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s_i和一个结束时间f_i,且$s_i < f_i$。如果选择了活动i，则它在半开时间区间$[s_i, f_i)$内占用资源。若区间$[s_i, f_i)$与区间$[s_j, f_j)$不相交，则称活动i与活动j是相容的。也就是说，当$s_i \geq f_j$或$s_j \geq f_i$时，活动i与活动j相容。

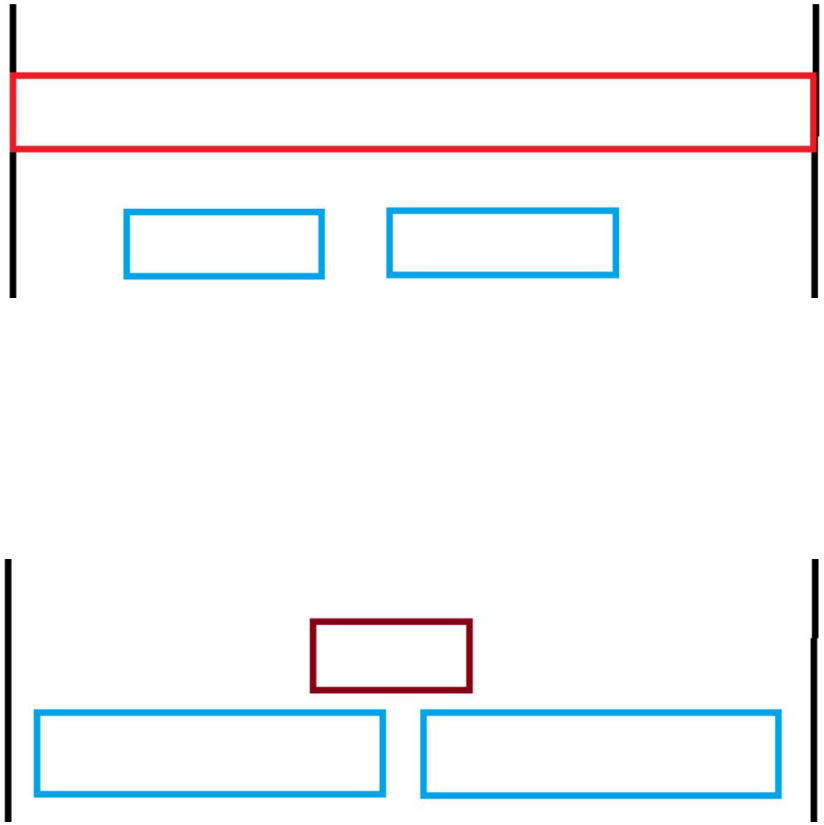
4.1 活动安排问题

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

- 有几种贪心策略:
- 贪开始最早
- 贪持续最短
- 贪结束最早



4.1 活动安排问题

- 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。

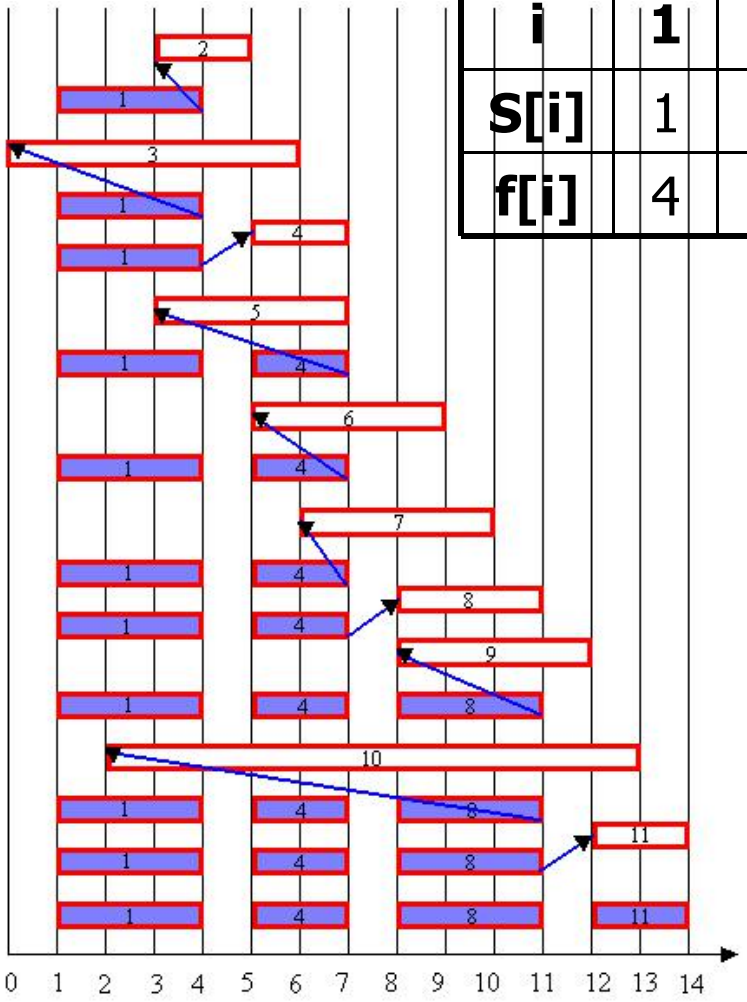
4.1 活动安排问题

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

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



算法greedySelector 的计算过程

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

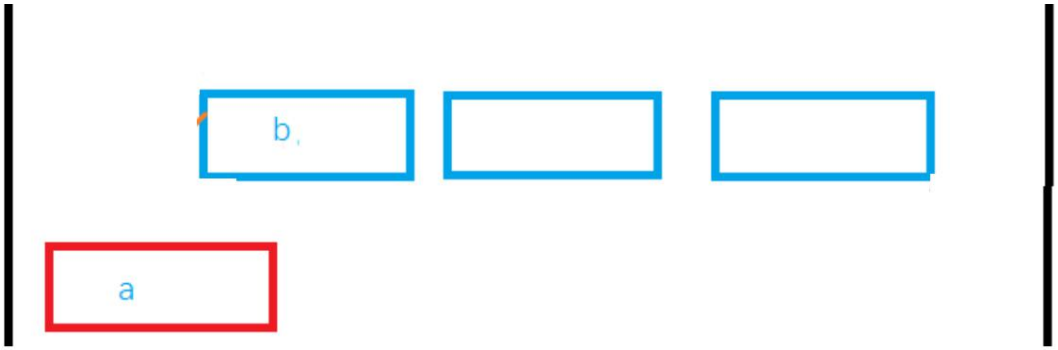
- 若被检查的活动i的开始时间s_i小于最近选择的活动j的结束时间f_i，则不选择活动i，否则选择活动i加入集合A中。
- 贪心算法并不总能求得问题的整体最优解。但对于活动安排问题，贪心算法greedySelector却总能求得的整体最优解，即它最终所确定的相容活动集合A的规模最大。这个结论可以证明。

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

4.1 活动安排问题

证明

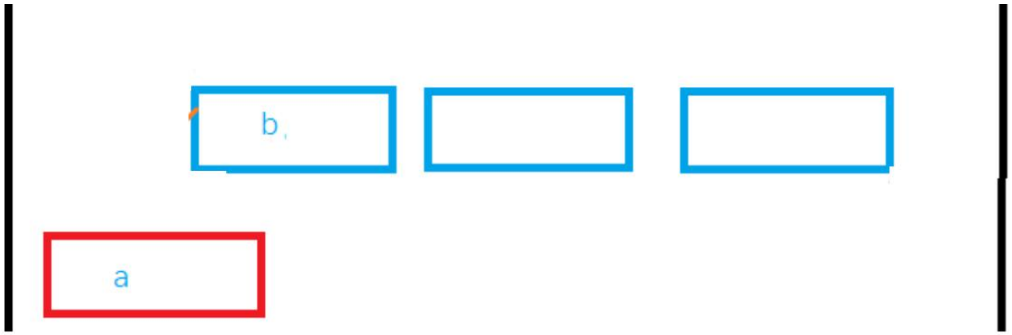
- 最优解一定包含结束时间最早的活动。
- 假设活动a结束最早，可是我们能够找到一种方法A，使得不选a也能参加活动次数最多，假设A方法中结束时间最早的活动是b，那么只可能有如下情况



a与b不重叠的情况不存在，因为A答案已经是最优解了，如果ab不重叠，那么A答案仍然可以加入a活动，A答案不是最优解

4.1 活动安排问题

- 因为a活动比b结束早（或者等于），而答案A中，b和下一个活动必定不冲突。
- 而a早于b结束，可以轻易想到，a与答案A中的活动也不冲突
- 那么可以把b替换成a，对结果没有影响的，所以最优解一定有a，即最优解一定含结束最早的活动



后续，A'=A-{1},一定是E'={si>f1}的最优解

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。

4.2 贪心算法的基本要素

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。

4.2 贪心算法的基本要素

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个经典的组合优化问题，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，$1 \le i \le n$。

- 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。

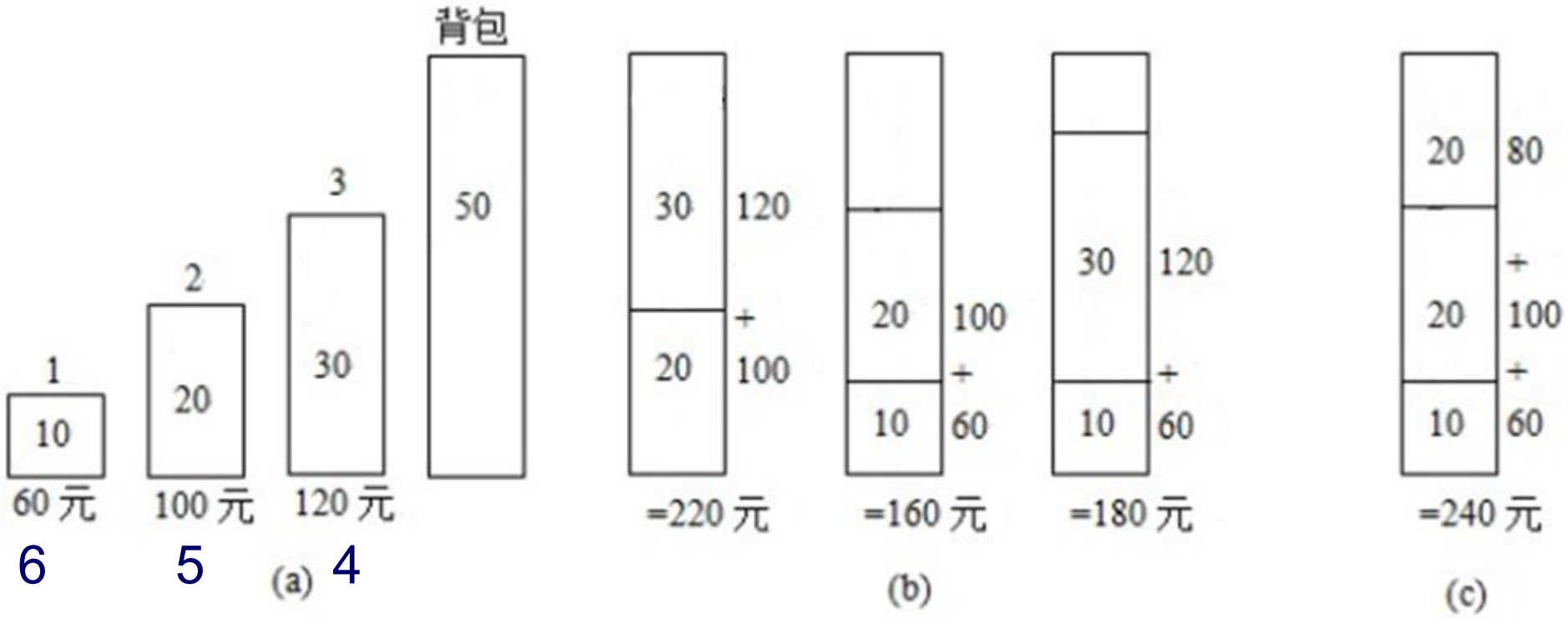
具体算法可描述如下页：

4.2 贪心算法的基本要素

```
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。

贪心选择对0-1 背包不适用



0-1背包

背包问题

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，动态规划算法的确可以有效地解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。

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

$$\max \sum_{i=1}^{n} x_i \qquad \text{目标函数}$$

$$\sum_{i=1}^{n} w_i x_i \le c \qquad \text{约束条件}$$

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。具体算法描述如下页。

4.3 最优装载

```
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)。**

最优装载问题贪心算法的正确性证明

命题： 对装载问题任何规模为n 的输入，算法得到最优解.

对问题规模归纳，设集装箱从轻到重记为$1, 2, \ldots, n$.

证：k=1, 只有1个箱子，算法显然正确.

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

只有k个元素，可用贪心法

由归纳假设，对于$N' = \{2,3,\ldots,k+1\}$，$C' = C - w_1$, 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。

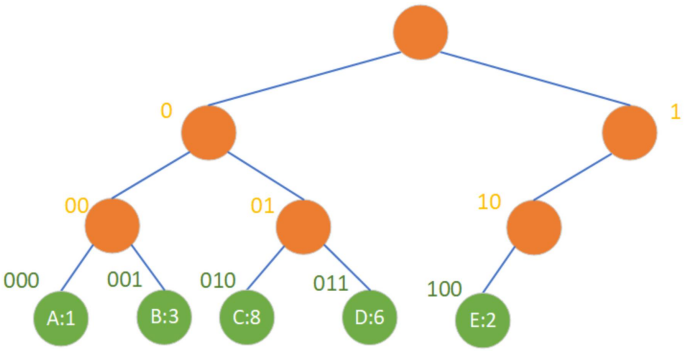
4.4 哈夫曼编码

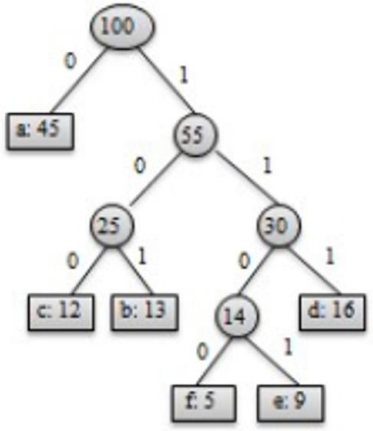
数据文件100,000个字符，以6个字符形式出现。

	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

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：

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

$f(c)$频率分布，$d_T(c)$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。

4.4 哈夫曼编码

2 构造哈夫曼编码

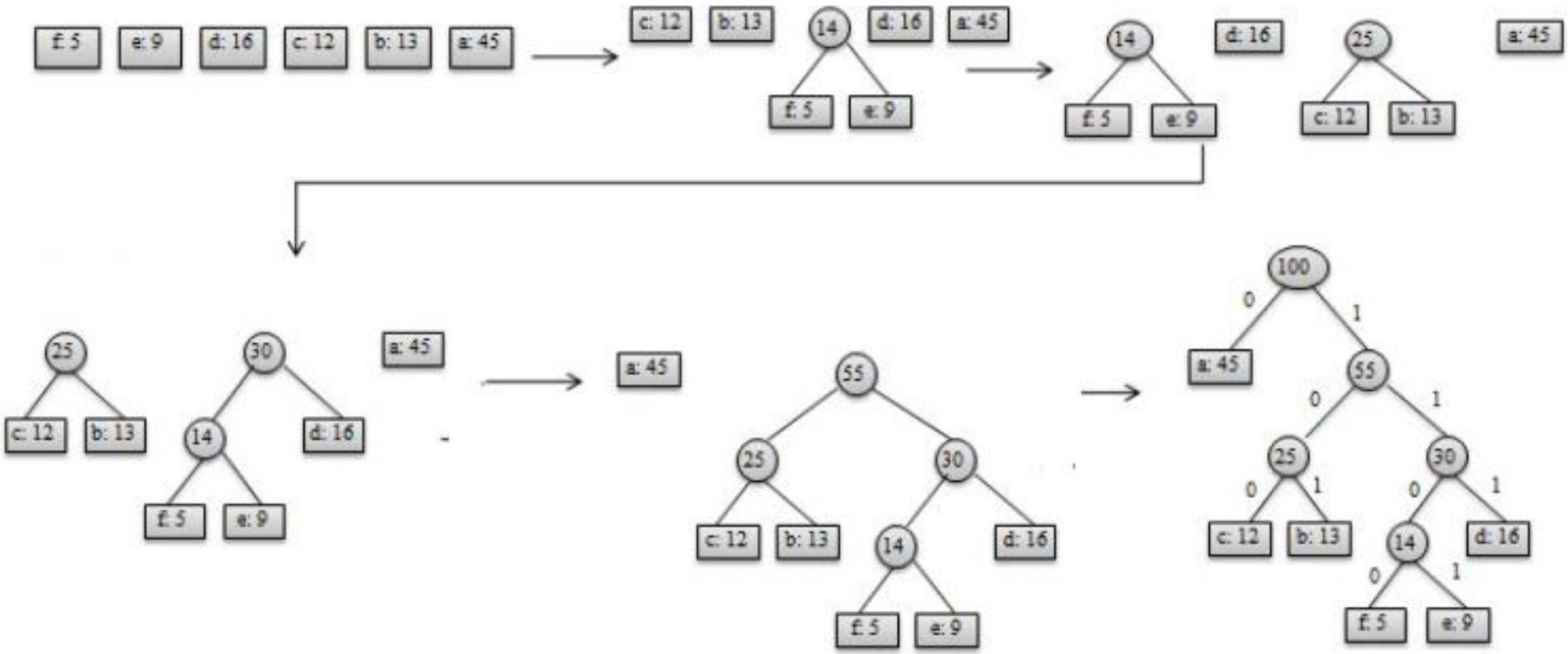
- 哈夫曼提出构造最优前缀码的贪心算法，由此产生的编码方案称为哈夫曼编码。
- 哈夫曼算法以自底向上的方式构造表示最优前缀码的二叉树T。
- 算法以|C|个叶结点开始，执行|C|－1次的“合并”运算后产生最终所要求的树T。

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



- 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) 。

4.4 哈夫曼编码

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

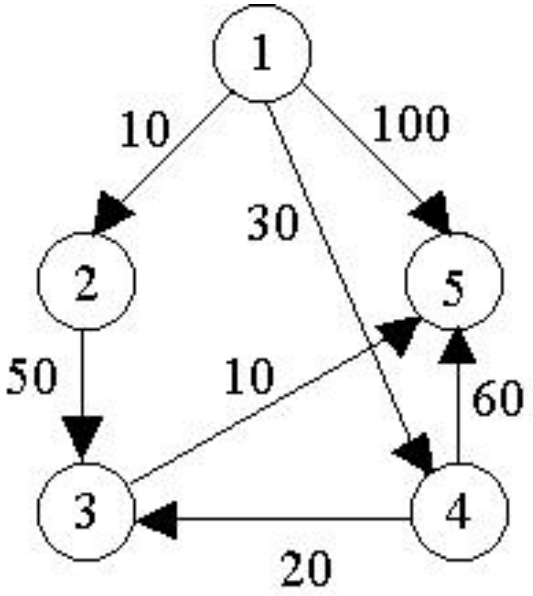
(2) 最优子结构性质

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。

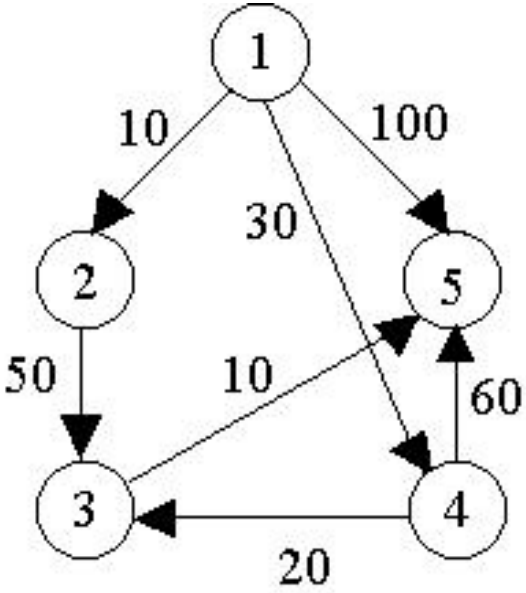


4.5 单源最短路径

- 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。

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

4.5 单源最短路径

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

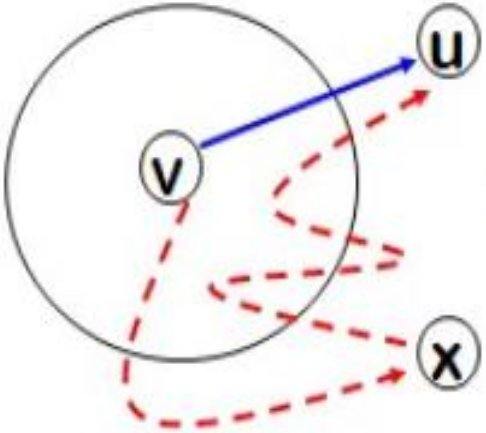
(3)计算复杂性

4.5 单源最短路径

(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]<=dist[u]与u是当前贪心选择矛盾！

4.5 单源最短路径

(2)最优子结构性质

如果S(i,j)={V_i....V_k..V_s...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)={Vi....Vk..Vs...Vj}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)+S'(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。

4.5 单源最短路径

(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$O(n^2)$。