猴　子

- 耍小聪明策略：重量小的优先放入。
- 物品n=3，背包的载重量C=20
- 各个物品的价值(v_1,v_2,v_3)=(25,24,15)
- 各个物品的重量(w_1,w_2,w_3)=(18,15,10)。

重量20　价值31

啄木鸟算盘子策略

啄木鸟

- 算盘子策略：单位重量价值高的优先放入。
- 物品n=3，背包的载重量C=20

(v_1,v_2,v_3)=(25,24,15)　(w_1,w_2,w_3)=(18,15,10)

(v_1/w_1, v_2/w_2, v_3/w_3)=(1.39,1.6,1.5)

重量20　价值31.5

啄木鸟算盘子策略

- 啄木鸟算盘子策略的时间复杂度？　$O(n^2)$

第一步：选出单位重量价值最高者装入。

　　n个中取最大值　$O(n)$

第二步：删除该物品。

第三步：重复1, 2步, 直至再装入就超出背包的载重量为止。　$O(n)$

第四步：把最后选择物品的一部分装入背包：

　　剩余载重量/最后选择物品的重量

啄木鸟算盘子策略

- 能否改进？　时间复杂度？　$O(n\log n)$

第一步：按照单位重量价值递减排序。

　　n个数排序　$O(n\log n)$

第二步：按排序顺序依次装入直至再装入就超出背包的载重量为止。　$O(n)$

第三步：把最后选择物品的一部分装入背包：

　　剩余载重量/最后选择物品的重量

背包问题

该问题就是背包问题：

已知：给定n种物品和一个背包。

　　物品i重量是w_i，其价值为v_i，背包容量为C。

求解：如何选择装入背包的物品，使得装入背包中物品总价值最大？

背包问题的形式化描述

- 每个物品 x_i
- 可以不被装入背包，也可以部分装入背包，$0 \leqslant x_i \leqslant 1$
- 每个物品的价值和重量值都大于0，总共有1个或者 n 个物品，$v_i>0$，$w_i>0$，$1 \leqslant i \leqslant n$
- 约束条件：背包载重量是 C，因此选入背包中物品的总重量不得超过 C

$$\sum_{1 \leqslant i \leqslant n} w_i x_i \leq C$$

背包问题

- 问题的求解目标：背包中的物品总价值最大。
- 目标函数：$\max \sum_{1\le i\le n} v_i x_i$

啄木鸟的选择是否使总价值最大?

最优解的证明

- 第一步证明：

首先选择单位重量价值最高的物品装入是正确的。

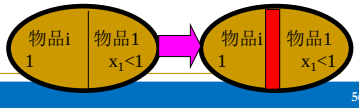
假设物品已按单位重量价值递减顺序排序且w_1<C。

证明：反证法。

假设存在最优解：$v_1x_1+v_2x_2+v_3x_3$

如果x_1=1显然成立。

如果x_1<1，从背包中取包含x_1的重量为w_1的一部分，用物品1代替。



最优解的证明

因此最优解中肯定包含单位重量价值最高的物品。

这样我们证明了首先选择单位重量价值最高的物品装入是正确的。

放入单位重量价值最高的物品以后，原问题变为子问题。

$$\sum_{1\le i\le n} x_i \quad \sum_{1\le i\le n} w_i x_i \le C \;\Rightarrow\; \sum_{2\le i\le n} x_i \quad \sum_{2\le i\le n} w_i x_i \le C - W_1$$

最优解的证明

- 第二步证明：

如果存在包含单位重量价值最高物品的最优解A，则A'=A-V1是选择单位重量价值最高的物品装入后子问题的最优解。

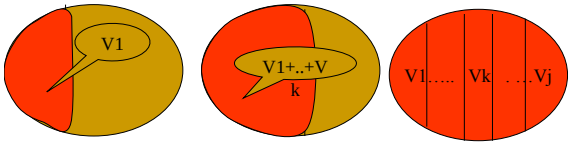
证明：反证法。

假如该子问题比A'更好的最优解B'，则B'+V1>A'+V1=A

这与A为原问题的最优解相矛盾。

最优解的证明

- 第一步选择正确。
- 选择后的子问题的最优解为A-V1。
- 由归纳法可证明，经这样的选择最终得到原问题的最优解。



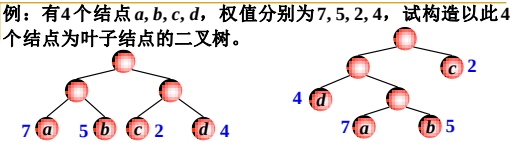
用贪心法求解背包问题的分析

- 黑熊策略：目标函数作为度量标准。此标准虽然使一个物品产生最大价值效益，但是它消耗载重量空间过快。
- 猴子策略：使背包载重量尽可能慢地被消耗作标准。此标准虽然使背包的载重量消耗慢了，但是背包的价值却没有能够迅速地增加。
- 啄木鸟策略：把二者综合起来，以单位重量中的最大价值(尽可能多的单位价值高的物品装入)作为衡量标准，即考虑v_i/w_i的值，使所占用的单位重量所带来的价值最大。因此把v_i/w_i的值按降序排序，v_i/w_i最高的首先放入背包，v_i/w_i次高的接着放入背包，依此类推。

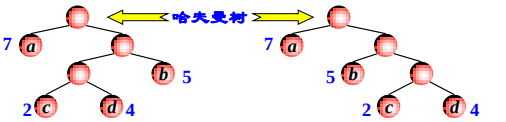
例3.9 Huffman编码

- Huffman树的构造算法

例：有4个结点 *a*, *b*, *c*, *d*，权值分别为7, 5, 2, 4，试构造以此4个结点为叶子结点的二叉树。



WPL=7×2+5×2+2×2+4×2= 36　　WPL=7×3+5×3+2×1+4×2= 46



WPL=7×1+5×2+2×3+4×3= 35　　WPL=7×1+5×2+2×3+4×3= 35

具有相同带权结点构成的哈夫曼树不惟一。

满二叉树不一定是哈夫曼树。

哈夫曼树：

目　标

带权路径长度(WPL) 最短的二叉树

(权值大的结点离根最近)

因为构造这种树的算法是由哈夫曼于1952年提出的，

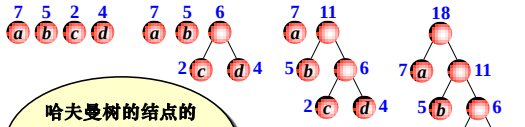
所以被称为哈夫曼树，相应的算法称为哈夫曼算法。

哈夫曼算法口诀：

1、构造森林全是根；　　2、选用两小造新树；

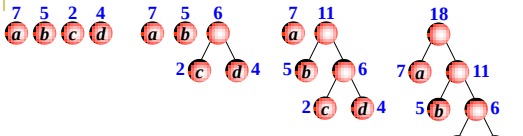
3、删除两小添新人；　　4、重复2、3剩单根。

例：有4个结点 *a*, *b*, *c*, *d*，权值分别为7, 5, 2, 4，构造哈夫曼树。



哈夫曼树的结点的度数为 0 或 2，没有度为 1 的结点。

例：有4个结点 *a*, *b*, *c*, *d*，权值分别为7, 5, 2, 4，构造哈夫曼树。



- 贪心策略
  - 每次合并两棵具有最小权值的树，新树权值为两树之和；
  - 满足贪心选择性质；
  - 需证明：设NodeSet是结点集合，其中结点i的权值是w(i)，又设x，y是NodeSet中具有最小权值的两个节点，则存在NodeSet组成的一个huffman树，使x，y是最深叶子且为兄弟。

1.证明：具有贪心选择性质

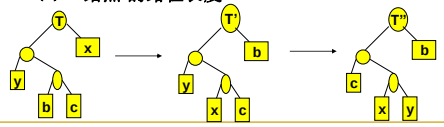
设二叉树T是一棵huffman树。b、c是二叉树T的最深叶子且为兄弟。不失一般性，设w(b)≤w(c)，设x、y是NodeSet中权值最小的两结点， w(x)≤w(y)，则有w(x) ≤ w(b)， w(y)≤w(c)

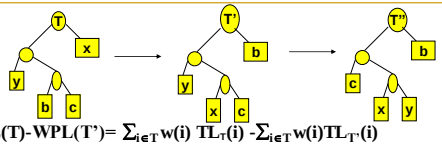
在树T中交换叶子b和x的位置得到树T'，再交换叶子c和y的位置，得到树T''。

带权路径长度：$WPL(T)=\sum_{i\in T} w(i)TL(c)$

i—二叉树上的结点　w(i)—结点i的权值

TL（i）--结点i的路径长度





WPL(T)-WPL(T')= $\sum_{i\in T} w(i)\,TL_T(i) - \sum_{i\in T} w(i)TL_{T'}(i)$

= $w(x)TL_T(x)+w(b)TL_T(b)-w(x)TL_{T'}(x) - w(b)TL_{T'}(b)$

= $w(x)TL_T(x) - w(x)TL_T(b) + w(b)TL_T(b)-w(b)TL_T(x)$

= $(w(b)-w(x))(TL_T(b)- TL_T(x)) \geq 0$

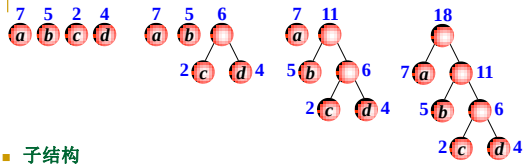
同理： WPL(T')-WPL(T'') ≥0 // $TL_T(b)=TL_{T'}(x)$

则 WPL(T'') ≤ WPL(T') ≤ WPL(T)

又T是哈夫曼树。 WPL(T) ≤WPL(T'') 则 WPL(T)=WPL(T'')。

T''是含最小权值结点的哈夫曼树，x和y为最深叶结点且为兄弟。

例：有4个结点 *a*, *b*, *c*, *d*，权值分别为7, 5, 2, 4，构造哈夫曼树。



- 子结构
  - 设T是表示NodeSet的huffman树，设x，y是两个叶子结点，且互为兄弟，z是它们的父亲。若将z看成权值为w(z)=w(a)+w(b)的结点，则对于：
    树T '=T - {x，y } ，结点集合N ' = NodeSet -{x，y}U{ z}，
    T '是N '的huffman树，是原问题的子结构。
  - 满足最优子结构性质。可以证明。

2.证明：具有最优子结构性质

二叉树T是哈夫曼树。x、y是二叉树T的最深叶子且为兄弟。

设z为其父亲，w(z)=w(x)+w(y)，则树T'=T-{x，y}是结点集合N'=NodeSet-{x，y}∪{z}的哈夫曼树。

证明：反证法。

由已知不难得出，$TL_T(x)=TL_T(y) = TL_{T'}(z)+1$

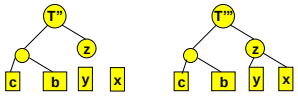
对任意的i ∈NodeSet-{x，y}，有 $TL_{T'}(i)=TL_T(i)$

WPL(T)-WPL(T') = $\sum_{i\in N} w(i)TL_T(i) - \sum_{i\in N'} w(i)TL_{T'}(i)$

= $w(x)TL_T(x)+w(y)TL_T(y) - w(z)TL_{T'}(z)$

= $(w(x)+w(y))\,TL_{T'}(z) + w(x)+w(y) - w(z)TL_{T'}(z)$

= $w(x)+w(y)$



假定T'不是结点集合N' 的哈夫曼树，则存在结点集合N' 的哈夫曼树，设为T''。则有WPL(T')>WPL(T'')

z是T''中的一个叶子，将x,y 加入作为z的儿子，得到结点集合N的二叉树T'''。

WPL(T''') = $\sum_{i\in N'} w(i)TL_{T''}(i)+w(x)(TL_{T''}(z)+1)+ w(y)(TL_{T''}(z)+1) - w(z)TL_{T''}(z)$

= $\sum_{i\in N'} w(i)TL_{T''}(i)+w(x)+w(y) \leq WPL(T')+ w(x)+w(y)$

=WPL(T)

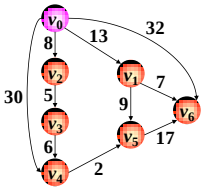
与T是结点集合N的哈夫曼树矛盾。

- 例3.10单源最短路径—Dijkstra算法

给定一个带权有向图G=（V，E），V={1,2,…,n }，其中每条边的权是一个非负实数。给定一个顶点 v，称为源。

单源最短路径问题：求从源到各顶点的最短路长度（路径上各边权之和）。

- 实例



最短路径	长度
(v_0, v_1)	13
(v_0, v_2)	8
(v_0, v_2, v_3)	13
(v_0, v_2, v_3, v_4)	19
$(v_0, v_2, v_3, v_4, v_5)$	21
(v_0, v_1, v_6)	20

- **路径长度最短的最短路径的特点：**

在此路径上，必定只含一条弧 $<v_0, v_1>$，且其权值最小。

$<a, c>$

- **下一条路径长度次短的最短路径的特点：**

它只可能有两种情况：

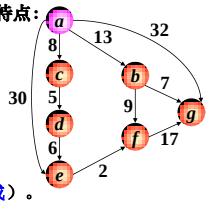
1)、直接从源点到 v_2 < v_0, v_2>

（只含一条弧）；**<a, b>**

2)、从源点经过顶点 v_1，再到达 v_2

< v_0, v_1>, < v_1, v_2>（由两条弧组成）。

$<a, c>$, $<c, d>$



- **再下一条路径长度次短的最短路径的特点：**

可能有四种情况：

1)、直接从源点到 v_3 <v_0, v_3>（由一条弧组成）；**<a, e>**

2)、从源点经过顶点 v_1，再到达 v_3 <v_0, v_1>, <v_1, v_3>

（由两条弧组成）；

3)、从源点经过顶点 v_2，再到达 v_3

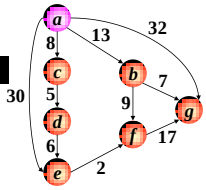
<v_0, v_2>, <v_2, v_3> **<a, b>，<b, g>**

（由两条弧组成）；

4)、从源点经过顶点 v_1,v_2，再到达 v_3

<v_0, v_1>, <v_1, v_2>, <v_2, v_3>

（由三条弧组成）；$<a, c>$，$<c, d>$，$<d, e>$

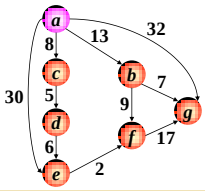


- **其余最短路径的特点：**

1)、直接从源点到 v_i <v_0, v_i >（只含一条弧）；

2)、从源点经过已求得的最短路径上的顶点，再到达 v_i

（含有多条弧）。



迪杰斯特拉 (Dijkstra) 算法：按路径长度递增次序产生最短路径

1、把 V 分成两组：

(1) S：已求出最短路径的顶点的集合。

(2) V - S = T：尚未确定最短路径的顶点集合。

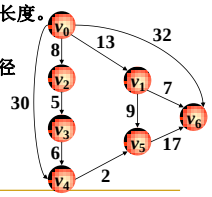
2、将 T 中顶点按最短路径递增的次序加入到 S 中，

保证：(1) 从源点 v_0 到 S 中各顶点的最短路径长度都不大于从 v_0 到 T 中任何顶点的最短路径长度。

(2) 每个顶点对应一个距离值：

S 中顶点：从 v_0 到此顶点的最短路径长度。

T 中顶点：从 v_0 到此顶点的只包括 S 中顶点作中间顶点的最短路径长度。



Dijkstra 算法步骤： 初始时令 S={v_0}，T={其余顶点}。

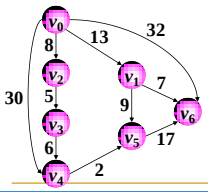
T 中顶点对应的距离值用辅助数组 D 存放。

$D[i]$ **初值：**若 $<v_0, v_i>$ 存在，则为其权值；否则为∞。

从 T 中选取一个其距离值最小的顶点 v_j，加入 S。 $D[j] = \min_i\{D[i] \mid v_i \in T\}$

对 T 中顶点的距离值进行修改：若加进 v_j 作中间顶点，从 v_0 到 v_i 的距离值比不加 v_j 的路径要短，则修改此距离值。

重复上述步骤，直到 S = V 为止。



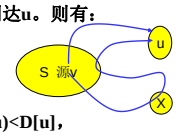
终点	从 v_0 到各终点的最短路径及长度					
	i=1	i=2	i=3	i=4	i=5	i=6
v_1	13	13				
v_2	8					
v_3	∞	∞	13	13		
v_4	30	30	30	19		
v_5	∞	∞	22	22	21	21
v_6	32	32	20	20	20	
v_j	v_2	v_1	v_3	v_4	v_6	v_5
s	8	13	8+5	8+5+6	13+7	13+9

- 贪心策略
  - 每一次迭代，从V - S中选择具有最短路径的顶点u(除了u，这个最短路径必须只经过S中的点)，从而确定从源点到u的最短路径长度$D[u]$。
  - 满足贪心选择性质；
  - 需证明：不存在一条经过S之外的路径，从源点到u，且长度小于$D[u]$；
  - 反证法。

终点	从 v_0 到各终点的最短路径及长度					
	i=1	i=2	i=3	i=4	i=5	i=6
v_1	13	13				
v_2	8					
v_3	∞	13	13			
v_4	30	30	30	19		
v_5	∞	∞	22	22	21	21
v_6	32	32	20	20	20	
v_j	v_2	v_1	v_3	v_4	v_6	v_5
s	8	13	8+5	8+5+6	13+7	13+9

1. 证明：贪心选择性质

- 贪心选择从V-S中选择具有最短特殊路径的顶点u，从而确定从源到该顶点u的最短路长度D[u]。
- 假如存在从源到u且长度＜D[u]的路径，设为Y。
- 如图，设这条路径初次走出S后首先经过顶点x∈V-S，然后徘徊于S内外若干次，最后离开S到达u。则有：



D[x] ≤d(v,x)

d(v,x)+d(x,u)=d(v,u)<D[u]

又权∈R+

则 D[x] ≤d(v,x)< d(v,x)+d(x,u)=d(v,u)<D[u]，

与D[u]是V-S中经过且只经过S中的顶点到u的最短路径相矛盾。

2.证明：最优子结构性质

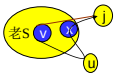
- 贪心选择从V-S中选择最短路径的顶点u后，可能出现一条到顶点j的新的特殊路。将添加u之前的S称为老的S。
- 如果出现从v经老的S中其他顶点到达u，再从u经一条边到达顶点j的新路，则有 newD=D[u]+ w[u][j] <D[j]

  则新路D[u]+w[u][j]是当前从v到达顶点j的最短路。
- 如果存在从v经u再经老S中其他顶点(设为x)到达顶点j的新路，则有 D[u]+d(u,x)+d(x,j) ＜D[j]

  又x比u先加入S中，所以D[x]≤D[u]

  则D[x]+d(x,j) ＜D[u]+d(u,x)+d(x,j) ＜D[j]

  此为矛盾。



- 因此从V-S中选择最短路径的顶点u后如果出现从v经u到达顶点j的新路则新路是当前从v到达顶点j的最短路。

例3.11-最小生成树

- 设G =(V,E)是无向连通带权图，即一个网络。E中每条边(v,w)的权为c[v][w]。
- 生成树：如果G的子图G'是一棵包含G所有顶点的树，则称G'为G的生成树。生成树上各边权的总和称为该生成树的耗费。
- 最小生成树：所有生成树中耗费最小的生成树。
- 网络的最小生成树在实际中有广泛应用。例如，在设计通信网络时，用图的顶点表示城市，用边(v,w)的权c[v][w]表示建立城市v和w之间的通信线路所需费用，则最小生成树就给出了建立通信网络的最经济的方案。

最小生成树性质

- MST性质：设G=(V,E)是连通带权图，U是V的真子集。U⊂V，如果（u，v）∈E且u∈U， v∈V - U，在所有这样的边中（u，v）的权最小，一定存在一棵以（u，v）为边的最小生成树。
- 证：假设不存在G的最小生成树包含边（u，v）。设T为一棵最小生成树，将边（u，v）加入，将产生一个含有边（u，v）的圈，圈中必存在另一条边（u'，v'）∈E且u'∈U， v'∈V - U。将边（u'，v'）删除，得到另一棵生成树T'。由c（u，v）〈=c（u'，v'），则T'比T耗费更小。矛盾。

贪心法解最小生成树问题

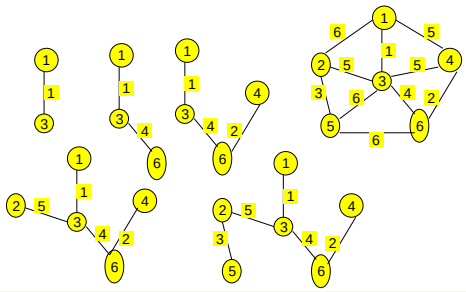
用贪心算法设计策略可以设计出构造最小生成树的有效算法。本节介绍的构造最小生成树的Prim算法和Kruskal算法都可以看作是应用贪心算法设计策略的例子。尽管这两个算法做贪心选择的方式不同，它们都利用了上面的最小生成树性质。

Prim算法（普里姆）

- 关键是选取度量标准：选择迄今为止所记入的那些边的成本的和有最小增量的那条边。
- 使得迄今所选择的边的集合S构成一棵树，将要记入到S中的下一条边(u,v)是一条不在S中且使得S∪{(u,v)}也是一棵树的最小成本的边。这是Prim方法。
- 设G=(V,E)是一个连通带权图，V={1,2,…,n}。

  首先置S={1}，然后只要S是V的真子集，就作如下的贪心选择：选取（u，v）∈E且u∈S， v∈V-S，在所有这样的边中（u，v）的权最小。v加入，…直至S=V为止。

- n-1条边的最小生成树



注意：

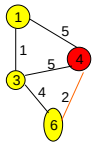
- 若候选轻权边集中的轻权边不止一条，可任选其中的一条扩充到T中。
- 连通图的最小生成树不一定是惟一的，但它们的权相等。

如何找出满足条件的最小权边

在上述Prim算法中，还应当考虑如何有效地找出满足条件i∈S,j∈V-S，且权c[i][j]最小的边(i,j)。实现这个目的较简单的办法是设置2个数组closest和lowcost。

在Prim算法执行过程中，先找出V-S中使lowcost值最小的顶点j，然后根据数组closest选取边(j,closest[j])，最后将j添加到S中，并对closest和lowcost作必要的修改。

取边（i，j）i ∈S 且j ∈V-S 的最小权边：



- 设对于某个结点j，数组closest(j)是已经在最小生成树中与j有边相连并且相连的边权值最小的那个结点。
- j已经在树中， closest(j)=0 ；如果j不在树中，则在树中寻找closest(j) 结点。

对于任意 k ∈V-S,计算closest(k)。确定使lowcost(j) =min C(k, closest(k))

- 边(j, closest(j))成为树边后，需要修改剩下不属于树的结点的closest值。
- 如果某个树外结点k到j的边值小于到{S-j}的边值= lowcost(k) ，则closest(k)=j。

普里姆算法的归纳法证明

假设由普里姆算法所产生的最小生成树的边集是T，无向带权图$G=(V,E,W)$的最小生成树是T^*，则有：

（1）开始时，$T=\varphi$，上述论点成立。

（2）假设算法在把边 $e=(i,j)$ 加入到 T 之前，论点成立，$G'=(S,T)$是 G 的最小生成树的子树，$N=V-S$。则按照普里姆算法，选择把 $e=(i,j)$ 加入 T 时，满足：$i \in S, j \in N$ 并且使 $c[i][j]$ 最小的 i 和 j 作为边 e 的关联顶点。有：

$$S'=S\cup\{j\},T'=T\cup\{e\},G'=(S',T')$$

因此有：

i) G'是树，因为 e 和 S 中的一个顶点关联，加入 e 后不会使 G' 产生回路，且G'仍联通。

ii) G' 是 G 的最小生成树，因为如果 $e \in T^*$，这个结论成立，如果 $e \notin T^*$

那么与 e 关联的顶点必定是T^*中的两个不相邻接的顶点，根据生成树的性质，$T^*\cup\{e\}$ 将包含在一个回路。E 是这个回路的一条边，且 $e=(i,j)$，$i \in S, j \in N$ 则回路中必定存在另外一条边 $e'=(x,y)$，$x \in S, y \in N$

按照普里姆算法的选择，e 的权小于或等于e' 的权。

令 $T^{**}=T^{*}\cup\{e\}-\{e'\}$

则 T^{**}的权小于或等于 T^{*} 的权。如果T^{**} 的权小于或等于T^{*} 的权，则与T^{*}是最小代价生成树矛盾。所以，$e\epsilon T^{*}$；若T^{**} 的权等于T^{*} 的权，这时用新的T^{*} 来标记T^{**}。在这两种情况下，都有 $e'\epsilon T^{*}$

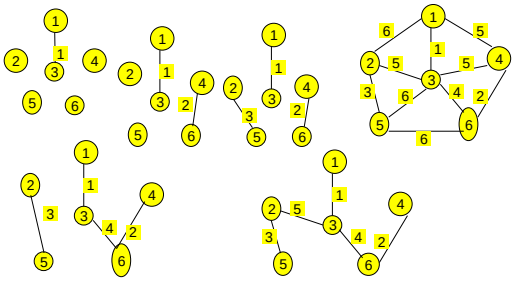
综上所述，$T=T^{*}$，普里姆算法所产生的生成树是G的最小生成树。

Kruskal方法（克鲁斯卡尔）

- 每一次选取不构成环路的最小成本的边。但是对于生成树迄今所选择的边集合T，应该使得它完成后的T有可能成为一棵树。这是Kruskal方法。
- N个顶点孤立，边按权排序。按权递增顺序选取边（v，w），使v，w分属不同的连通分支，…，直至一个连通分支为止。

Kruskal算法

该算法的特点：当前形成的集合T除最后的结果外，始终是一个森林。



Kruskal算法的抽象描述

```
KruskalMST(G){//求连通网G的一棵MST
   T=(V，φ)；//初始化，T是只含n个顶点不包含边的森林。
   依权值的递增序对E(G)中的边排序，并设结果在E[0..e-1]中
   for(i=0;i<e;i++) { //e为图中边总数
     取E[0．．e-1]中的第i条边(u，v)；
     if (u和v分别属于T中两棵不同的树)
                T=T∪{(u，v)}；//(u，v)是安全边，加入T中
     if (T已是一棵生成树) return T；
     }//endfor
     return T；
  }
```

分析：算法时间复杂度为$O(elge)$。主要取决于边数，较适合于稀疏图。

证明Kruskal算法的正确性

设 G是无向连通图，T* 是 G 的最小代价生成树的边集，T 是由Kruskal算法所产生的生成树边集。则 G 中的顶点，既是 T* 中的顶点，也是 T 中的顶点。若G的顶点树为n，则 $|T^{*}|=|T|=n-1$，下面用归纳法证明 T=T*。

证明：

（1）设e_1是G中权最小的边，根据Kruskal算法，有 $e_1\epsilon T$。此时，若 $e_1\notin T^{*}$

但因为T*是G的最小代价生成树，所以和e_1关联的顶点必定是T*中的两个不相邻的顶点，根据生成树的性质，把e_1加入T*，将使T*构成唯一的一条回路。假定这条回路是 $e_1,e_{a1},\cdots,e_{ak}$ ，且e_1是这条回路中权最小的边。

令 $T^{**}=T^{*}\cup\{e_1\}-\{e_{ai}\}$ ，e_{ai} 是回路 $e_1,e_{a1},\cdots,e_{ak}$ 中除e_1外的任意一条边，

则边集T**仍然是G的生成树，且T**的权小于或等于T*的权。

如果T**的权小于T*的权，则与T*是G的最小代价生成树的边集相矛盾，所以 $e_1\in T^{*}$ 。

如果T**的权等于T*的权，则与T**也是G的最小代价生成树的边集，且 $e_1\in T^{**}$

这时可以用新的T*来标记T**，在这两种情况下，都有 $e_1\in T^{*}$ 。

（2）若e_2是G中权第二小的边，同理可证 $e_2\in T,且e_2\in T^{*}$

（3）设 $e_1,\cdots,e_k$ 是G中前面k条权最小的边，且他们都属于T，也属于T*，令e_{k+1}是G中第k+1条权最小的边，且 $e_{k+1}\in T$ 但 $e_{k+1}\notin T^{*}$。

同样和e_{k+1}关联的顶点也是T*中的两个不相邻接的顶点。

把e_{k+1}加入T*，将使T*构成唯一的一条回路。假定这条回路是 $e_{k+1},e_{a1},\cdots,e_{am}$

在 $e_{a1},\cdots,e_{am}$ 中必有一条边 $e_{ai}\in T*$ 但 $e_{ai}\notin T$ ，否则T将存在回路。

因为 $e_1,\cdots,e_{k+1}$ 是G中前面k+1条权最小的边，并且 $e_1,\cdots,e_k$ 都属于T，所以

e_{ai}的权大于或等于e_{k+1}的权。令 $T**=T\bigcup\{e_{k+1}\}-\{e_{ai}\}$

则T**仍然是G的生成树，且T**的权小于或等于T*的权。同上面理由，必有：

$e_{k+1}\in T*$

（4）设 $e_1,\cdots,e_k$ 是G中前面k条权最小的边，且他们都属于T，也属于T*，令

e_{k+1}是G中第k+1条权最小的边，且 $e_{k+1}\notin T$ 但 $e_{k+1}\in T*$。

因为 $e_1,\cdots,e_k$ 都属于T，而 $e_{k+1}\notin T$ ，根据Kruskal算法，必有 $e_1,\cdots,e_{k+1}$

构成回路。因为 $e_1,\cdots,e_k$ 也属于T*，若 $e_{k+1}\in T*$ ，将使T*存在回路，所以

只有 $e_{k+1}\notin T*$ 。

综上所述，有T=T*，所以Kruskal算法正确的得到无向带权图的最小代价生成树。

小 结

- 理解贪心算法的概念；掌握贪心算法的基本思想。
- 掌握贪心算法的基本要素
- （1）贪心选择性质
- （2）最优子结构性质
- 理解可绝对贪心问题和相对或近似贪心问题。
- 理解贪心算法的一般理论；通过应用范例学习贪心设计策略。
- 理解各种问题的具体算法。

小 结

如何知道找到的解是最优的?

- 1. 证存在最优解包含贪心选择的开始值。

  证明:修改最优解,使其包含贪心选择的开始值后仍为最优解。

- 2. 证包含贪心选择的开始值的最优解,也是包含贪心选择的开始值后的子问题的最优解。

  -----最优子结构性质

  -反证

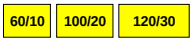
例子：0-1背包问题

- 0-1背包问题能否使用贪心算法求解?

  在选择装入背包的物品时，对每种物品 *i*只有2种选择，即装入背包或不装入背包。不能将物品 *i*装入背包多次，也不能只装入部分的物品 *i*。

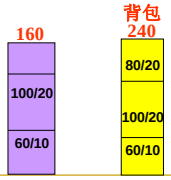
例子：0-1背包问题

- 例: n=3 C=50 v=(60,100,120) w=(10,20,30)
- vi / wi 排序 6 5 4



- 0-1背包

  啄木鸟算盘子策略



啄木鸟算盘子策略是否最优解?

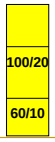
例子：0-1背包问题

- 例: n=3 C=50 v=(60,100,120) w=(10,20,30)
- vi / wi 排序

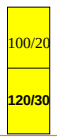
6	5	4
60/10	100/20	120/30

- 0-1背包

猴子策略



黑瞎子策略

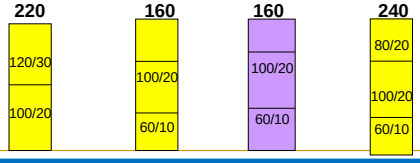


例子：0-1背包问题

- 例: C=50
- vi / wi

6	5	4
60/10	100/20	120/30

- 0-1背包

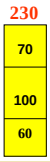


例子：0-1背包问题

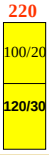
- 黑瞎子策略能否保证最优解?
- 例: n=4 C=55 v=(60,100,120,70)
  w=(10,20,30,20)

0-1背包

猴子策略



黑瞎子策略



例子：0-1背包问题

- 为什么0-1背包问题不能用贪心算法得到最优解?

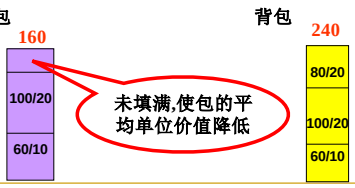
对于0-1背包问题，贪心选择之所以不能得到最优解是因为在这种情况下，它无法保证最终能将背包装满，部分闲置的背包空间使每公斤背包的价值降低了。

例子：0-1背包问题

- 例: C=50
- vi / wi

6	5	4
60/10	100/20	120/30

- 0-1背包



例子：0-1背包问题

事实上，在考虑0-1背包问题时，应比较选择该物品和不选择该物品所导致的最终方案，然后再作出最好选择。由此就导出许多互相重叠的子问题。

这正是该问题可用动态规划算法求解的重要特征。