算法分析与设计

Analysis and Design of Algorithm

Lesson 10

第三章小结

- 理解动态规划算法的概念
- 掌握动态规划算法的基本要素
  - 最优子结构性质
  - 重叠子问题性质
- 掌握设计动态规划算法的步骤
  - 建模→分段→分析→求解
- 动态规划法应用实例
  - 最短路径、矩阵连乘（递归、迭代）
  - 最长公共子序列、背包问题

第四章 贪心算法

学习要点

- 理解贪心算法的概念，掌握贪心算法的基本要素
  - 最优子结构性质
  - 贪心选择性质
- 理解贪心算法一般理论，与动态规划算法的差异
- 通过应用范例学习贪心设计策略
  1. 背包问题
  2. 活动安排问题
  3. 最优装载问题
  4. 哈夫曼编码
  5. 单源最短路径
  6. 最小生成树

贪心算法概述

贪心算法(Greedy Algorithm)

- 顾名思义，贪心算法总是作出在当前阶段看来最好的选择。也就是说贪心算法并不从整体最优考虑，它所作出的选择只是在某种意义上的局部最优选择，是一种“短视”行为。
  - 希望贪心算法得到的最终结果也是整体最优的。
- 不能对所有问题都得到整体最优解，但对许多问题能产生整体最优解。
  - 如单源最短路径问题，最小生成树问题等。
  - 在一些情况下，即使不能得到整体最优解，其最终结果却是最优解的很好近似。

贪心算法思想

- 从一个实例开始：**找零钱问题**

设有5、2、1元和5、2、1角货币，需找给顾客4元6角现金，为使付出的货币数目最少。

1. 首先选出1张最大面值不超过4元6角→2元；
2. 再选2元6角→ 2元；
3. 再选不超过6角的最大面值→ 5角；
4. 再选一张不超过1角的→ 1角；

共付出**4**张货币。

找零钱问题的策略

- 在每一步贪心选择中，在不超过应找零钱的条件下，只选择面值最大的货币，不考虑选择的合理性，且不会改变决定：一旦选出了一张货币，就永远选定。
  - 策略：尽可能使付出的货币最快地满足支付要求，目的是使付出的货币张数最慢增加。
  - 例如，上述实例用贪心法可以得到最优解。
  - 将面值改为3元、1元、8角、5角、1角
  - **贪心法：**结果是3元、1元、5角和1角各一张，共4张。
  - **最优解：**却为1个3元+2个8角，共3张。

贪心法求解的问题的特征

- 最优子结构性质
  - 当一个问题的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**，也称此问题满足最优性原理/优化原则。
- 贪心选择性质
  - 所谓贪心选择性质是指问题的**整体最优解**可以通过一系列**局部最优**的选择，即贪心选择来得到。

动态规划实质：

- 以**自底向上**的方式求解各个子问题

贪心算法实质：

- 以**自顶向下**的方式做出一系列的贪心选择

组合问题中的贪心算法

背包问题

- 一般背包问题的解是$\langle x_1, x_2, \ldots, x_n \rangle$，其中$x_i$是装入背包的第$i$种物品一部分，不一定要全部装入背包

  目标函数 $\max \sum_{i=1}^{n} v_i x_i$

  约束条件 $\sum_{i=1}^{n} w_i x_i \le b, \quad x_i \in [0,1]$

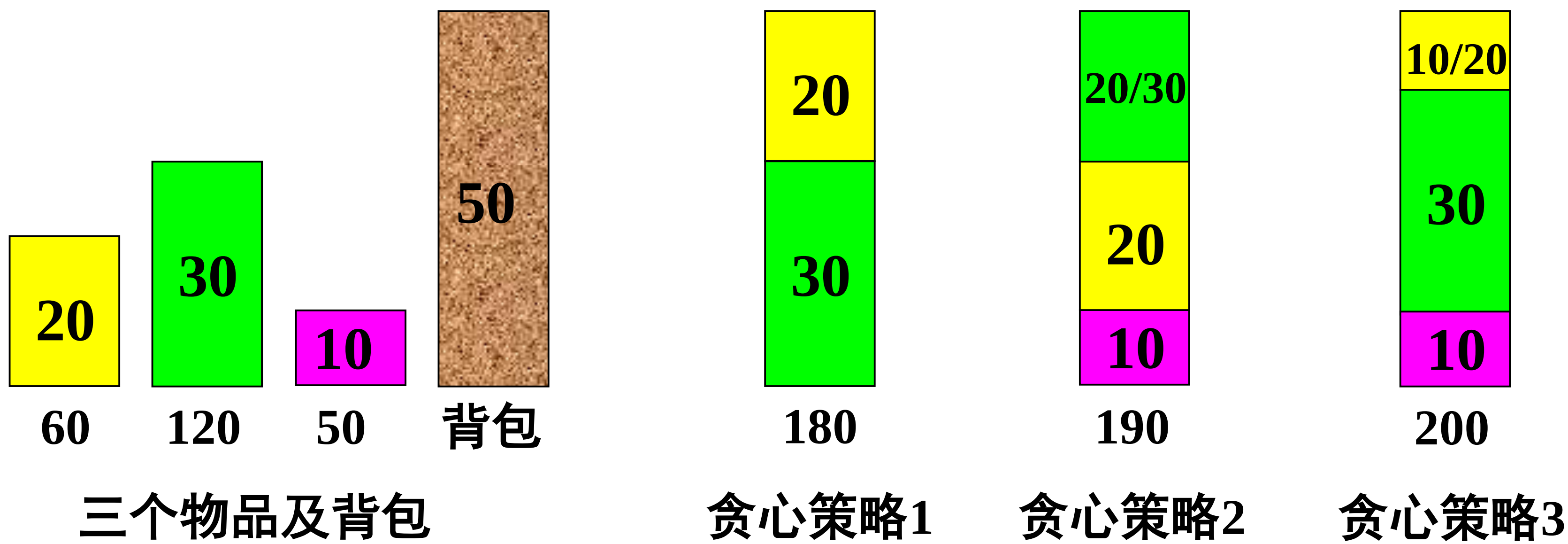
- 当 $x_i \in \{0,1\}$ 时，变为0-1背包问题的解
- 背包问题具有最优子结构性质!

背包问题的贪心法

- 至少有三种看似合理的贪心策略：
  1. 选择价值最大的物品，因为这可以尽可能快地增加背包的总价值。然而，虽然每一步选择获得了背包价值的极大增长，但背包容量却可能消耗得太快，使得装入背包的物品个数减少，从而不能保证目标函数达到最大。
  2. 选择重量最轻的物品，因为这可以装入尽可能多的物品，从而增加背包的总价值。但是，虽然每一步选择使背包的容量消耗得慢了，但背包的价值却没能保证迅速增长，从而不能保证目标函数达到最大。
  3. 选择单位重量价值最大的物品，在背包价值增长和背包容量消耗两者之间寻找平衡。

背包问题的贪心法

- **一般背包问题实例：**
  - 有3个物品，其重量分别是{20, 30, 10}，价值分别为{60, 120, 50}，背包的容量为50。应用三种贪心策略装入背包的物品和获得的价值如图所示。

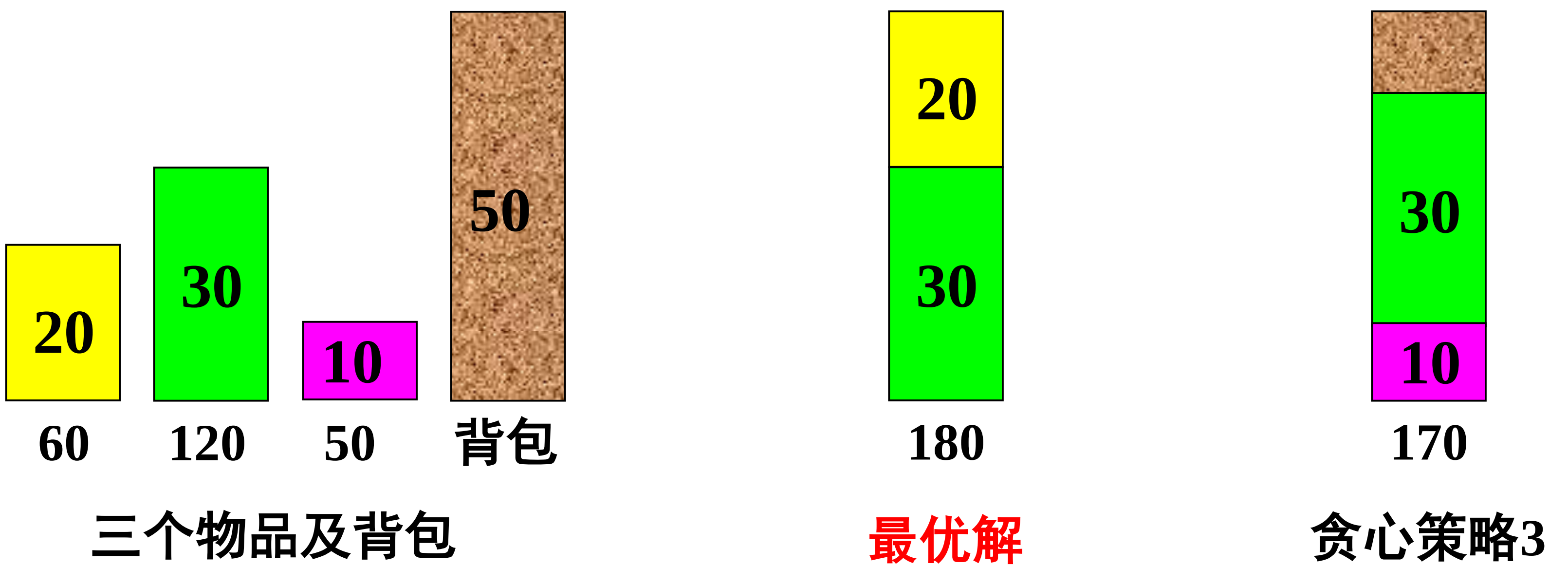


背包问题的贪心法

- 应用第3种贪心策略，每次从物品集合中选择性价比最高的物品，如果其重量小于背包容量，就可以把它装入，并将背包容量减去该物品的重量。
- 然后就面临一个最优子问题——同样是背包问题，只不过背包容量减少了，物品集合减少了。
- 因此背包问题具有最优子结构性质。

背包问题的贪心法

- **0-1背包问题实例：**
  - 有3个物品，其重量分别是{20, 30, 10}，价值分别为{60, 120, 50}，背包的容量为50。采用第3种贪心策略装入背包的物品和获得的价值如图所示。



背包问题的贪心法

- **结论：**
  - 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，无法保证最终能将背包装满，部分闲置的背包空间使每千克背包空间的价值降低了。
  - 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，然后再做出最好选择。
  - 由此可导出许多互相重叠的子问题。这正是该问题可以用动态规划算法求解的另一重要特征。动态规划算法可以有效解决0-1背包问题。

活动选择问题

活动选择问题

问题实例：有11个活动，每个活动的时间表如下

极坐标话剧团（10:00-22:00）
唐仲英爱心社（16:00-21:00）
机器人俱乐部（16:00-20:00）
创行（14:00-18:00）
电子竞技协会（20:00-23:00）
魔方协会（11:00-17:00）
东南大学电视台（08:00-15:00）
辩论协会（09:00-13:00）
大学生心理健康协会（11:00-14:00）
跆拳道协会（14:00-19:00）
华风汉韵文化社（13:00-16:00）

活动选择问题

- **输入：** S={1,2,...,n}为n项活动的集合，s_i，f_i 分别为活动 i 的开始和结束时间。

  活动 i 与 j 相容 $\Leftrightarrow$ $s_i \geq f_j$或$s_j \geq f_i$

- **输出：** 最大的两两相容的活动集A
- **实例：**

i	1	2	3	4	5	6	7	8	9	10
s_i	1	3	2	5	4	5	6	8	8	2
f_i	4	5	6	7	9	9	10	11	12	13

解： A={1,4,8}

贪心算法

挑选过程是多步判断过程，每步依据某种“短视”的策略进行活动的选择，选择时候需要满足相容性条件。

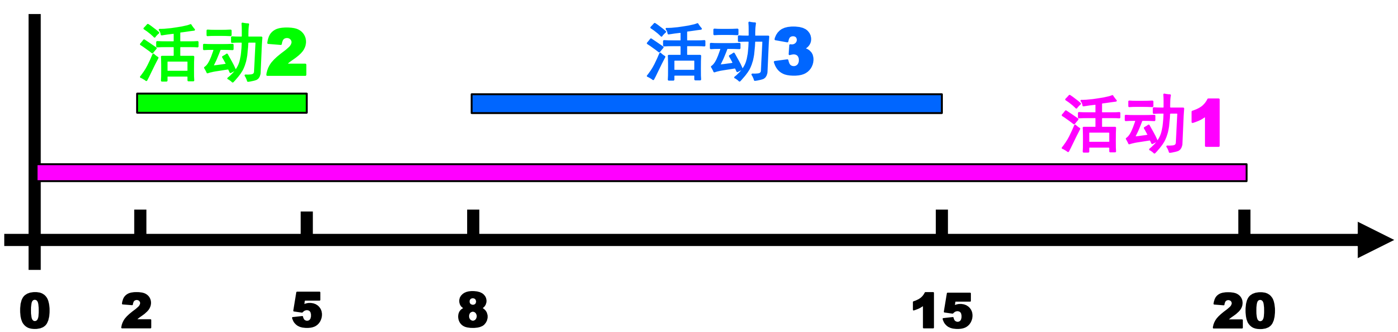
- 策略1：开始时间早的优先
  - 排序使 $s_1 \leqslant s_2 \leqslant ... \leqslant s_n$，从前往后挑选
- 策略2：占用时间少的优先
  - 排序使 $f_1 - s_1 \leqslant f_2 - s_2 \leqslant ... \leqslant f_n - s_n$，从前往后挑选
- 策略3：结束时间早的优先
  - 排序使 $f_1 \leqslant f_2 \leqslant ... \leqslant f_n$，从前往后挑选

策略1的反例

- 策略1：开始早的优先

- 反例：S={1,2,3}

  $s_1=0, f_1=20, s_2=2, f_2=5, s_3=8, f_3=15$

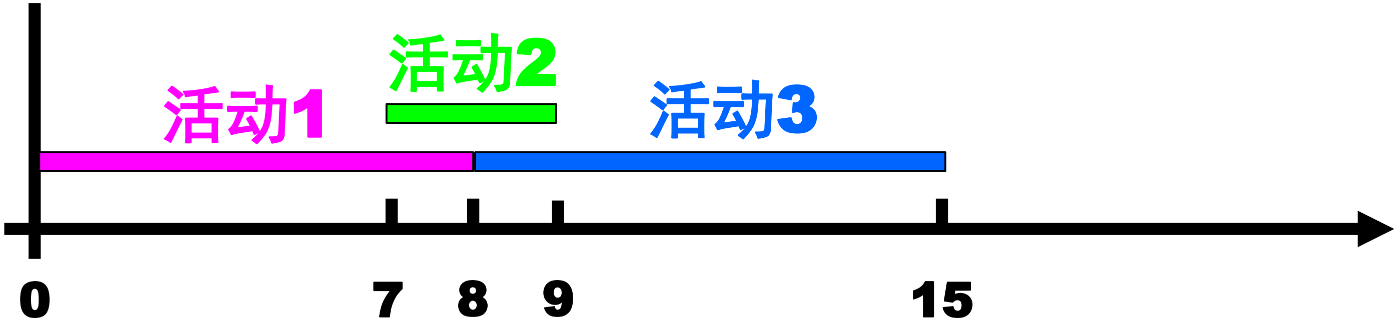


策略2的反例

- 策略2：占时少的优先

- 反例：S={1,2,3}

  $s_1=0, f_1=8, s_2=7, f_2=9, s_3=8, f_3=15$



策略3伪码

- 算法GreedySelect
- 输入：活动集S, s_i, f_i, i =1,2,...,n, $f_1 \leqslant f_2 \leqslant ... \leqslant f_n$
- 输出：A⊆S，选中的活动子集

```
n ← length[S]
A ←{1}
j ← 1
for i ← 2 to n do
    if s_i ≥ f_j
    then A ← A∪{ i }
         j ← i
return A
```

已选入的最后标号

判断相容性

完成时间 $t = \max \{ f_k : k \in A \}$

运行实例

- **输入：S={1,2, ..., 10}**

i	1	2	3	4	5	6	7	8	9	10
s_i	1	3	0	5	3	5	6	8	8	2
f_i	4	5	6	7	8	9	10	11	12	13

解：A={1,4,8}，完成时间 $t = 11$

时间复杂度：

$$O(n\log n)+O(n)=O(n\log n)$$

问题：如何证明该算法对所有实例都得到正确的解？

贪心算法的特点

- 设计要素：
  - 贪心法适用于组合优化问题
  - 求解过程是多步判断过程，最终的判断序列对应于问题的最优解
  - 依据某种“短视”的贪心选择性质判断，性质好坏决定算法的成败
  - 贪心法如何进行正确性证明？
  - 证明贪心法不正确的技巧：反证法

贪心优势：算法简单，时空复杂度低

贪心法正确性证明

- 数学归纳法
  - 例：证明对于任何自然数n，

    $1+2+\ldots+n=n(n+1)/2$

  - 证：$n=1$，左边$=1$，右边$=1\times(1+1)/2=1$
    假设对于任意自然数n等式成立，则
    $1+2+\ldots+(n+1)$
    $=(1+2+\ldots+n)+(n+1)$
    $=n(n+1)/2+(n+1)$
    $=(n+1)(n/2+1)$
    $=(n+1)(n+2)/2$
    $=(n+1)((n+1)+1)/2$

代入归纳假设

第一数学归纳法

适合证明涉及自然数的**命题**P(n)

- **归纳基础：** 证明P(1)为真（或P(0)为真）
- **归纳步骤：** 若对所有n有P(n)为真，证明P(n+1)为真

$$\forall n,\ P(n) \rightarrow P(n+1)$$

$$P(1)$$

$$n=1,\ P(1) \Rightarrow P(2)$$

$$n=2,\ P(2) \Rightarrow P(3)$$

$$n=3,\ P(3) \Rightarrow P(4)$$

……

第二数学归纳法

适合证明涉及自然数的**命题**P(n)

- **归纳基础：** 证明P(1)为真（或P(0)为真）
- **归纳步骤：** 若对所有小于n的k有P(k)为真，证明P(n)为真

$$\forall k\ (k<n,\ P(n)) \rightarrow P(n)$$

$$P(1)$$

$n=2$, $P(1) \Rightarrow P(2)$

$n=3$, $P(1) \wedge P(2) \Rightarrow P(3)$

$n=4$, $P(1) \wedge P(2) \wedge P(3) \Rightarrow P(4)$

……

两种归纳法的区别

- 归纳基础一样　　都是P(1/0)为真
- 归纳步骤不同

证明逻辑

- 归纳法1：$P(1) \Rightarrow P(2) \Rightarrow P(3) \Rightarrow \ldots$
- 归纳法2：

$$\left.\begin{array}{r} P(1) \\ P(2) \\ \left.\begin{array}{r} P(1) \\ P(1) \Rightarrow P(2) \end{array}\right\} \Rightarrow P(3) \end{array}\right\} \Rightarrow P(4) \Rightarrow \ldots$$

贪心算法正确性归纳证明

- **证明步骤**
  1. 叙述一个有关自然数n的命题，该命题断定该贪心策略的执行最终将导致最优解。其中自然数n可以代表步数或问题规模
  2. 证明命题对所有的自然数为真
     - 2.1 归纳基础（从最小实例规模开始）
     - 2.2 归纳步骤（第一或第二数学归纳法）

活动选择贪心策略3伪码

- 算法GreedySelect
- 输入：活动集S, s_i, f_i, i =1,2,...,n, $f_1 \leqslant f_2 \leqslant \ldots \leqslant f_n$
- 输出：A⊆S，选中的活动子集

```
n ← length[S]
A ←{1}
j ← 1
for i ← 2 to n do
    if s_i ≥ f_j
    then A ← A∪{ i }
         j ← i
return A
```

已选入的最后标号

判断相容性

完成时间 $t = \max \{ f_k : k \in A \}$

活动选择算法的命题

1. 命题：

算法GreedySelect执行到第 k 步，选择 k 项活动

$$i_1=1, i_2, \ldots, i_k$$

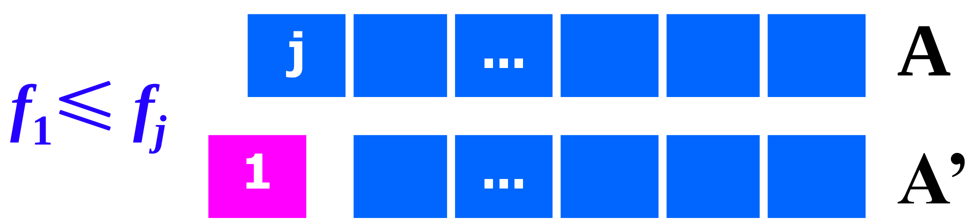
则存在最优解 A 包含活动 $i_1=1, i_2, \ldots, i_k$。

根据上述命题：对于任何 k，算法前 k 步的选择都将导致最优解，至多到第 n 步将得到问题实例的最优解。

归纳法证明

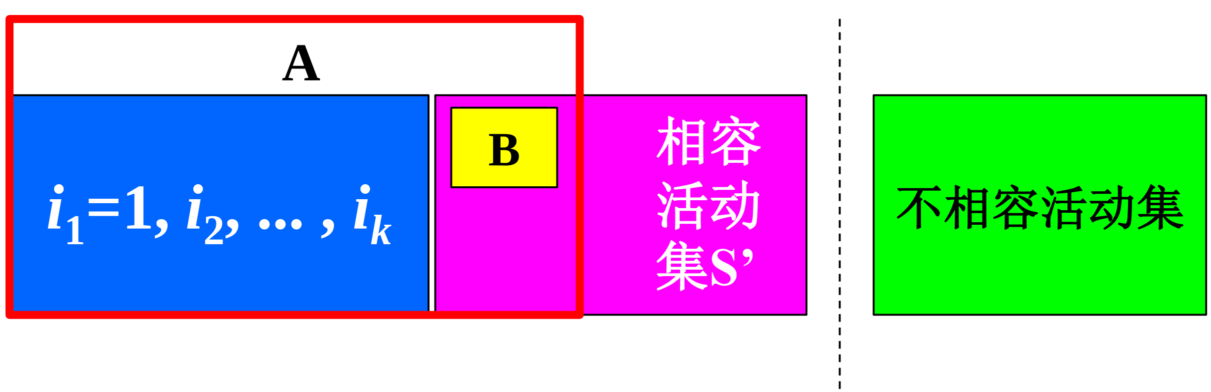
令S={1,2,...,n}是活动集，且$f_1 \leqslant f_2 \leqslant ... \leqslant f_n$

- 2.1 归纳基础：k=1, 证明存在最优解包含活动1
- 证明：
  - 任取最优解A，A中活动按截止时间递增排序。如果A的第一个活动为 j，且 $j \neq 1$，用1替换A的活动 j 得到解A'，即A'=(A-{ j })∪{1}
  - 由于$f_1 \leqslant f_j$，A'也是最优解，且含有1。



归纳法证明

- 2.2 归纳步骤：假设命题对k为真，证明对k+1也为真
- 证明：
  - 算法执行到第k步，选择了活动 $i_1=1, i_2, \ldots, i_k$，根据归纳假设存在最优解A包含 $i_1=1, i_2, \ldots, i_k$，且A中剩下活动B选自集合S’
  - S’$=\{ i \mid i \in S, s_i \geqslant f_k\}$，A$=\{i_1=1, i_2, \ldots, i_k\} \cup$B

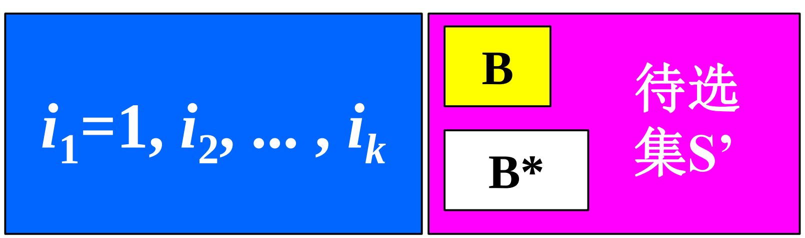


归纳步骤(cont.)

- **B是S’的最优解。**
  - 若不然，S’的最优解为B*，B*的活动比B多，那么

$$B^* \cup \{i_1=1, i_2, \ldots, i_k\}$$

是S的最优解，且比A的活动多，与A的最优性矛盾！



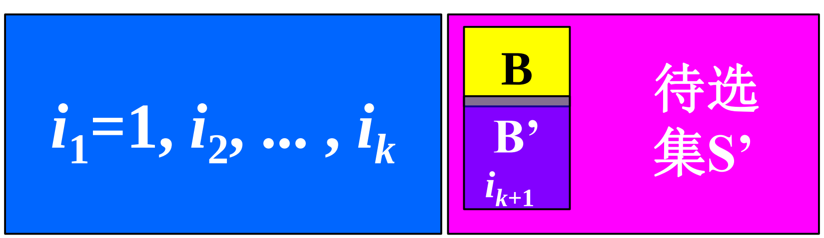
归纳步骤(cont.)

- 将S’看成子问题，根据归纳基础，**存在S’的最优解B’有S’中的第一个活动 i_{k+1}**，且 |B’|=|B|，于是：

$$\{i_1=1, i_2, \ldots, i_k\} \cup B'$$

$$=\{i_1=1, i_2, \ldots, i_k, i_{k+1}\} \cup (B' - i_{k+1})$$

也是原问题的最优解。



证明方法小结

- 贪心法正确性证明方法：数学归纳法
  - 第一数学归纳法
  - 第二数学归纳法
- 活动选择问题的贪心法证明：
  - 叙述一个涉及步数的算法正确性命题
  - 证明归纳基础
  - 证明归纳步骤

算法分析与设计

Analysis and Design of Algorithm

Lesson 11

要点回顾

- 贪心算法正确性归纳证明
  - 叙述一个有关自然数n的命题，该命题断定该贪心策略的执行最终将导致最优解。其中自然数n可以代表步数或问题规模
  - 证明命题对所有的自然数为真
    - 归纳基础（从最小实例规模开始）
    - 归纳步骤（第一或第二数学归纳法）
- 几个实例：
  - 背包问题
  - 活动安排问题（最优性证明）

最优装载问题

- **问题：** 有n个集装箱1,2,...,n装上轮船，集装箱 i 的重量w_i，轮船装载重量限制为C，无体积限制。问如何装，使得船上的集装箱的**数目最多**？不妨设每个箱子的重量$w_i \leqslant C$

最优装载问题

- 该问题是**0-1背包问题的子问题**。集装箱相当于物品，物品的重量是w_i，价值v_i都等于1，轮船载重限制C相当于背包的重量限制b。

问题建模

- 设 $\langle x_1,x_2,\ldots,x_n\rangle$ 是解向量，其中 x_i=0,1，x_i=1 当且仅当第 i 个集装箱上船

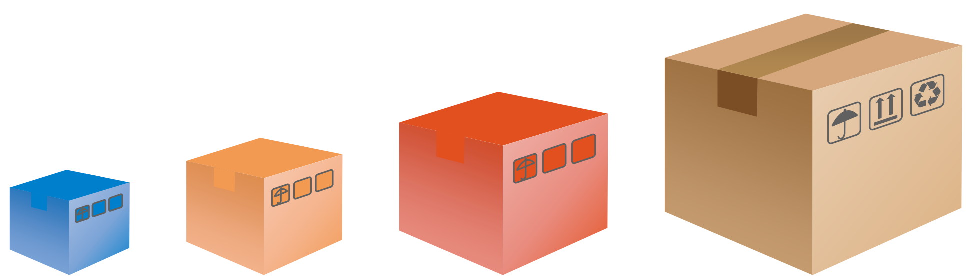
目标函数 $$\max\sum_{i=1}^{n}x_i$$

约束条件 $$\sum_{i=1}^{n}w_i x_i \le C$$

$$x_i=0,1 \quad i=1,2,\ldots,n$$

算法设计

- 贪心策略G：轻者优先
- 算法设计：
  - 将集装箱排序，使得 $w_1 \leqslant w_2 \leqslant ... \leqslant w_n$
  - 按照标号从小到大装箱，直到装入下一个箱子将使得集装箱总重超过轮船的装载重量限制，则停止。



正确性证明思路

- 命题：对于装载问题任何规模为*n*的输入实例，贪心算法G得到最优解。
- 设集装箱从轻到重记为1,2,..., *n*
- 归纳基础　证明对任何只含1个箱子的输入实例，贪心法得到最优解。显然正确！
- 归纳步骤　证明：假设对于任何*n*个箱子的输入实例贪心法都能得到最优解，那么对于任何*n*+1个箱子的输入实例贪心法也得到最优解。

二元前缀码及其应用

- 二元前缀码是广泛用于数据**文件压缩**的编码方法，其使用字符在文件中出现的频率表来建立一个用0,1串表示各个字符的最优表示方式。

WinZip

Google

Draco 3D compressor

facebook

Zstandard

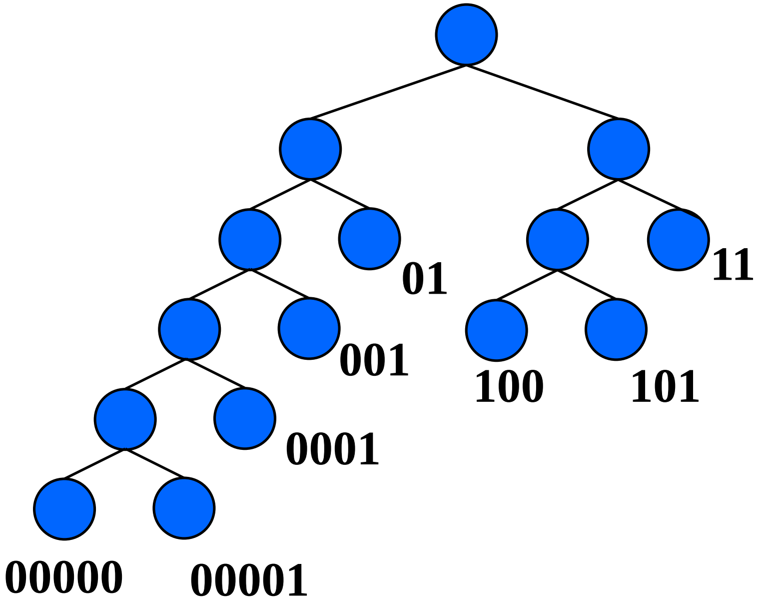
最优前缀码

- 二元前缀码
  - 用0-1字符串作为代码表示字符，要求任何字符的代码都不能作为其他字符代码的前缀
  - 非前缀码的例子

    a: 001, b: 00, c:010, d:01
  - 解码的歧义，例如字符串0100001
    - 解码1： 01, 00, 001 d, b, a
    - 解码2： 010, 00, 01 c, b, d

前缀码的二叉树表示

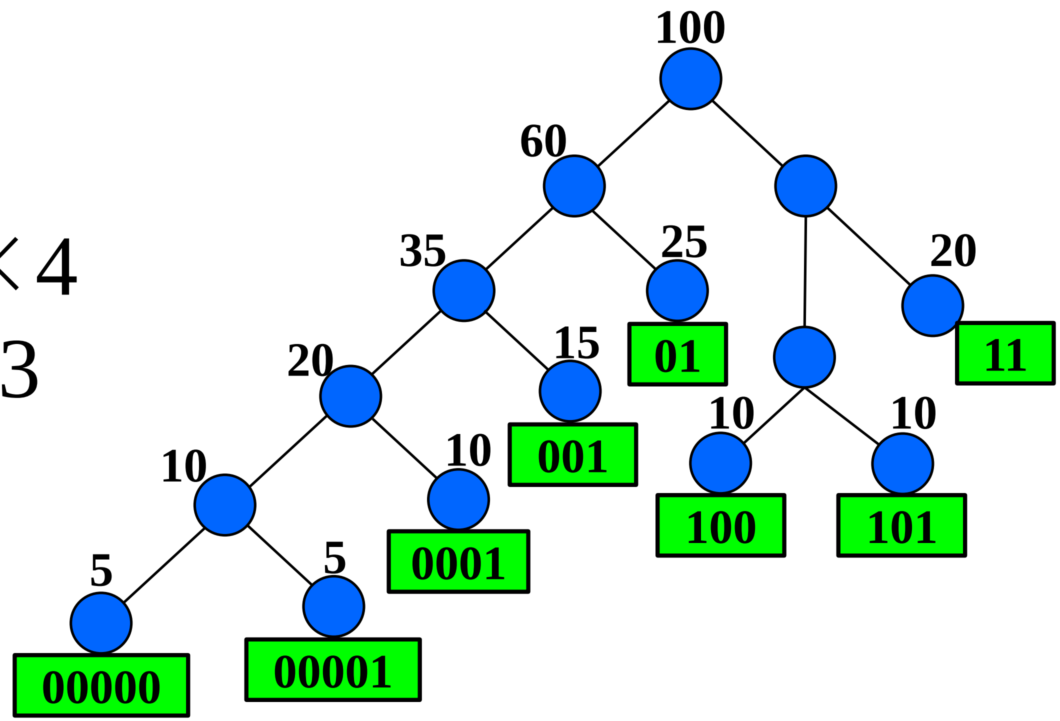
- 前缀码:
  {00000, 00001, 0001, 001, 01, 100, 101, 11}
- 构造树:
  - 0-左子树
  - 1-右子树
  - 码对应一片树叶
  - 最大位数为树深



平均传输位数

$$B=\sum_{i=1}^{n} f(x_i)d(x_i)$$

$$\begin{aligned}B = [&(5+5)\times 5+10\times 4\\ &+(15+10+10)\times 3\\ &+(25+20)\times 2]\\ &\div 100\\ = \ & \mathbf{2.85}\end{aligned}$$



问题：给定字符集C={$x_1,x_2,\ldots,x_n$}和每个字符的频率$f(x_i)$，$i=1,2,\ldots,n$。求关于C的一个最优前缀码（平均传输位数最小）。

哈夫曼树算法伪码

- 算法 Huffman(C)
- 输入：C=$\{x_1,x_2,\ldots,x_n\}$, $f(x_i)$, i=1,2,...,n
- 输出：Q //队列

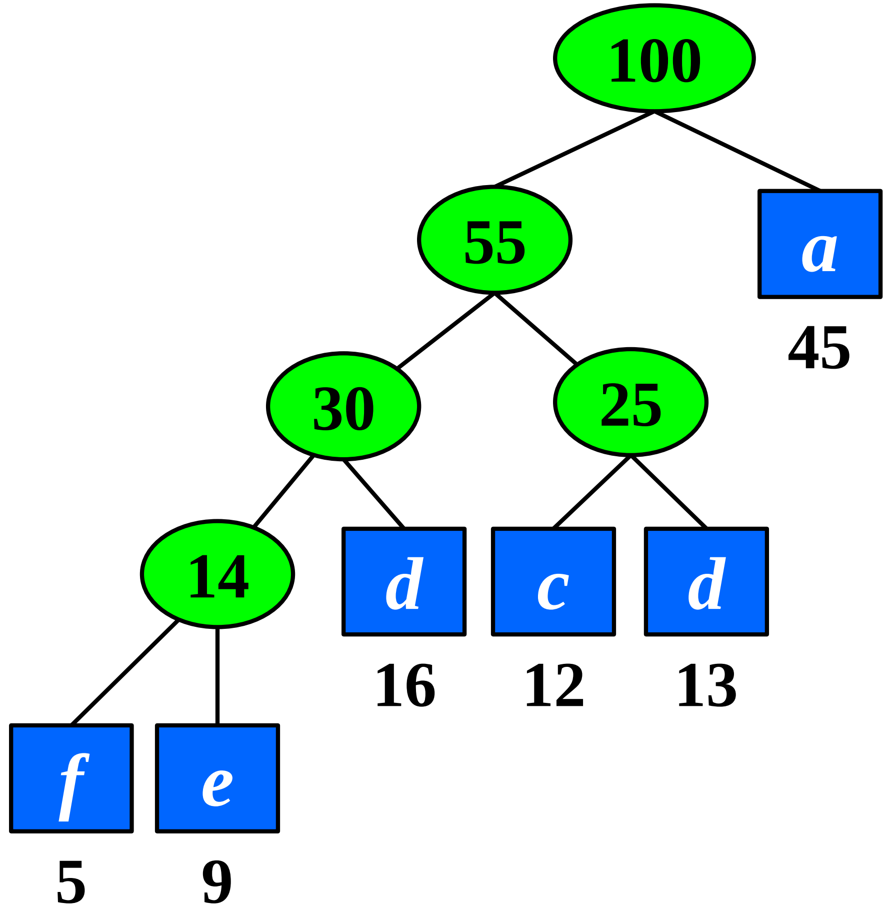
1. $n \leftarrow |C|$
2. Q ← C //频率递增队列Q
3. **for** $i \leftarrow 1$ **to** n-1 **do**
4. $z \leftarrow$ Allocate-Node() //生成结点z
5. $z.left \leftarrow$ Q中最小元 //最小作z左儿子
6. $z.right \leftarrow$ Q中最小元 //最小作z右儿子
7. $f(z) \leftarrow f(x)+f(y)$
8.   Insert(Q, z) //将z插入Q
9. **return** Q

实例

输入： a:45, b:13, c:12, d:16, e:9, f:5

编码：

- $f \rightarrow 0000$
- $e \rightarrow 0001$
- $d \rightarrow 001$
- $c \rightarrow 010$
- $b \rightarrow 011$
- $a \rightarrow 1$



平均位数？

图问题中的贪心算法

最小生成树

无向连通带权图

$$G=(V,E,W)$$

其中$w(e)\in W$是边e的权值。

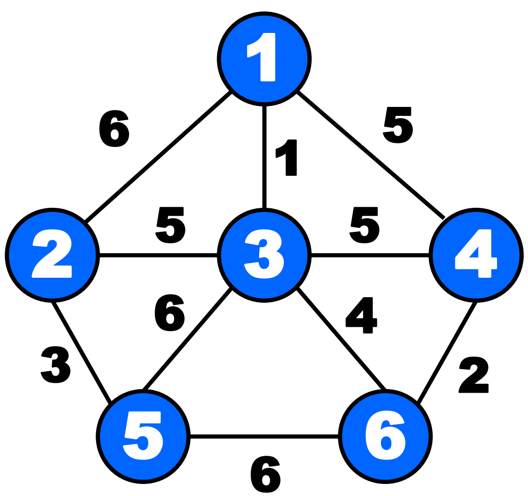
G的一棵生成树T是包含了G所有顶点的树，树中各边的权值之和W(T)称为树的权，具有最小权的生成树称为G的最小生成树。

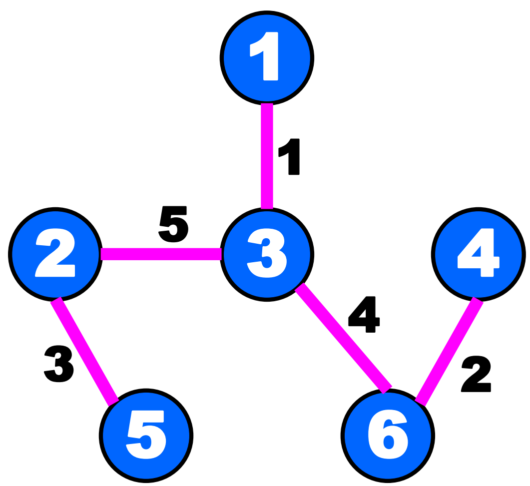
最小生成树的实例

G=(V, E, W), V={1,2,3,4,5,6}, W如图所示。

E={{1,2},{1,3},{1,4},{2,3},{2,5},

{3,4},{3,5},{3,6},{4,6},{5,6}}

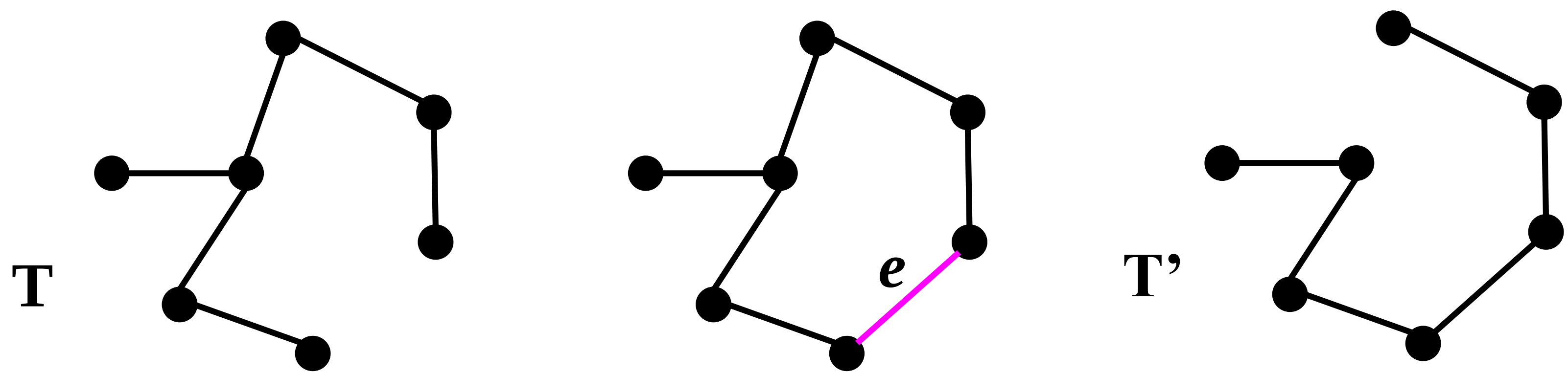




生成树的性质

命题1：设G是n阶连通图，那么

1. T是G的生成树当且仅当T无圈且有n-1条边；
2. 如果T是G的生成树，$e \notin$T，那么T∪{e}含有一个圈C(回路)
3. 去掉圈C的任意一条边，就得到G的另外一棵生成树T’



生成树性质的应用

- 算法步骤：选择边。
  约束条件：不形成回路
  截止条件：边数达到n-1

- 改进生成树T的方法
  在T中加一条非树边e，形成回路C，在C中去掉一条树边e'，形成一棵新的生成树T'

$$W(T')-W(T)=W(e)-W(e')$$

  若$W(e) \leqslant W(e')$，则$W(T') \leqslant W(T)$

求最小生成树

- **问题：**
  给定连通带权图G=(V,E,W), W(*e*)∈W是边*e*的权。求G的一棵最小生成树。

- **贪心法：**
  - **Prim算法**
  - **Kruskal算法**

生成树在网络中有着重要应用！

Prim算法

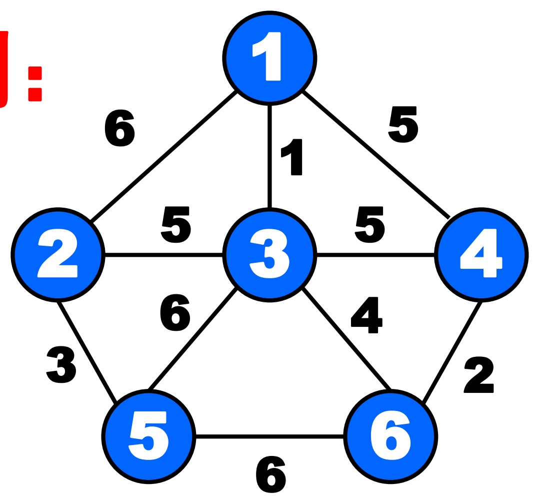
- **设计思想**
  - **输入：图G=(V,E,W), V={1,2,...,*n*}**
  - **输出：最小生成树T**
  - **步骤：**
    - **初始S={1};**
    - **选择连接S与V-S集合的最短边*e*={*i*, *j*}，其中*i*∈S, *j*∈V-S。将*e*加入树T，*j*加入S；**
    - **继续执行上述过程，直到S=V为止。**

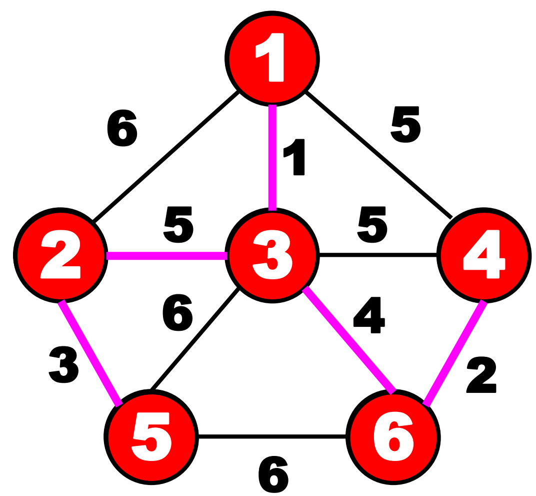
Prim算法伪码

- 算法Prim(G,E,W)

```
S ← {1}
while V–S≠ ∅ do
    从V-S中选择j使得j到S中顶点的边权最小
    S ← S∪{ j }
```

- 实例：





算法复杂度

- 算法步骤执行O(n)次
- 每次执行O(n)时间：
  - 找连接S与V-S的最短边
- 算法时间：$T(n)=O(n^2)$

Kruskal算法

- 设计思想
  - 输入：图G=(V,E,W), V={1,2,...,*n*}
  - 输出：最小生成树T
  - 步骤：
    - 按照长度从小到大对边进行排序；
    - 依次考察当前最短边*e*，如果*e*与T的边不构成回路，则把*e*加入到树T，否则跳过*e*。直到选择了*n*-1条边为止。

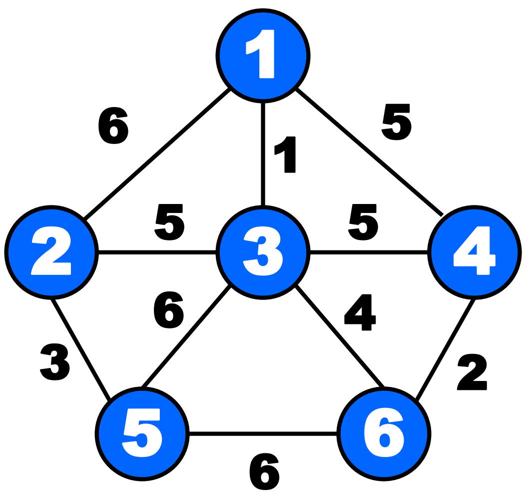
Kruskal算法伪码

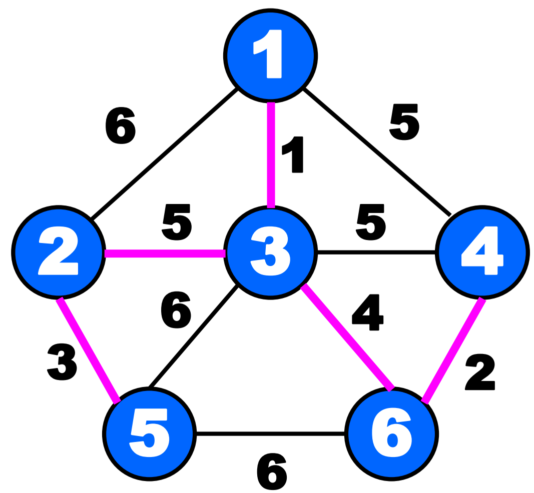
- **输入：图G** //顶点数n，边数m
- **输出：最小生成树T**

1. 权从小到大排序E的边，$E=\{e_1,e_2,\ldots,e_m\}$
2. $T \leftarrow \emptyset$
3. **repeat**
4. $\quad e \leftarrow$ E中的最短边
5. $\quad$ **if** e的两端点不在同一连通分支
6. $\quad$ **then** $T \leftarrow T \cup \{e\}$
7. $\quad E \leftarrow E - \{e\}$
8. **until** T包含了n-1条边

Kruskal算法

- **实例**





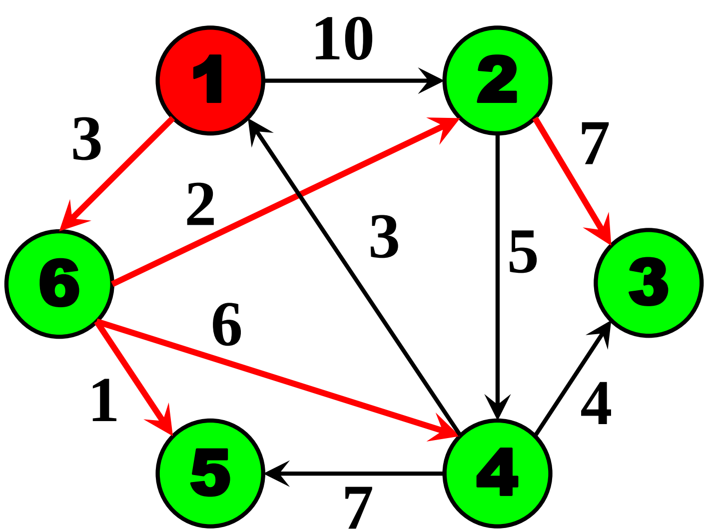
思考

- 基于上述Prim和Kruskal算法思想
- 对于边数相对较多（即比较接近于完全图）的无向连通带权图，比较适合于用哪种方法求解?
- 对边数较少的无向连通带权图有较高效率的又是哪一种算法?
- 请分析原因

单源最短路径问题

给定**带权有向图**G=(V,E,W)，每条边e=<i, j>的权$w(e)$为非负实数，表示i到j的距离。**源点**$s \in$V。

求：从s出发到达其他结点的最短路径。



- 源点：1
- 1→6→2: *short*[2]=5
- 1→6→2→3 : *short*[3]=12
- 1→6→4 : *short*[4]=9
- 1→6→5 : *short*[5]=4
- 1→6 : *short*[6]=3

Dijkstra算法有关概念

- $x\in S \Leftrightarrow x\in V$且从$s$到$x$的最短路径已经找到
- 初始：S={s}，S=V时算法结束
- 从s到u相对于S的最短路径：从s到u且仅经过S中顶点的最短路径
- *dist*[*u*]：从s到u相对S的最短路径的长度
- *short*[*u*]：从s到u的最短路径的长度
- *dist*[*u*]≥*short*[*u*]

算法的设计思想

- **输入：** 有向图G=(V,E,W), V={1,2,...,n}, s=1
- **输出：** 从s到每个顶点的最短路径
- **步骤：**
  1. 初始S={1}；
  2. 对于i∈V-S，计算1到i的相对S的最短路径，长度记为$dist[i]$；
  3. 选择V-S中$dist$值最小的j，将j加入到S，修改V-S中的顶点的$dist$值；
  4. 继续上述过程，直到S=V为止。

算法伪码

- 算法Dijkstra

1. S←{*s*}
2. *dist*[*s*]←0
3. **for** *i*∈V-{*s*} **do**
4.   *dist*[*i*]←*w*(*s*,*i*) //*s*到*i*没边，*w*(*s*,*i*) =∞
5. **while** V-S ≠ ∅ **do**
6.   从V-S取相对S的最短路径顶点 *j*
7.   S←S∪{ *j* }
8.   **for** *i*∈V-S **do**
9.   **if** *dist*[*j*]+*w*(*i*,*j*)<*dist*[*i*]
10.   **then** *dist*[*i*]←*dist*[*j*]+*w*(*i*,*j*)

只更新与*j*相邻的顶点

更新*dist*值

运行实例

- 输入：G=(V,E,W), V={1,2,3,4,5,6}, 源点1

S={1}

dist[1]=0

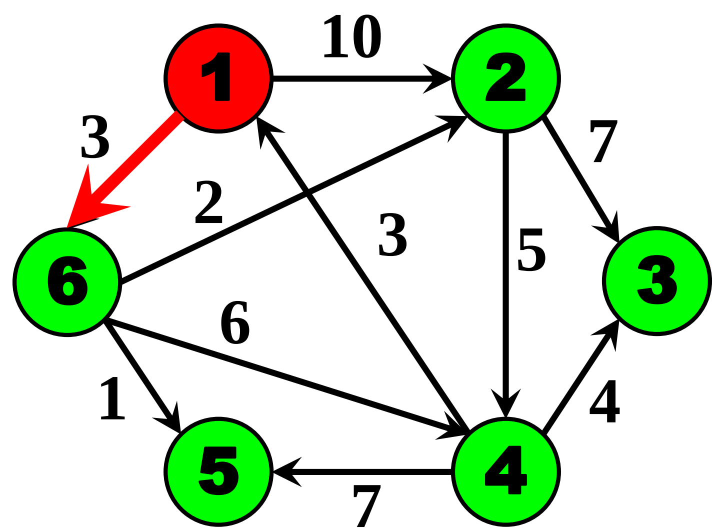
dist[2]=10

$\boxed{dist[6]=3}$

dist[3]=∞

dist[4]=∞

dist[5]=∞



运行实例

- 输入：G=(V,E,W), V={1,2,3,4,5,6}, 源点1

S={1,6}

dist[1]=0

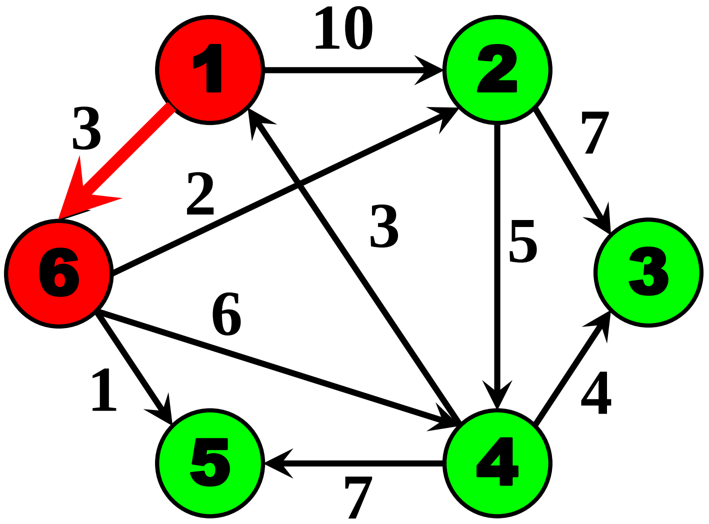
dist[6]=3

dist[2]=5

dist[4]=9

$\boxed{dist[5]=4}$

dist[3]=∞



运行实例

- **输入：G=(V,E,W), V={1,2,3,4,5,6}, 源点1**

S={1,6,5}

***dist*[1]=0**

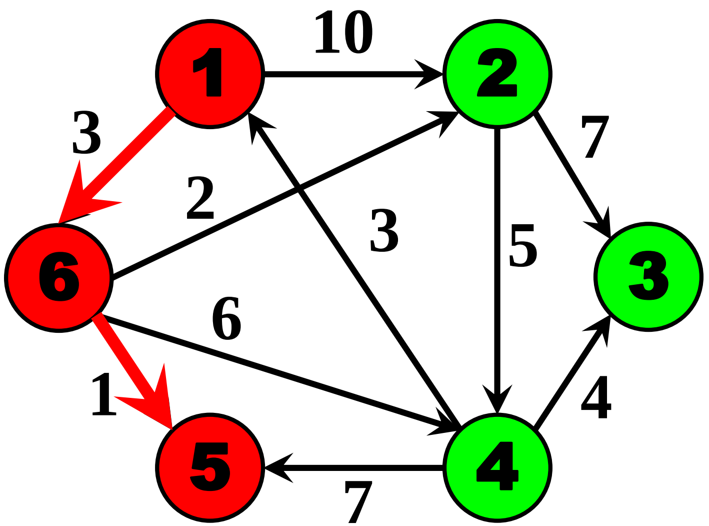
***dist*[6]=3**

***dist*[5]=4**

$\boxed{\textbf{\textit{dist}[2]=5}}$

***dist*[4]=9**

***dist*[3]=∞**



运行实例

- 输入：G=(V,E,W), V={1,2,3,4,5,6}, 源点1

S={1,6,5,2}

dist[1]=0

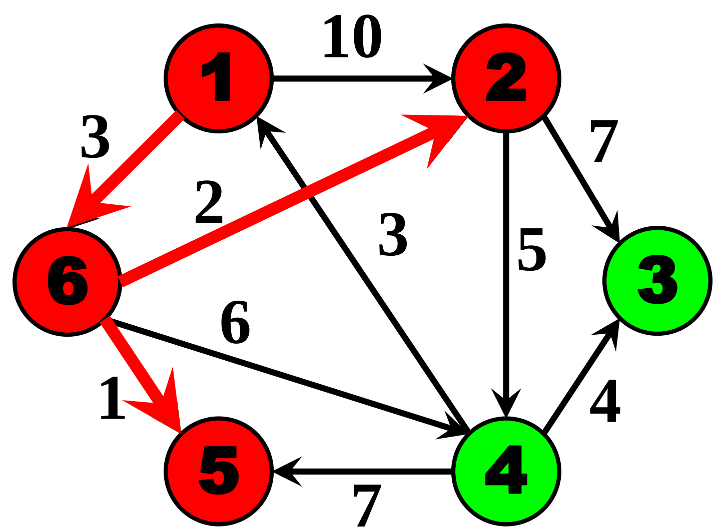
dist[6]=3

dist[5]=4

dist[2]=5

dist[4]=9

dist[3]=12



运行实例

- **输入：G=(V,E,W), V={1,2,3,4,5,6}, 源点1**

S={1,6,5,2,4}

dist[1]=0

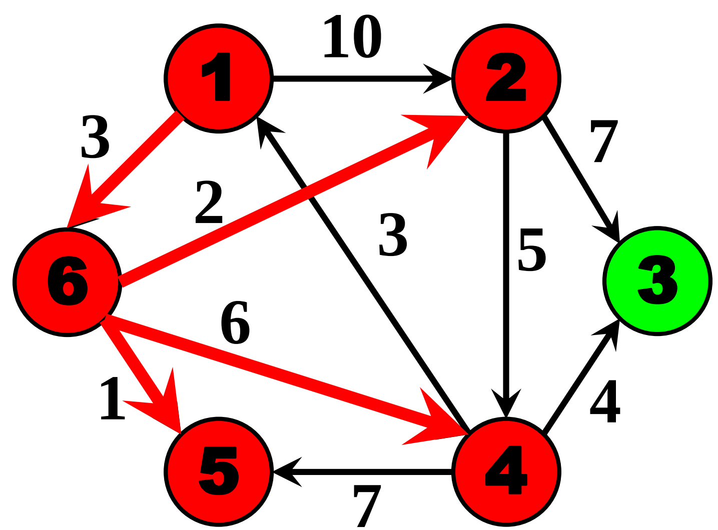
dist[6]=3

dist[5]=4

dist[2]=5

dist[4]=9

$\boxed{dist[3]=12}$



运行实例

- 输入：G=(V,E,W), V={1,2,3,4,5,6}, 源点1

S={1,6,5,2,4,3}

dist[1]=0

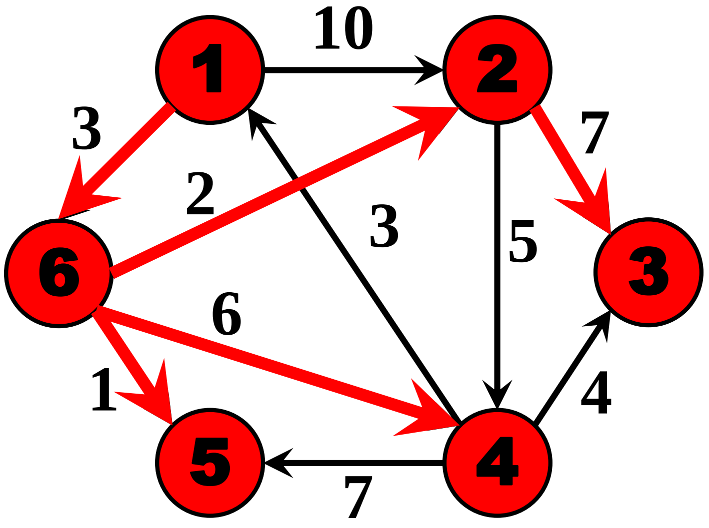
dist[6]=3

dist[5]=4

dist[2]=5

dist[4]=9

dist[3]=12



找到了问题的解！

时间复杂度分析

- 时间复杂度：
  - 算法进行n-1步
  - 每步挑选1个具有最小*dist*函数值的结点进入到S，需要O(n)时间
  - →O(n^2)
- 选用基于堆实现的优先队列的数据结构，可以将算法时间复杂度降低到O($m\log n$)

算法分析与设计

Analysis and Design of Algorithm

Lesson 12

要点回顾

- 贪心算法正确性归纳证明
  - 叙述一个有关自然数n的命题，该命题断定该贪心策略的执行最终将导致最优解。其中自然数n可以代表步数或问题规模
  - 证明命题对所有的自然数为真
    - 归纳基础（从最小实例规模开始）
    - 归纳步骤（第一或第二数学归纳法）
- 几个实例：
  - 最优装载（0-1背包问题的子问题）
  - 最优前缀码（Huffman树）
  - 最小生成树（Prim和Kruskal算法）
  - 单源最短路径（Dijkstra算法）

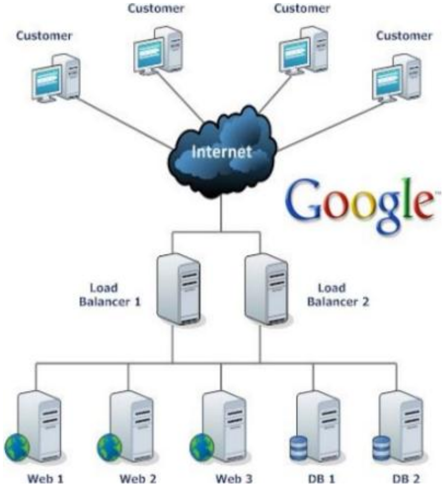
贪心算法求解NP完全问题

回顾：NP完全问题

- P问题的集合
  - 所有可以在多项式时间内求解的判定问题。
- NP问题的集合
  - 可用多项式时间的非确定性算法来进行判定或求解的问题。
- NP完全问题的集合
  - NP中某些问题不确定能否在多项式时间内求解。
  - 这些问题中，任何一个如果存在多项式时间的算法，那么所有NP问题都是多项式时间可解。
- P=NP？
  - 七个“千僖年数学难题”之一。

多机调度问题及其应用

- 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，每个作业运行时间为t_i，要求在尽可能短的时间内由m台相同的机器加工处理完成。



计算机集群调度

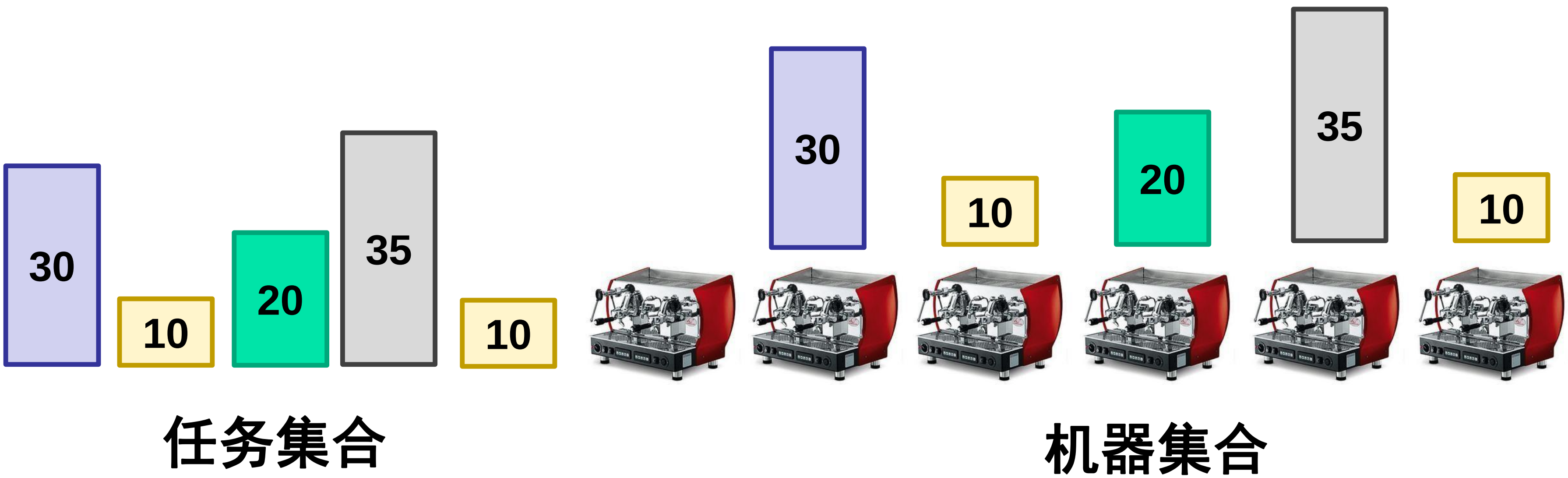
工厂机床调度

多机调度问题及其应用

- **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，每个作业运行时间为t_i，要求在尽可能短的时间内由m台相同的机器加工处理完成。
- 多机调度问题是**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。

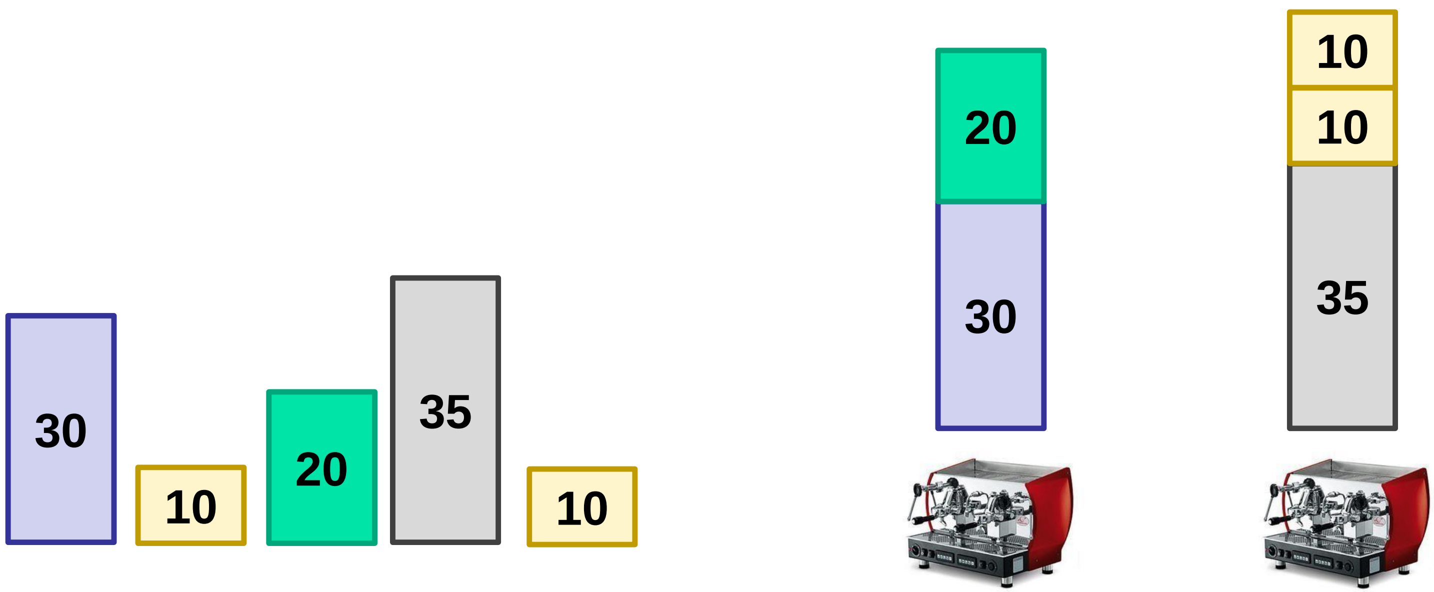
多机调度问题—贪心策略

- 当机器数$m \geqslant$任务数n时
  - 每台机器放一个任务



多机调度问题—贪心策略

- 当机器数m<任务数n时
  - 优先放长任务
  - 优先选择负载最轻机器



任务集合　　　　机器集合

第四章小结

- 贪心法适用于组合优化问题
- 判断依据某种“短视”贪心选择性质，性质的好坏决定了算法的成败。贪心性质往往依赖于直觉或者经验
- 贪心法正确性证明
  - 数学归纳法（对算法步骤或者问题规模归纳）
  - 证明贪心法不一定正确—找到一个反例

贪心法小结(cont.)

- 对于某些不能保证对所有的实例都得到最优解的贪心算法，可以求得近似解，可做参数化分析或者误差分析（高级算法课程）。
- 贪心算法的优势：
  - 算法简单
  - 时空复杂度低
- 几个著名的贪心算法
  - 最小生成树的Prim算法和Kruskal算法
  - 单源最短路径的Dijkstra算法

归纳步骤证明思路

01 $N=\{1, 2, \ldots, n, n+1\}$, $w_1 \leqslant w_2 \leqslant \ldots \leqslant w_{n+1}$

02 去掉箱子1，令$C'=C-w_1$
得到规模为n的输入$N'=\{2,3,\ldots,n,n+1\}$

03 关于输入N’和C’的最优解I’

04 在I’加入箱子1，得到I

05 证明I是关于输入N的最优解

正确性证明

- 假设对于n个集装箱的输入，贪心法都可以得到最优解，考虑输入

$$N=\{ 1, 2 , \ldots , n , n+1 \}$$

其中$w_1 \leqslant w_2 \leqslant \ldots \leqslant w_{n+1}$

- 由归纳假设，对于

$$N'=\{2,3,\ldots,n,n+1\}，C'=C- w_1$$

- 贪心法得到最优解I'。令

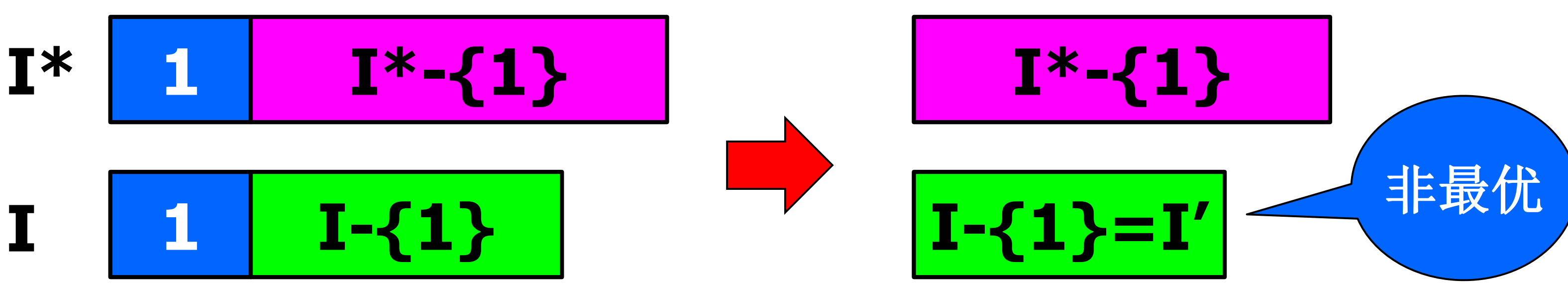
$$I = I' \cup \{1\}$$

正确性证明

- **I(算法解)是关于N的最优解**
  - 若不然，存在包含1的关于N的最优解I*(如果I*中没有1，用1替换I*中的第一个元素得到的解也是最优解)，且|I*|>|I|；那么I*-{1}是N’和C’的解且

    |I*-{1}|>|I-{1}|=|I’|

    与I’是关于N’和C’的最优解矛盾。



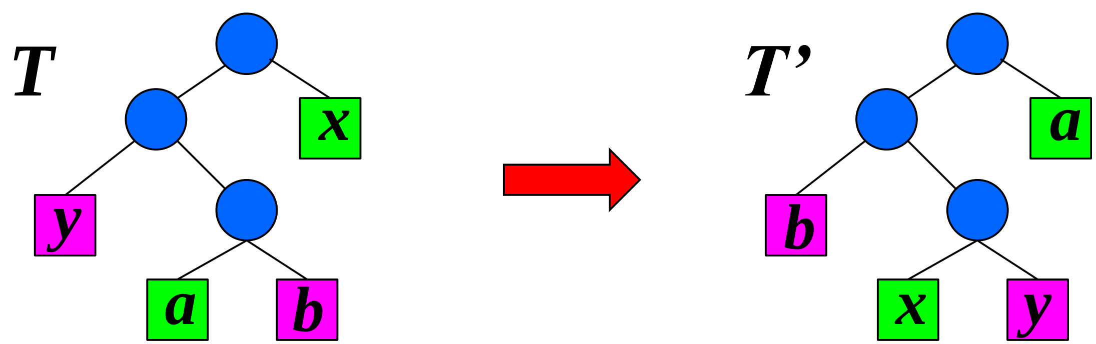
小结

- 装载问题是0-1背包问题的子问题（每件物品重量为1），NP难的问题存在多项式时间可解的子问题。
- 贪心法证明：对规模进行归纳

最优前缀码性质

引理1： C是字符集，$\forall c \in C$, $f(c)$为频率，$x,y \in C$, $f(x)$, $f(y)$频率最小，那么存在最优二元前缀码，使得x, y码字等长且仅在最后一位不同。

$f(x) \leqslant f(a)$
$f(y) \leqslant f(b)$



$$B(T)-B(T')=\sum_{i \in C} f(i)d_T(i)-\sum_{i \in C} f(i)d_{T'}(i) \geq 0$$

其中$d_T(i)$为i在T中的层数(i到根的距离)

最优前缀码性质

引理2：设T是二元前缀码的二叉树，$\forall x, y \in T$，x, y是树叶兄弟，z是x, y的父亲，令

$$T' = T - \{x, y\}$$

且令 z 的频率

$$f(z) = f(x) + f(y)$$

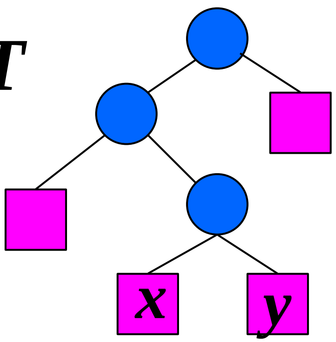
T'是对应二元前缀码

$$C' = (C - \{x, y\}) \cup \{z\}$$

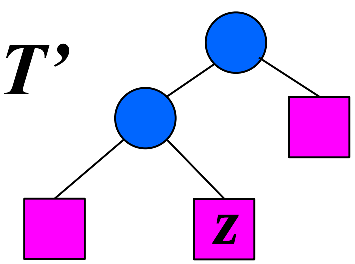
的二叉树，那么

$$B(T) = B(T') + f(x) + f(y)$$

T



T'



引理2证明

证明： $\forall c \in C-\{x, y\}$，有

$d_T(c) = d_{T'}(c) \Rightarrow f(c)d_T(c) = f(c)d_{T'}(c)$

$d_T(x) = d_T(y) = d_{T'}(z)+1$

$$B(T) = \sum_{i \in T} f(i) d_T(i)$$

$$= \sum_{i \in T, i \neq x, y} f(i) d_T(i) + f(x) d_T(x) + f(y) d_T(y)$$

$$= \sum_{i \in T', i \neq z} f(i) d_{T'}(i) + f(z) d_{T'}(z) + (f(x) + f(y))$$

$$= B(T') + f(x) + f(y)$$

算法正确性证明思路

定理　Huffman算法对任意规模为n(n≥2)的字符集C都得到关于C的最优前缀码的二叉树。

归纳基础　证明：对于n=2的字符集，Huffman算法得到最优前缀码。

归纳步骤　证明：假设Huffman算法对于规模为k的字符集都得到最优前缀码，那么对于规模为k+1的字符集也得到最优前缀码。

要点回顾

- 贪心算法正确性归纳证明
  - 叙述一个有关自然数n的**命题**，该命题断定该贪心策略的执行最终将导致最优解。其中自然数n可以代表步数或问题规模
  - 证明命题对所有的自然数为真
    - **归纳基础**（从最小实例规模开始）
    - **归纳步骤**（第一或第二数学归纳法）
- 几个实例：
  - 最优装载（0-1背包问题的子问题）
  - 最优前缀码（Huffman树）
  - 最小生成树（Prim和Kruskal算法）
  - 单源最短路径（开篇）