算法设计与分析

第4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
  （1）最优子结构性质
  （2）贪心选择性质
- 理解贪心算法与动态规划算法的差异
- 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
  （1）活动安排问题；（2）最优装载问题；
  （3）哈夫曼编码；（4）单源最短路径；
  （5）最小生成树；（6）多机调度问题。

贪心算法

顾名思义，贪心算法总是作出在当前看来最好的选择。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。当然，希望贪心算法得到的最终结果也是整体最优的。虽然贪心算法不能对所有问题都得到整体最优解，但对许多问题它能产生整体最优解。如单源最短路经问题，最小生成树问题等。在一些情况下，即使贪心算法不能得到整体最优解，其最终结果却是最优解的很好近似。

4.1 活动安排问题

设有n个活动的集合E={1, 2, ···, n}，其中每个活动都要求使用同一资源，如演讲会场等，而在同一时间内只有一个活动能使用这一资源。每个活动i都有一个要求使用该资源的起始时间si和一个结束时间fi,且si ＜ fi 。如果选择了活动i，则它在半开时间区间[si, fi)内占用资源。若区间[si, fi)与区间[sj, fj)不相交，则称活动i与活动j是相容的。也就是说，当si ⩾ fj 或sj ⩾ fi时，活动i与活动j相容。

4.1 活动安排问题

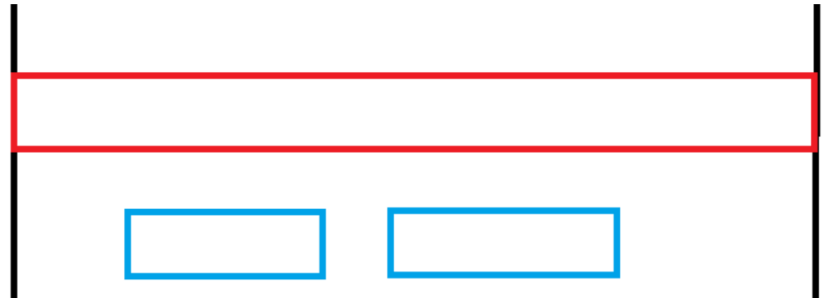
活动安排问题要求高效地安排一系列争用某一公共资源的活动。贪心算法提供了一个简单、漂亮的方法使得尽可能多的活动能兼容地使用公共资源。

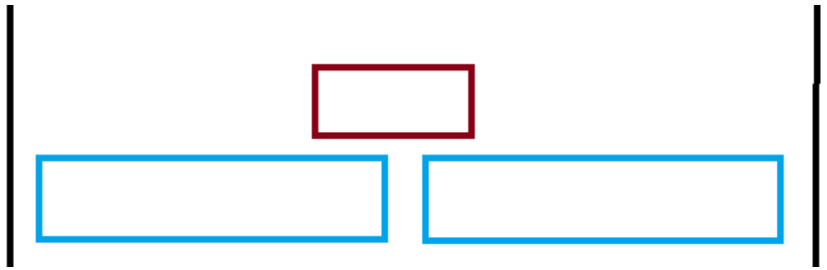
活动安排问题就是要在所给的活动集合中选出最大的相容活动子集合，是可以用贪心算法有效求解的很好例子。

4.1 活动安排问题

几种贪心策略:

- 贪开始最早
- 贪持续最短
- 贪结束最早





4.1 活动安排问题

同济大学 TONGJI UNIVERSITY

求解活动安排问题的贪心算法GreedySelector:

```
template<class Type>
void GreedySelector(int n, Type s[], Type f[], bool A[]){
    A[1]=true;
    int j=1;
    for (int i=2;i<=n;i++) {
      if (s[i]>=f[j]) { A[i]=true; j=i; }
      else A[i]=false;
      }
}
```

各活动的起始时间和结束时间存储于数组s和f中且按结束时间的非减序排列

4.1 活动安排问题

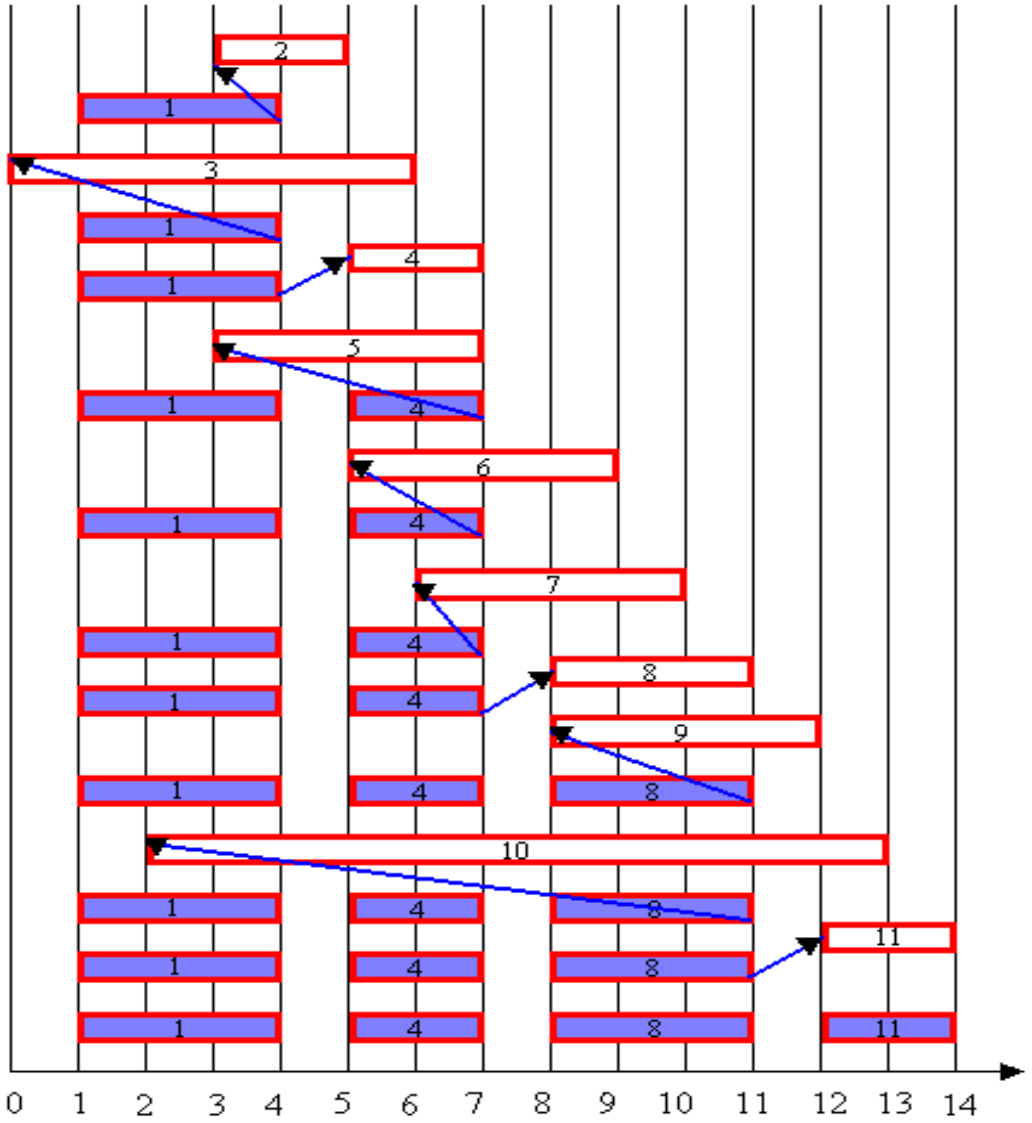
- 由于输入的活动以其完成时间的非减序排列，所以算法greedySelector每次总是选择具有最早完成时间的相容活动加入集合A中。直观上，按这种方法选择相容活动为未安排活动留下尽可能多的时间。也就是说，该算法的贪心选择的意义是使剩余的可安排时间段极大化，以便安排尽可能多的相容活动。
- 算法greedySelector的效率极高。当输入的活动已按结束时间的非减序排列，算法只需O(n)的时间安排n个活动，使最多的活动能相容地使用公共资源。如果所给出的活动未按非减序排列，可以用O(nlogn)的时间重排。

4.1 活动安排问题

- 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非减序排列如下：

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

4.1 活动安排问题



算法greedySelector 的计算过程

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

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。

4.1 活动安排问题

- 若被检查的活动i的开始时间si小于最近选择的活动j的结束时间fi，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。

1）数学归纳法

2）每一步选择不比其他算法差

3）基于算法的输出结果

贪心算法的特点

- 贪心算法适用于组合优化问题；
- 求解过程是多步判断的过程，最终的判断序列对应问题的最优解；
- 依据某种“短视的”贪心选择性质判断，性质的好坏决定算法的成败；
- 贪心算法必须进行正确性证明；
- 证明贪心法不正确的技巧：举反例
- 贪心算法的优势：算法简单，时空复杂度低

活动安排问题贪心算法

- 命题：算法GreedySelector执行到第k步，选择k项活动$i_1=1,i_2,\cdots.i_k$，则存在最优解A,包含活动$i_1=1, i_2,\cdots.i_k$

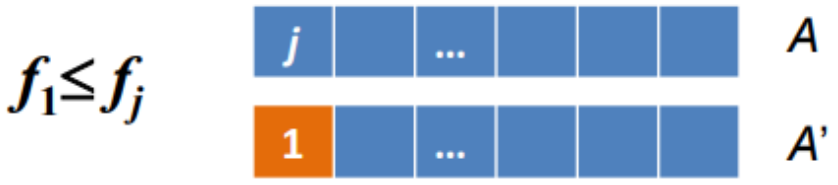
- 根据上述命题，对于任何k,算法前k步的选择都将导致最优解，至多到第n步将得到问题实例的最优解

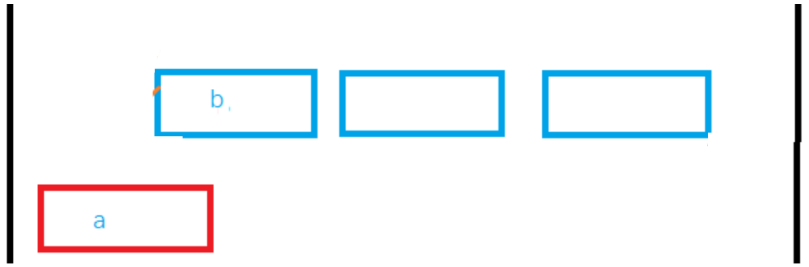
归纳证明

- 设S={1,2,3,…,n}是活动集，且$f_1 \leqslant f_2 \leqslant f_3 \leqslant \cdots \leqslant f_n$

k=1, 证明存在最优解包含活动1

证：任取最优解A，A中活动按完成时间递增排列，如果A的第一个活动为j，j≠1，可以用活动1替换A中的第一个活动j，得到解A’，A’=(A-{j})∨{1}，$f_1 \leqslant f_j$，A’也是最优解，且包含活动1。





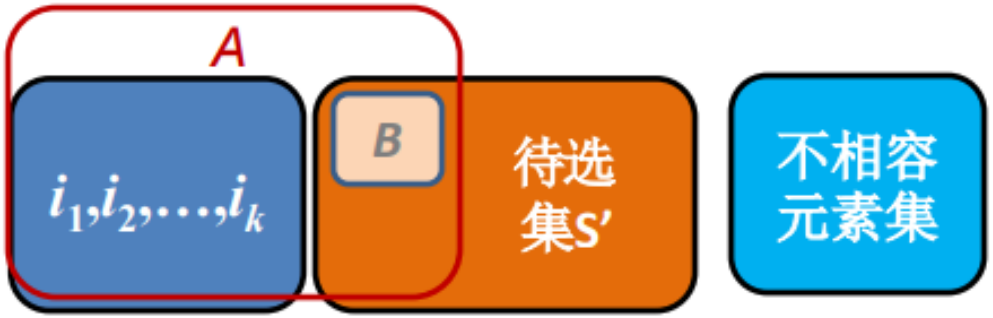
归纳证明

- 假设命题对k成立，证明对k+1也成立

证：算法执行到第k步，选择了活动$i_1=1, i_2, \cdots, i_k$，根据归纳假设存在最优解A包含$i_1=1, i_2, \cdots, i_k$，A中剩下的活动选自集合S'

$$S' = \{ i \mid i \in S,\ s_i \geq f_k \}$$

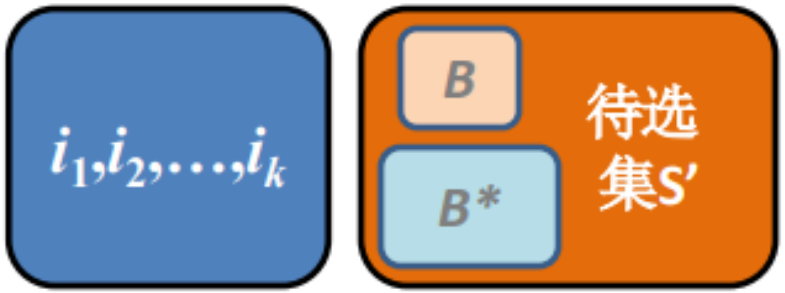
$$A = \{ i_1, i_2, \ldots, i_k \} \cup B$$



归纳证明

- B是S’的最优解。
- 若B不是S’最优解，设S’的最优解为B*, 即B*的活动比B要多，那么是S的最优解，且比A的活动数多，这跟A的最优解矛盾

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

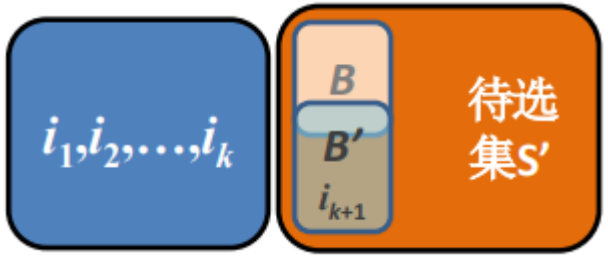


归纳证明

- 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|，

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

也是原问题的最优解



活动选择问题的贪心法证明:涉及步数的算法正确性命题

4.2 贪心算法的基本要素

可以用贪心算法求解的问题的一般特征:

- 对于一个具体的问题，怎么知道是否可用贪心算法解此问题，以及能否得到问题的最优解呢?这个问题很难给予肯定的回答。
- 但是，从许多可以用贪心算法求解的问题中看到这类问题一般具有2个重要的性质：贪心选择性质和最优子结构性质。

4.2 贪心算法的基本要素

1.贪心选择性质

- 所谓贪心选择性质是指所求问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来达到。这是贪心算法可行的第一个基本要素，也是贪心算法与动态规划算法的主要区别。
- 动态规划算法通常以自底向上的方式解各子问题，而贪心算法则通常以自顶向下的方式进行，以迭代的方式作出相继的贪心选择，每作一次贪心选择就将所求问题简化为规模更小的子问题。
- 对于一个具体问题，要确定它是否具有贪心选择性质，必须证明每一步所作的贪心选择最终导致问题的整体最优解。

2.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。

问题的最优子结构性质是该问题可用动态规划算法或贪心算法求解的关键特征。

3.贪心算法与动态规划算法的差异

贪心算法和动态规划算法都要求问题具有最优子结构性质，这是2类算法的一个共同点。但是:

- 对于具有最优子结构的问题应该选用贪心算法还是动态规划算法求解?
- 是否能用动态规划算法求解的问题也能用贪心算法求解?

下面研究2个经典组合优化问题，并以此说明贪心算法与动态规划算法的主要差别。

4.2 贪心算法的基本要素

- 0-1背包问题：

给定n种物品和一个背包。物品i的重量是Wi，其价值为Vi，背包的容量为C。
应如何选择装入背包的物品，使得装入背包中物品的总价值最大?

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。

4.2 贪心算法的基本要素

- 背包问题：

- 与0-1背包问题类似，所不同的是在选择物品i装入背包时，可以选择物品i的一部分，而不一定要全部装入背包，1≤i≤n。

- 这2类问题都具有最优子结构性质，极为相似，但背包问题可以用贪心算法求解，而0-1背包问题却不能用贪心算法求解。

4.2 贪心算法的基本要素

- 用贪心算法解背包问题的基本步骤：

- 计算每种物品单位重量的价值Vi/Wi，并按非增次序进行排序。
- 依贪心选择策略，将尽可能多的单位重量价值最高的物品装入背包。若将这种物品全部装入背包后，背包内的物品总重量未超过C，则选择单位重量价值次高的物品并尽可能多地装入背包。依此策略一直地进行下去，直到背包装满为止。

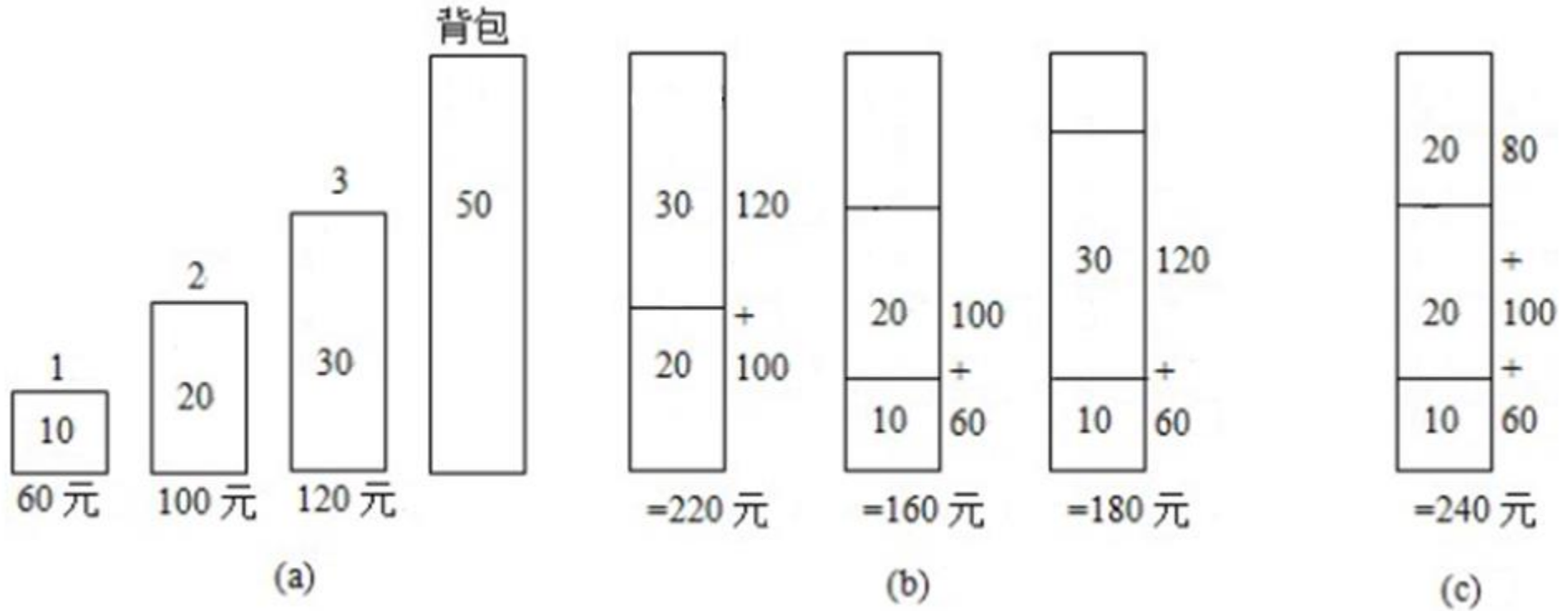
4.2 贪心算法的基本要素

算法描述：

```
void Knapsack(int n, float M, float v[], float w[], float x[]){
    Sort(n,v,w);
    int i;
    for (i=1;i<=n;i++) x[i]=0;
    float c=M;
    for (i=1;i<=n;i++) {
      if (w[i]>c) break;
      x[i]=1;
      c-=w[i];
      }
    if (i<=n) x[i]=c/w[i];
}
```

- 算法knapsack的主要计算时间在于将各种物品依其单位重量的价值从大到小排序。因此，算法的计算时间上界为O(nlogn)。
- 为了证明算法的正确性，还必须证明背包问题具有贪心选择性质。

4.2 贪心算法的基本要素



0-1背包　　背包问题

贪心选择对0-1 背包不适用

4.2 贪心算法的基本要素

- 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另一重要特征。
- 实际上也是如此，动态规划算法可以有效地解0-1背包问题。

动态规划和贪心算法

- 动态规划和贪心算法都是一种递推算法，均有最优子结构性质，通过局部最优解来推导全局最优解。

两者之间的区别在于：

- 贪心算法中作出的每步贪心决策都无法改变，因为贪心策略是由上一步的最优解推导下一步的最优解，而上一部之前的最优解则不作保留，贪心算法每一步的最优解一定包含上一步的最优解。
- 动态规划算法中全局最优解中一定包含某个局部最优解，但不一定包含前一个局部最优解，因此需要记录之前的所有最优解。

4.3 最优装载

有一批集装箱要装上一艘载重量为c的轮船。其中集装箱i的重量为W_i。最优装载问题要求确定在装载体积不受限制的情况下，将尽可能多的集装箱装上轮船。

$x_i \in \{0,1\},\ 1 \le i \le n$　　解向量

$\max \sum_{i=1}^{n} x_i$　　目标函数

$\sum_{i=1}^{n} w_i x_i \le c$　　约束条件

1.算法描述

最优装载问题可用贪心算法求解。采用重量最轻者先装的贪心选择策略，可产生最优装载问题的最优解。具体算法描述如右图。

```
template<class Type>
void Loading(int x[],  Type w[], Type c, int n){
    int *t = new int [n+1];
    Sort(w, t, n);//排序
    for (int i = 1; i <= n; i++) x[i] = 0;//初始化
    for (int i = 1; i <= n && w[t[i]] <= c; i++) {
    //判断是否能入
        x[t[i]] = 1;
        c -= w[t[i]];}
}
```

4.3 最优装载

2.贪心选择性质

可以证明最优装载问题具有贪心选择性质。

3.最优子结构性质

最优装载问题具有最优子结构性质。

设$(x_1,x_2,\ldots\ldots x_n)$是最优装载问题的满足贪心选择性质的最优解，则易知，$x_1=1,(x_2,x_3,\ldots\ldots x_n)$是轮船载重量为c-w1，待装船集装箱为{2,3，……n}时相应最优装载问题的最优解。因此，最优装载问题具有最优子结构性质。

- **算法loading的主要计算量在于将集装箱依其重量从小到大排序，故算法所需的计算时间为** O(nlogn)。

最优装载正确性证明

命题：对装载问题任何规模为 n 的 输入实例，算法得到最优解。

- 设集装箱重量从小到大记为$w_1 \leqslant w_2 \leqslant w_3 \leqslant \cdots \leqslant w_n$
- 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k个箱子的输入实例贪心法都能得到最优解，那么对于任何k+1个箱子的输入实例贪心法也得到最优解。

正确性证明

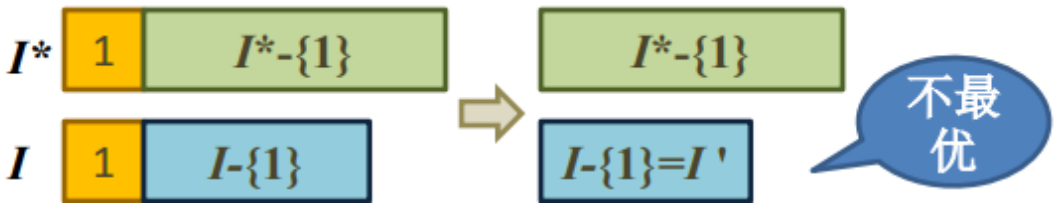
假设对于k个集装箱的输入，贪心算法都可以得到最优解，考虑输入$N = \{1,2,\ldots,k+1\}$，其中$w_1 \le w_2 \le \ldots \le w_{k+1}$。

由归纳假设，对于$N' = \{2,\ldots,k+1\}$，$C' = C - w_1$。贪心法得到最优解I'。令$I = \{1\} \cup I'$，则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'的解，且：

$$|I^* - \{1\}| > |I - \{1\}| = |I'|$$

与I'的最优性矛盾。



4.4 哈夫曼编码

- 哈夫曼编码，又称霍夫曼编码，是一种编码方式，哈夫曼编码是可变字长编码的一种。哈夫曼于1952年提出一种编码方法，该方法完全依据字符出现概率来构造异字头的平均长度最短的码字，有时称之为最佳编码。

- 哈夫曼编码是广泛地用于数据文件压缩的十分有效的编码方法。其压缩率通常在20%～90%之间。哈夫曼编码算法用字符在文件中出现的频率表来建立一个用0，1串表示各字符的最优表示方式。

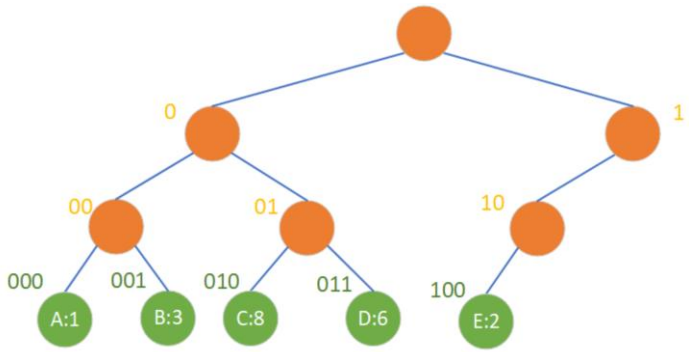
- 给出现频率高的字符较短的编码，出现频率较低的字符以较长的编码，可以大大缩短总码长。

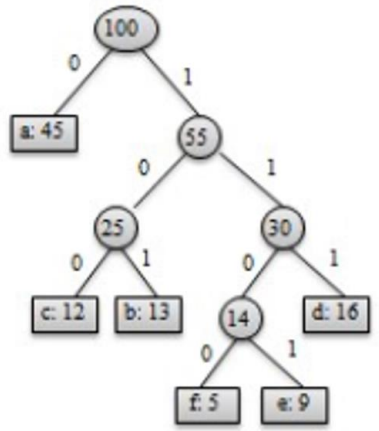
4.4 哈夫曼编码

数据文件100,000个字符，以6个字符形式出现。

定长码： 3*1000000= 300,000 位

变长码：45*1+13*3+12*3+16*3+9*4+5*5=224,000 位





4.4 哈夫曼编码

1.前缀码

- 对每一个字符规定一个0,1串作为其代码，并要求任一字符的代码都不是其它字符代码的前缀。这种编码称为前缀码。
- 编码的前缀性质可以使译码方法非常简单。
- 表示最优前缀码的二叉树总是一棵完全二叉树，即树中任一结点都有2个儿子结点。
- 平均码长定义为： $B(T)=\sum_{c\in C} f(c)d_T(c)$

f(c)为频率分布，$d_T(c)$为深度，使平均码长达到最小的前缀码编码方案称为给定编码字符集C的最优前缀码。

2.构造哈夫曼编码

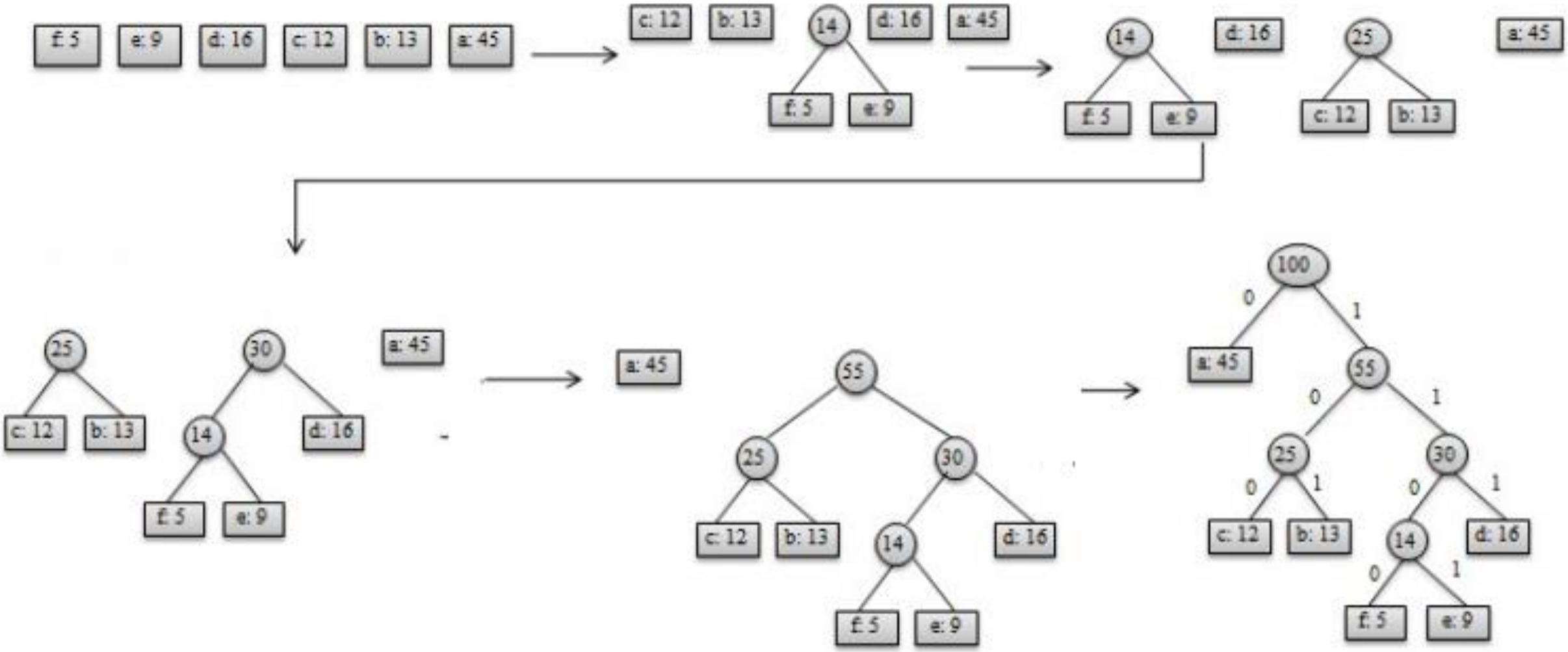
- 哈夫曼提出构造最优前缀码的贪心算法，由此产生的编码方案称为哈夫曼编码。
- 哈夫曼算法以自底向上的方式构造表示最优前缀码的二叉树T。
- 算法以|C|个叶结点开始，执行|C|－1次的“合并”运算后产生最终所要求的树T。

4.4 哈夫曼编码

- 在书上给出的算法huffmanTree中，编码字符集中每一字符c的频率是f(c)。以f为键值的优先队列Q用在贪心选择时有效地确定算法当前要合并的2棵具有最小频率的树。一旦2棵具有最小频率的树合并后，产生一棵新的树，其频率为合并的2棵树的频率之和，并将新树插入优先队列Q。经过n－1次的合并后，优先队列中只剩下一棵树，即所要求的树T。

	a	b	c	d	e	f
频率(千次)	45	13	12	16	9	5
定长码	000	001	010	011	100	101
变长码	0	101	100	111	1101	1100

4.4 哈夫曼编码



	a	b	c	d	e	f
变长码	0	101	100	111	1101	1100

3.哈夫曼算法的正确性

- 要证明哈夫曼算法的正确性，只要证明最优前缀码问题具有贪心选择性质和最优子结构性质。

(1)贪心选择性质

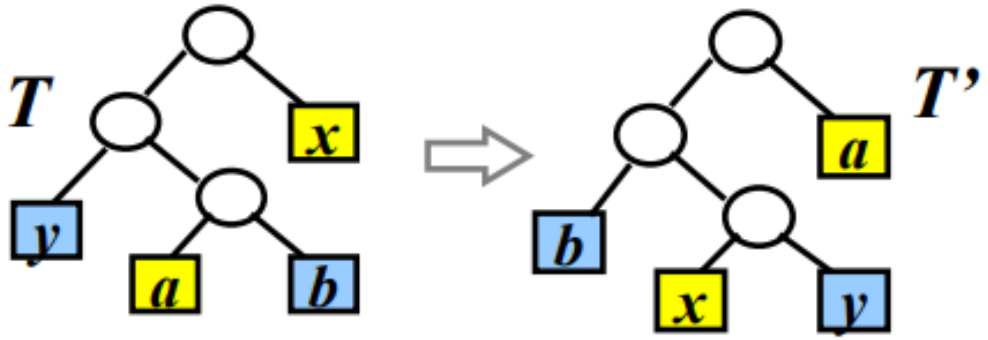
(2)最优子结构性质

- 算法huffmanTree用最小堆实现优先队列Q。初始化优先队列需要O(n)计算时间，由于最小堆的removeMin和put运算均需O(logn)时间，n－1次的合并总共需要O(nlogn)计算时间。因此，关于n个字符的哈夫曼算法的计算时间为O(nlogn) 。

最优前缀码性质：引理1

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.

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



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

其中$d_T(i)$为i 在T 中的层数(i到根的距离)

引理2

引理 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

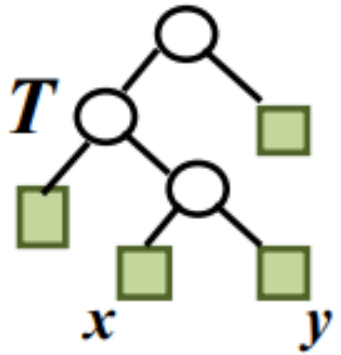
$$f(z)=f(x)+f(y)$$

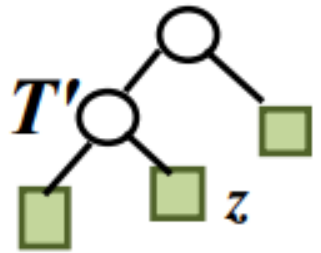
T'是对应二元前缀码

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

的二叉树，那么

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

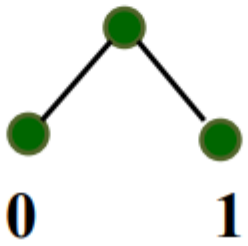




归纳基础

n=2，字符集　$C=\{x_1, x_2\}$，

对任何代码的字符至少都需要1位二进制数字. Huffman算法得到的代码是 0 和 1，是最优前缀码.



算法正确性证明思路

定理 Huffman 算法对任意规模为n（$n\geq 2$）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.

归纳步骤

假设Huffman算法对于规模为 k 的字符集都得到最优前缀码. 考虑规模为 $k+1$的字符集

$$C = \{x_1, x_2, ..., x_{k+1}\},$$

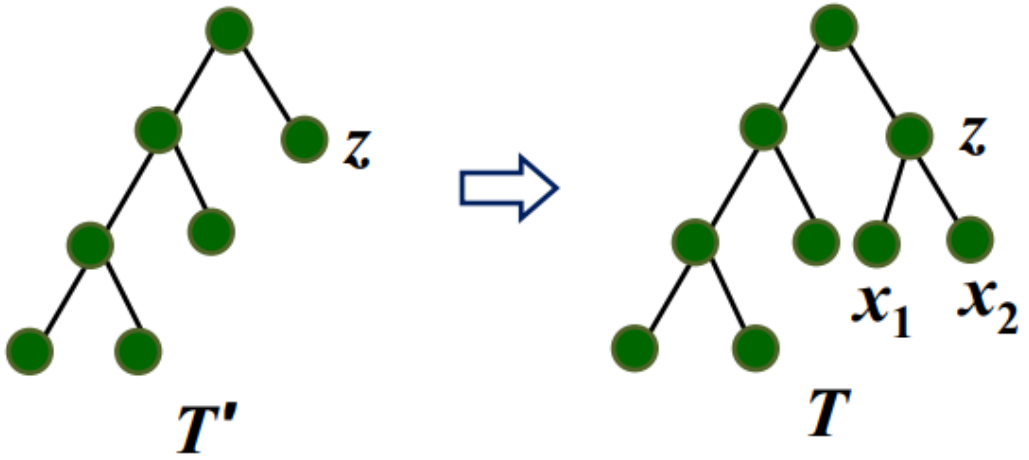
其中 $x_1, x_2 \in C$是频率最小的两个字符.

令

$$C' = (C - \{x_1, x_2\}) \cup \{z\},$$
$$f(z) = f(x_1) + f(x_2)$$

根据归纳假设，算法得到一棵关于字符集 C',频率$f(z)$和$f(x_i)$ $(i = 3,4, ...,k+1)$ 的最优前缀码的二叉树T'.

把 x_1, x_2作为 z 的儿子附到 T'上，得到树T，那么T是关于$C=(C'-\{z\})\cup\{x_1, x_2\}$的最优前缀码的二叉树.



归纳步骤

如若不然, 存在更优树 T^*, $B(T^*)<B(T)$,
且由引理1，其树叶兄弟是 x_1和 x_2.

去掉 T^*中 x_1和 x_2，得到$T^{*\prime}$. 根据引理2

$$\begin{aligned} B(T^{*\prime}) &= B(T^*)-(f(x_1)+f(x_2)) \\ &< B(T)-(f(x_1)+f(x_2)) \\ &= B(T') \end{aligned}$$

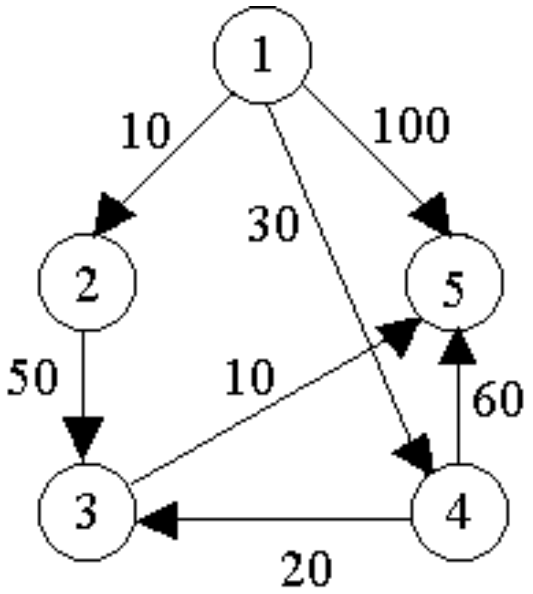
与 T'是一棵关于 C'的最优前缀码的二叉树矛盾.

4.5 单源最短路径

给定带权有向图G =(V,E)，其中每条边的权是非负实数。另外，还给定V中的一个顶点，称为源。现在要计算从源到所有其它各顶点的最短路长度。这里路的长度是指路上各边权之和。这个问题通常称为单源最短路径问题。

1.算法基本思想

Dijkstra算法是解单源最短路径问题的贪心算法。



4.5 单源最短路径

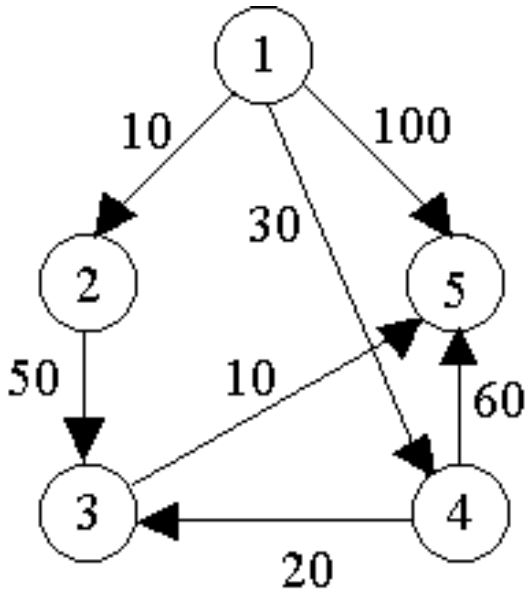
- 其基本思想是，设置顶点集合S并不断地作贪心选择来扩充这个集合。一个顶点属于集合S当且仅当从源到该顶点的最短路径长度已知。
- 初始时，S中仅含有源。设u是G的某一个顶点，把从源到u且中间只经过S中顶点的路称为从源到u的特殊路径，并用数组dist记录当前每个顶点所对应的最短特殊路径长度。
- Dijkstra算法每次从V-S中取出具有最短特殊路长度的顶点u，将u添加到S中，同时对数组dist作必要的修改。一旦S包含了所有V中顶点，dist就记录了从源到所有其它顶点之间的最短路径长度。

V $S=\{v_0\}$, V-S

V=S

4.5 单源最短路径

例如，对下图中的有向图，应用Dijkstra算法计算从源顶点1到其它顶点间最短路径的过程列在下表中。



- Dijkstra算法的迭代过程：

迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1, 2}	2	10	60	30	100
2	{1, 2, 4}	4	10	50	30	90
3	{1, 2, 4, 3}	3	10	50	30	60
4	{1, 2, 4, 3, 5}	5	10	50	30	60

2.算法的正确性和计算复杂性

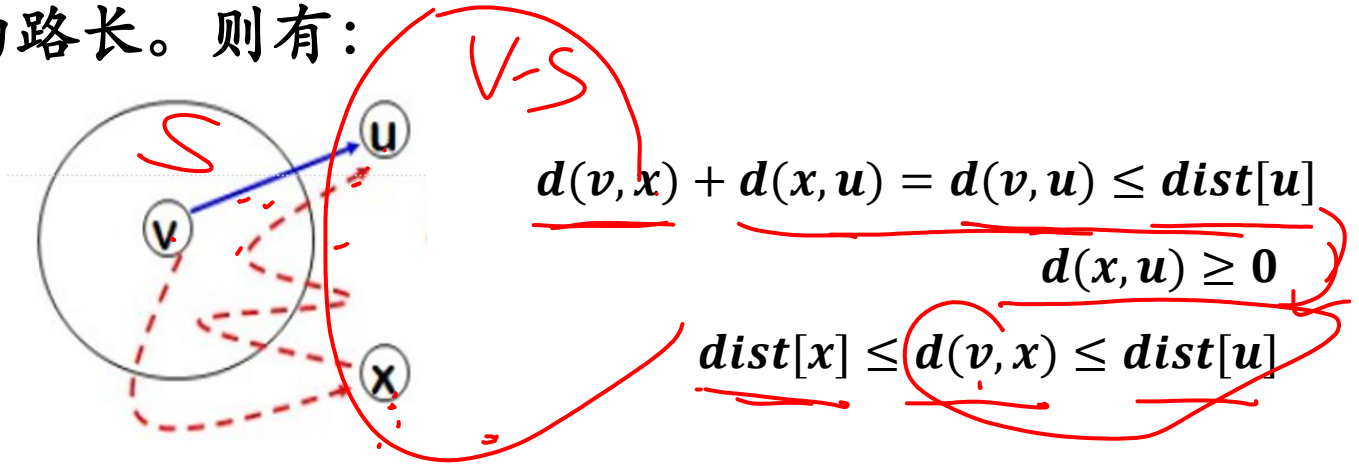
(1)贪心选择性质

(2)最优子结构性质

(3)计算复杂性

(1).贪心选择性质

- 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u的最短路径长度dist[u]。
- 为什么从源到u没有更短的其他路径？如果存在一条从源到u且长度比dist[u]更短的路，设这条路初次走出S之外到达的顶点为x，然后徘徊于S内外若干次，最后离开S到达u。在这条路上分别记$d(v,x)$,$d(x,u)$和$d(v,u)$为顶点v到顶点x,顶点x到顶点u，顶点v到顶点u的路长。则有：



$$d(v,x)+d(x,u)=d(v,u)\le dist[u]$$

$$d(x,u)\ge 0$$

$$dist[x]\le d(v,x)\le dist[u]$$

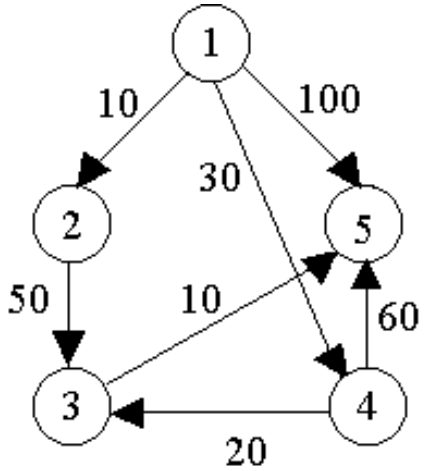
$dist[x]\le dist[u]$与u是当前贪心选择矛盾！

(2).最优子结构性质

- 如果$S(i,j)=\{V_i,\ldots,V_k,\ldots,V_s,\ldots,V_j\}$是从顶点i到j的最短路径，k和s是这条路径上的一个中间顶点，那么$S(k,s)$必定是从k到s的最短路径。
- 证明：
  假设$S(i,j)=\{V_i,\ldots,V_k,\ldots,V_s,\ldots,V_j\}$是从顶点i到j的最短路径，则有$S(i,j)=S(i,k)+S(k,s)+S(s,j)$。而$S(k,s)$不是从k到s的最短距离，那么必定存在另一条从k到s的最短路径$S'(k,s)$，那么$S'(i,j)=S(i,k)+(k,s)+S(s,j)<S(i,j)$。则与$S(i,j)$是从i到j的最短路径相矛盾。因此该性质得证。

(3).计算复杂性

对于具有n个顶点和e条边的带权有向图，如果用带权邻接矩阵表示这个图，那么Dijkstra算法的主循环体需要$O(n)$时间。这个循环需要执行n-1次，所以完成循环需要$O(n^2)$时间。算法的其余部分所需要时间不超过$O(n^2)$。



4.6 最小生成树

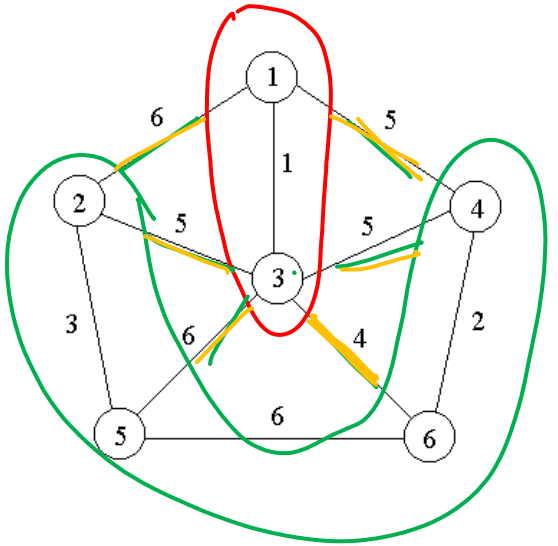
- 设G =(V,E)是无向连通带权图，即一个网络。E中每条边(v,w)的权为c[v][w]。如果G的子图G'是一棵包含G的所有顶点的树，则称G'为G的生成树。生成树上各边权的总和称为该生成树的耗费。在G的所有生成树中，耗费最小的生成树称为G的最小生成树。
- 网络的最小生成树在实际中有广泛应用。例如，在设计通信网络时，用图的顶点表示城市，用边(v,w)的权c[v][w]表示建立城市v和城市w之间的通信线路所需的费用，则最小生成树就给出了建立通信网络的最经济的方案。

1.最小生成树性质

- 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。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2个算法做贪心选择的方式不同，它们都利用了下面的最小生成树性质：
- 设G=(V,E)是连通带权图，U是V的真子集。如果(u,v)∈E，且u∈U，v∈V-U，且在所有这样的边中，(u,v)的权c[u][v]最小，那么一定存在G的一棵最小生成树，它以(u,v)为其中一条边。这个性质有时也称为MST性质。

4.6 最小生成树

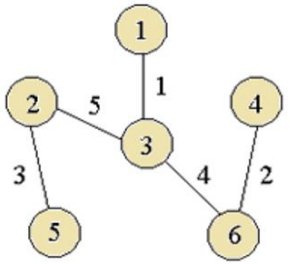
- 设G=(V,E)是连通带权图，U是V的真子集。如果(u,v)∈E，且u∈U，v∈V-U，且在所有这样的边中，(u,v)的权c[u][v]最小，那么一定存在G的一棵最小生成树，它以(u,v)为其中一条边。这个性质有时也称为MST性质。



G=(V,E,A)
V={1,2,3,4,5,6}
E={(1,2),(1,3),….}
A={0, 6, 1, 5, ∞, ∞ }
{ }

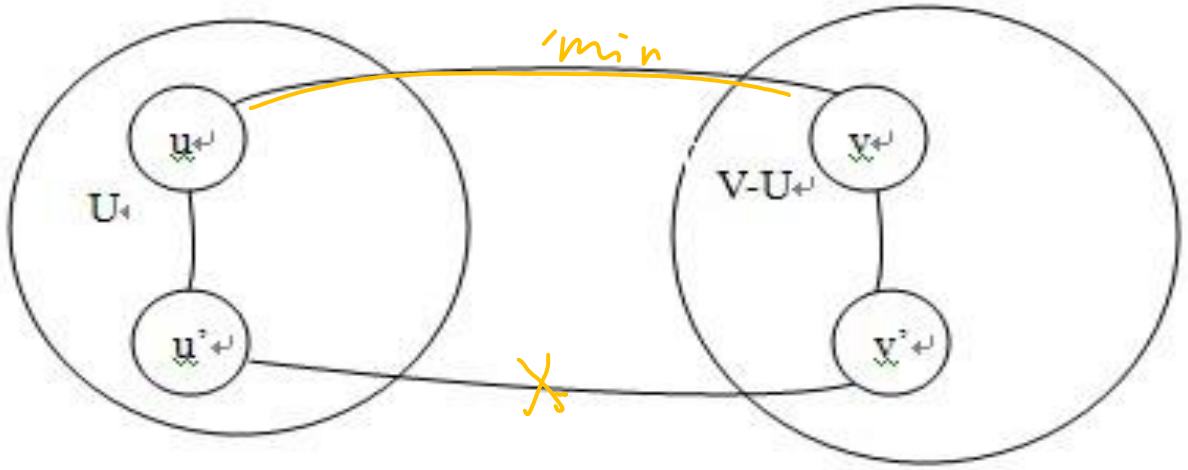
U={1,3}
V-U={2,4,5,6}
U和V-U权值最小的边是（3,6）

G的最小生成树里面一定有一棵是包含（3, 6）这边条的。



MST性质证明：

- 假设G的任何一颗最小生成树都不包含边(u,v)。将边(u,v)添加到G的一颗最小生成树T上，将产生含有边(u,v)的圈，并且在这个圈上有一条不同于(u,v)的边(u',v')，使得u' ∈ U，v ' ∈ V-U。将边(u',v')删去，得到G的另一颗生成树T'。由于c[u][v]<=c[u'][v']，所以T'的耗费<=T的耗费。于是T'是一颗含有边(u,v)的最小生成树，这与假设矛盾。



2.Prim算法

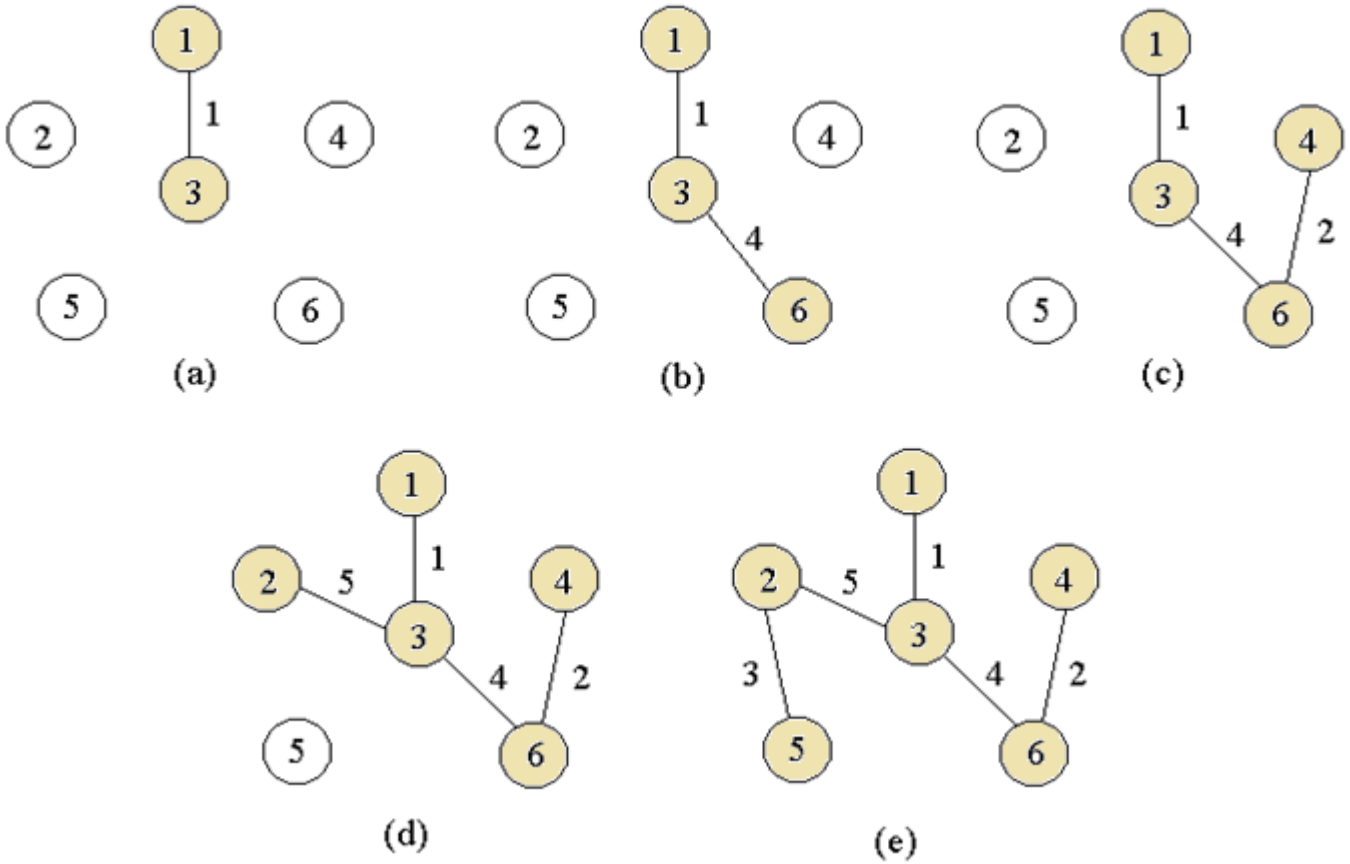
- 设G=(V,E)是连通带权图，V={1,2,…,n}。
- 构造G的最小生成树的Prim算法的基本思想是：首先置U={1}，然后，只要U是V的真子集，就作如下的贪心选择：选取满足条件i∈U，j∈V-U，且c[i][j]最小的边，将顶点j添加到U中。这个过程一直进行到U=V时为止。

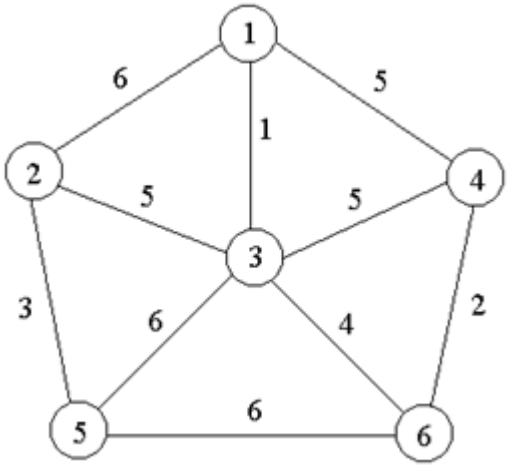
在这个过程中选取到的所有边恰好构成G的一棵最小生成树。

4.6 最小生成树

- 利用最小生成树性质和数学归纳法容易证明，上述算法中的边集合T始终包含G的某棵最小生成树中的边。因此，在算法结束时，T中的所有边构成G的一棵最小生成树。
- 例如，对于下图中的带权图，按Prim算法选取边的过程如下页图所示。

4.6 最小生成树





4.6 最小生成树

- 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)。实现这个目的的较简单的办法是设置2个数组closest和lowcost。
- closest[j]是j在S中的邻接顶点满足c[j][closest[j]]<=c[j][k],即为同j最近的节点； lowcost[j]=c[j][closest[j]]。
- 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。
- 用这个办法实现的Prim算法所需的计算时间为$O(n^2)$。

正确性证明

命题：对于任意 $k < n$, 存在一棵最小生成树包含算法前 k 步选择的边.

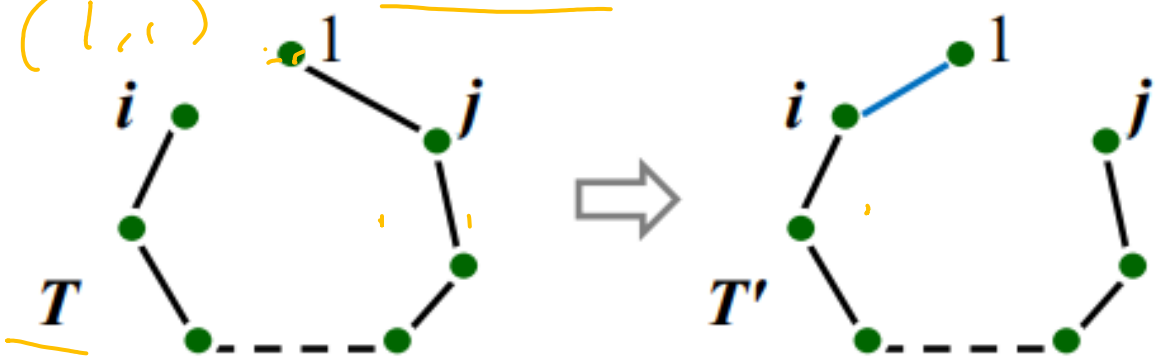
归纳基础：$k = 1$, 存在一棵最小生成树 T 包含边 $e = \{1, i\}$, 其中$\{1, i\}$是所有关联 1 的边中权最小的.

归纳步骤：假设算法前 k 步选择的边构成一棵最小生成树的边，则算法前 $k+1$ 步选择的边也构成一棵最小生成树的边.

归纳基础

证明：存在一棵最小生成树 T 包含关联结点1的最小权的边$e=\{1,i\}$.

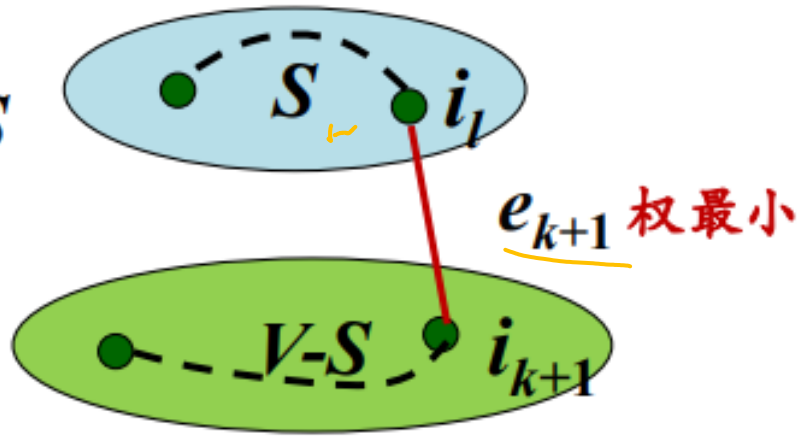
证　设 T 为一棵最小生成树，假设 T 不包含$\{1, i\}$, 则$T \cup \{\{1,i\}\}$ 含有一条回路，回路中关联1的另一条边$\{1, j\}$. 用$\{1, i\}$替换$\{1, j\}$得到树 T'，　则 T' 也是生成树，且 $W(T') \le W(T)$.



归纳步骤

假设算法进行了k步，生成树的边为$e_1, e_2, \cdots, e_k$，这些边的端点构成集合S. 由归纳假设存在 G 的一棵最小生成树 T 包含这些边.

算法第 $k+1$ 步选择顶点 i_{k+1}，则 i_{k+1} 到S中顶点边权最小，设此边 $e_{k+1}=\{i_{k+1}, i_l\}$.
若$e_{k+1} \in T$，算法$k+1$步显然正确.



归纳步骤

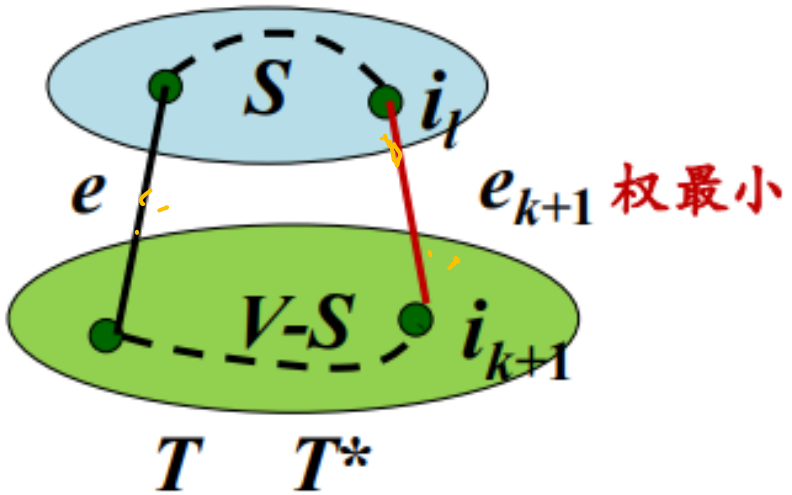
假设 T 不含有 e_{k+1}，则将 e_{k+1} 加到 T 中形成一条回路. 这条回路有另外一条连接 S 与 V–S中 顶点的边 e,

令 $T^*=(T-\{e\})\cup\{e_{k+1}\}$

则 T^*是 G 的一棵生成树,包含$e_1,e_2,\ldots,e_{k+1}$，且

$$W(T^*)\le W(T)$$

算法到 k+1步仍得到最小生成树.



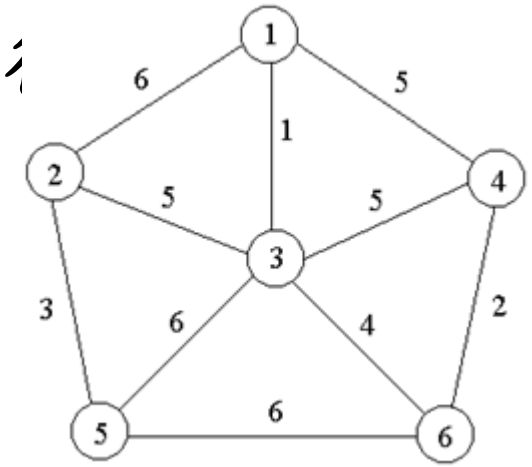
3.Kruskal算法

Kruskal算法构造G的最小生成树的基本思想是，首先将G的n个顶点看成n个孤立的连通分支。将所有的边按权从小到大排序。然后从第一条边开始，依边权递增的顺序查看每一条边，并按下述方法连接2个不同的连通分支：当查看到第k条边(v,w)时，如果端点v和w分别是当前2个不同的连通分支T1和T2中的顶点时，就用边(v,w)将T1和T2连接成一个连通分支，然后继续查看第k+1条边；如果端点v和w在当前的同一个连通分支中，就直接再查看第k+1条边。这个过程一直进行到只剩下一个连通分支时为止。

4.6 最小生成树

例如，对右图的连通带权图，按Kruskal算法顺序得

最小生成树上的边如下图所示。



4.6 最小生成树

- 关于集合的一些基本运算可用于实现Kruskal算法。
- 按权的递增顺序查看等价于对优先队列执行removeMin运算。可以用堆实现这个优先队列。
- 对一个由连通分支组成的集合不断进行修改，需要用到抽象数据类型并查集UnionFind所支持的基本运算。
- 当图的边数为e时，Kruskal算法所需的计算时间是$O(eloge)$。

算法名	普利姆算法	克鲁斯卡尔算法
基本思想	以**顶点**为对象	以**边**为对象
时间复杂度	$O(n^2)$	O(eloge)
适应范围	稠密图	稀疏图

4.7 多机调度问题

- 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。

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。

- 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。

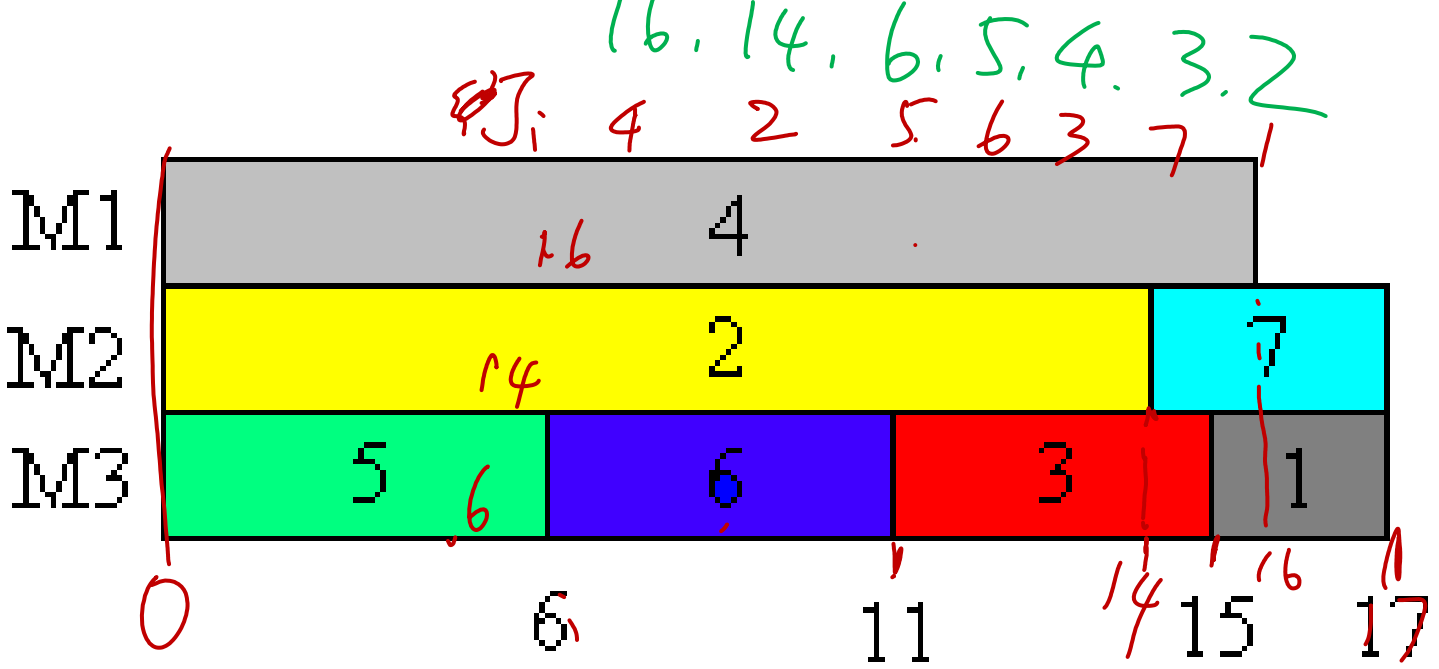
4.7 多机调度问题

- 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$n \leq m$时，只要将机器i的[0, ti]时间区间分配给作业i即可，算法只需要O(1)时间。
- 当$n > m$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。算法所需的计算时间为O(nlogn)。

4.7 多机调度问题

同济大学 TONGJI UNIVERSITY

- 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}。按算法greedy产生的作业调度如下图所示，所需的加工时间为17。



贪心法小结

- 1.适用于组合优化问题。求解过程是多步判断，判断的依据是局部最优策略，使目标值达到最大（或最小），与前面的子问题 计算结果无关。
- 2.局部最优策略的选择是算法正确性的关键。
- 3.正确性证明方法：数学归纳法、交换论证。使用数学归纳法主要通过对算法步数或者问题规模进行归纳。
- 如果要证明贪心策略是错误的，只需举出反例。
- 4.自顶向下求解，通过选择将问题归约为小的子问题。
- 5.如果贪心法得不到最优解，可以对问题的输入进行分析，或估计算法的近似比。
- 6.对原始数据排序后，贪心法往往是一轮处理，时间复杂度和空间复杂度都很低。

贪心法小结

	标准分治	动态规划	贪心选择
适用类型	通用问题	优化问题	优化问题
子问题结构	每个子问题不同	很多子问题重复	只有一个子问题
最优子结构性质	满足	满足	满足
求解子问题数	全部子问题都要解决	全部子问题都要解决	只解决一个子问题
子问题在最优解里	全部	部分	部分
选择与求解次序	先选择后解决子问题	先解决子问题后选择	先选择后解决子问题