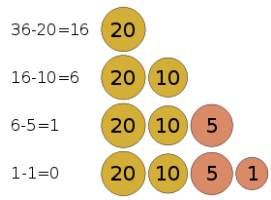
《算法设计与分析》
第4.1讲 贪心法(1)

山东师范大学信息科学与工程学院
段会川
2014年10月

目录

贪心法—找零钱问题

- 问题：
  - 有币值20、10、5、1元的零钱若干
  - 用这些零钱组成36元，最少要多少张？
- 贪心求解



贪心法—找零钱问题

- 普适问题：
  - 有币值x1>x2> … >xn元的零钱若干
  - 用这些零钱组成y元，最少要多少张？
- 贪心求解

```
1. for i = 1 to n
2.     c[i] = y/x[i]
3.     y = y%x[i]
4. end for
5. return y=0
```


贪心法—找零钱问题

- 有些问题用贪心法找到的不是最优解，如：
  - 币值：10、8、1
  - 钱数：56、57
- 有些问题用贪心法找不到解，如：
  - 币值：25、10、4
  - 钱数：41

贪心法—概述

- A greedy algorithm always makes the choice that looks best at the moment. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.

Introduction to Algorithms

- 贪心算法总是基于当前状况做出看起来最好的选择，也就是说，它期望从局部最优的选择获取全局最优的解。

算法导论

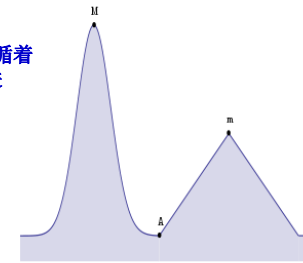
贪心法—概述

在众多的算法设计策略中，贪心法可以算得上是最接近人们日常思维的一种解题策略，它以其简单、直接和高效而受到重视。尽管该方法并不是从整体最优方面来考虑问题，而是从某种意义上的局部最优角度做出选择，但对范围相当广泛的许多实际问题它通常都能产生整体最优解，如单源最短路径问题、最小生成树等。在一些情况下，即使采用贪心法不能得到整体最优解，但其最终结果却是最优解的很好近似解。正是基于此，该算法在对 NP 完全问题的求解中发挥着越来越重要的作用。

P34

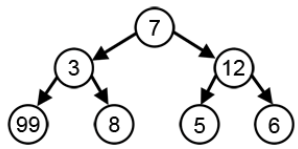
贪心法—图示

- 贪心法可能仅找到局部最优
  - 求右图中的最大值
  - 从A点出发，贪心法循着当前最优的方向前进
  - 最终到达m



贪心法—动画

- 贪心法可能仅找到局部最优
  - 求树中结点值和最大的路径
  - 从根出发，贪心法循着当前最优的方向前进
  - 最终得到“最优”路径的上结点的和为25



贪心法—基本思想

贪心法是一种稳扎稳打的算法，它从问题的某一个初始解出发，在每一个阶段都根据贪心策略来做出当前最优的决策，逐步逼近给定的目标，尽可能快地求得更好的解。当达到算法中的某一步不能再继续前进时，算法终止。贪心法可以理解为以逐步的局部最优，达到最终的全局最优。

从算法的思想中，很容易得出以下结论：

（1）贪心法的精神是“今朝有酒今朝醉”。每个阶段面临选择时，贪心法都做出对眼前来讲是最有利的选择，不考虑该选择对将来是否有不良影响。

（2）每个阶段的决策一旦做出，就不可更改，该算法不允许回溯。

P34

贪心法—基本思想

（3）贪心法是根据贪心策略来逐步构造问题的解。如果所选的贪心策略不同，则得到的贪心算法就不同，贪心解的质量当然也不同。因此，该算法的好坏关键在于正确地选择贪心策略。

贪心策略是依靠经验或直觉来确定一个最优解的决策。该策略一定要精心确定，且在使用之前最好对它的可行性进行数学证明，只有证明其能产生问题的最优解后再使用，不要被表面上看似正确的贪心策略所迷惑。

（4）贪心法具有高效性和不稳定性，因为它可以非常迅速地获得一个解，但这个解不一定是最优解，即便不是最优解，也一定是最优解的近似解。

P35

贪心法—基本要素

1. 最优子结构性质

当一个问题的最优解一定包含其子问题的最优解时，称此问题具有最优子结构性质。换句话说，一个问题能够分解成各个子问题来解决，通过各个子问题的最优解能递推到原问题的最优解。那么原问题的最优解一定包含各个子问题的最优解，这是能够采用贪心法来求解问题的关键。因为贪心法求解问题的流程是依序研究每个子问题，然后综合得出最后结果。而且，只有拥有最优子结构性质才能保证贪心法得到的解是最优解。

在分析问题是否具有最优子结构性质时，通常先设出问题的最优解，给出子问题的解一定是最优的结论。然后，采用反证法证明“子问题的解一定是最优的”结论成立。证明思路是：设原问题的最优解导出的子问题的解不是最优的，然后在这个假设下可以构造出比原问题的最优解更好的解，从而导致矛盾。

最优子结构指的是局部最优解能决定全局最优解的问题结构。即问题能够分解成子问题系列，而子问题的最优解能递推得到整个问题的最优解。 *P35*

贪心法—基本要素

2. 贪心选择性质

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获得，即通过一系列的逐步局部最优选择使得最终的选择方案是全局最优的。其中每次所做的选择，可以依赖于以前的选择，但不依赖于将来所做的选择。

可见，贪心选择性质所做的是一个非线性的子问题处理流程，即一个子问题并不依赖于另一个子问题，但是子问题间有严格的顺序性。

在实际应用中，至于什么问题具有什么样的贪心选择性质是不确定的，需要具体问题具体分析。对于一个具体问题，要确定它是否具有贪心选择性质，必须证明每一步所做的贪心选择能够最终导致问题的一个整体最优解。

P35

贪心法—基本步骤

利用贪心法求解问题的过程通常包含如下三个步骤：

(1) 分解：将原问题分解为若干个相互独立的阶段。

(2) 解决：对于每个阶段依据贪心策略进行贪心选择，求出局部的最优解。

(3) 合并：将各个阶段的解合并为原问题的一个可行解。

P36

贪心法—基本算法

```
Greedy(A,n)
{
  //A[0:n-1]包含 n 个输入，即 A 是问题的输入集合
  将解集合 solution 初始化为空;
  for(i=0; i<n; i++)                    //原问题分解为 n 个阶段
    {
      x=select(A);                      //依据贪心策略做贪心选择，求得局部最优解
      if(x 可以包含在 solution)          //判断解集合 solution 在加入 x 后是否满足约束条件
          solution=union(solution,x);   //部分局部最优解进行合并
    }
  return(解向量 solution);              //n 个阶段完成后，得到原问题的最优解
}
```

P36

贪心法—效能

- 贪心法(Greedy algorithm)，又称贪婪法
  - 贪心法可以解决一些最优化问题，如：求图中的最小生成树、求哈夫曼编码，等等
  - 对于其他问题，贪心法一般不能得到最优的答案。
  - 一旦一个问题可以通过贪心法来解决，那么贪心法一般是解决这个问题的最好办法。
  - 由于贪心法的高效性以及其所求得的答案比较接近最优结果，贪心法常用作辅助算法或者直接解决一些要求结果不特别精确的问题。

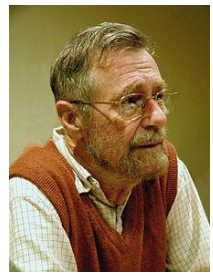
贪心法—总结

- 主要组成元素
  1. 一个待求解的候选集合
  2. 一个能够将最优候选元素加入到解集中的选择函数
  3. 一个决定候选元素是否可以加入到解集中的可行性函数
  4. 一个给解或部分解赋予一个值的目标函数
  5. 一个示出是否找到最终解的函数
- 基本过程
  1. 从问题的某一初始解出发；
  2. 当能向最优解前进一步 时 ，求出可行解的一个解元素；
  3. 将所有解元素组合成问题的一个可行解。

目录

Edsger W. Dijkstra

- Dijkstra最短路径算法是由荷兰计算机科学家艾兹赫尔·戴克斯特拉于1956年提出，1959年发表的
  - Dijkstra获得了1972年的图灵奖
  - 结构化程序设计之父
  - Go To Statement Considered Harmful (1968)



Edsger Wybe Dijkstra
1930.5.11-2002.8.6

Dijkstra最短路径算法—概述

- Dijkstra最短路径算法是一个解决非负权值的图中单源最短路径问题(即从一个给定顶点到所有其它顶点)的一个权威算法。
  - 该算法是网络中距离向量路由所采用的算法
  - 它还被广泛地作为其它图论算法的基础

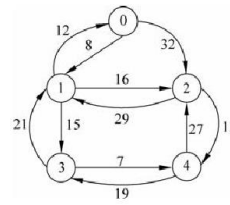
Dijkstra算法思想

对于一个具体的单源最短路径问题，如何求得该最短路径呢？一个传奇人物的出现使得该问题迎刃而解，他就是迪杰斯特拉(Dijkstra)。他提出按各个顶点与源点之间路径长度的递增次序，生成源点到各个顶点的最短路径的方法，即先求出长度最短的一条路径，再参照它求出长度次短的一条路径，以此类推，直到从源点到其他各个顶点的最短路径全部求出为止，该算法俗称 Dijkstra 算法。Dijkstra 对于它的算法是这样说的："这是我自己提出的第一个图问题，并且解决了它。令人惊奇的是我当时并没有发表。但这在那个时代是不足为奇的，因为那时，算法基本上不被当作一种科学研究的主题。"

P39

Dijkstra算法设计

假定源点为 u。顶点集合 V 被划分为两部分：集合 S 和 $V-S$，其中 S 中的顶点到源点的最短路径的长度已经确定，集合 $V-S$ 中所包含的顶点到源点的最短路径的长度待定，称从源点出发只经过 S 中的点到达 $V-S$ 中的点的路径为特殊路径。Dijkstra 算法采用的贪心策略是选择特殊路径长度最短的路径，将其相连的 $V-S$ 中的顶点加入到集合 S 中。



P40

Dijkstra算法求解步骤

步骤 1：设计合适的数据结构。设置带权邻接矩阵 C，即如果$<u,x>\in E$，令 $C[u][x]=<u,x>$的权值，否则，$C[u][x]=\infty$；采用一维数组 dist 来记录从源点到其他顶点的最短路径长度，例如 $dist[x]$表示源点到顶点 x 的路径长度；采用一维数组 p 来记录最短路径。

步骤 2：初始化。令集合 $S=\{u\}$，对于集合 $V-S$ 中的所有顶点 x，设置 $dist[x]=C[u][x]$(注意，x 只是一个符号，它可以表示集合 $V-S$ 中的任一个顶点)；如果顶点 i 与源点相邻，设置 $p[i]=u$，否则 $p[i]=-1$。

步骤 3：在集合 $V-S$ 中依照贪心策略来寻找使得 $dist[x]$具有最小值的顶点 t，即 $dist[t]=\min\{dist[x]|x\in(V-S)\}$，满足该公式的顶点 t 就是集合 $V-S$ 中距离源点 u 最近的顶点。

步骤 4：将顶点 t 加入集合 S 中，同时更新集合 $V-S$。

步骤 5：如果集合 $V-S$ 为空，算法结束；否则，转步骤 6。

步骤 6：对集合 $V-S$ 中的所有与顶点 t 相邻的顶点 x，如果 $dist[x]>dist[t]+C[t][x]$，则 $dist[x]=dist[t]+C[t][x]$并设置 $p[x]=t$。转步骤 3。

P40

Dijkstra最短路径算法—伪代码

- 输入：图G(V,E)，权矩阵W，顶点s
- 输出：s到其余各顶点的最短路径和距离

```
1.  for v in V
2.      dist(v) = ∞; prev(v) = nil
3.  dist(s) = 0; U = ∅
4.  H = makequeque(V) //以距离为键值
5.  while H is not empty
6.      v = deletemin(H)
7.      U = U ∪ {v}
8.      for all edges (v,u) ∈ E and u ∉ U
9.          if dist(v) + w(v,u) < dist(u)
10.             dist(u) = dist(v) + w(v,u)
11.             prev(u) = v
12.             decresekey(H,u)
```

Dijkstra最短路径算法—复杂度

- 算法的基本复杂度为$O(|V|\cdot em_Q+|E|\cdot dk_Q)$
  - em_Q—从距离队列中获取最小距离结点(extract minimum)的代价
  - dk_Q—减小距离队列一个元素键值(delete key)的代价
- 用基本线性表实现距离队列
  - $O(|V|^2+|E|)=O(|V|^2)$
- 运用二叉最小堆实现距离队列
  - 算法复杂度改进为$O((|E|+|V|)\log|V|)$
- 运用Fibonacci堆实现距离队列
  - 算法复杂度改进为$O(|E|+|V|\log|V|)$

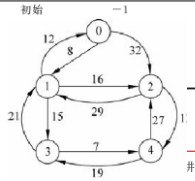
Dijkstra算法—举例

表 2-2 Dijkstra 算法的求解过程

	S	V−S	dist[1]	dist[2]	dist[3]	dist[4]	t
初始	{0}	{1,2,3,4}	8	32	∞	∞	1
1	{0,1}	{2,3,4}	—	24	23	∞	3
2	{0,1,3}	{2,4}	—	24	—	30	2
3	{0,1,3,2}	{4}	—	—	—	30	4
4	{0,1,3,2,4}	{}					

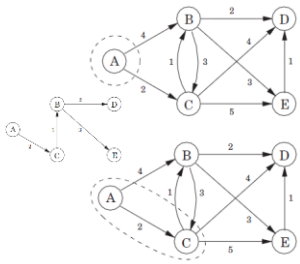
表 2-3 前驱数组 p 变化情况表

	0	1	2	3	4
初始	−1	0	0	−1	−1
		0	1	1	−1
		0	1	1	3
		0	1	1	3
		0	1	1	3



P40

Dijkstra最短路径算法—示例

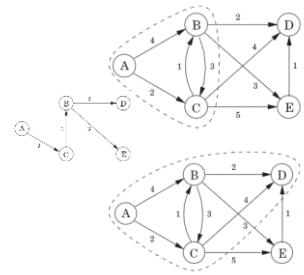


A: 0	D: ∞
B: 4	E: ∞
C: 2	

A: 0	D: 6
B: 3	E: 7
C: 2	

P126，图4-9

Dijkstra最短路径算法—示例

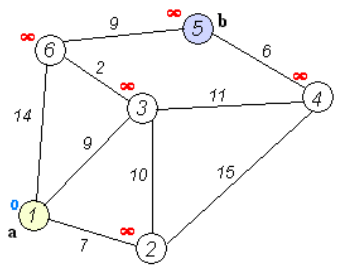


A: 0	D: 5
B: 3	E: 6
C: 2	

A: 0	D: 5
B: 3	E: 6
C: 2	

P126，图4-9

Dijkstra最短路径算法—演示



Dijkstra最短路径算法—演示

Dijkstra算法正确性—贪心选择性质

Dijkstra 算法是应用贪心法设计策略的又一个典型例子。它所做的贪心选择是从集合 $V-S$ 中选择具有最短路径的顶点 t，从而确定从源点 u 到 t 的最短路径长度 dist[t]。这种贪心选择为什么能得到最优解呢？换句话说，为什么从源点到 t 没有更短的其他路径呢？

事实上，假设存在一条从源点 u 到 t 且长度比 dist[t]更短的路，设这条路径初次走出 S 之外到达的顶点为 $x \in V-S$，然后徘徊于 S 内外若干次，最后离开 S 到达 t。

在这条路径上，分别记 $d(u,x)$、$d(x,t)$ 和 $d(u,t)$为源点 u 到顶点 x，顶点 x 到顶点 t 和源点 u 到顶点 t 的路径长度，那么，依据假设容易得出：

$$\text{dist}[x] \leqslant d(u,x)$$

$$d(u,x)+d(x,t)=d(u,t)<\text{dist}[t]$$

利用边权的非负性，可知 $d(x,t) \geqslant 0$，从而推得 dist[x]<dist[t]。此与前提矛盾，从而证明了 dist[t]是从源点到顶点 t 的最短路径长度。

P42

Dijkstra算法正确性—最优子结构性质

要完成 Dijkstra 算法正确性的证明，还必须证明最优子结构性质，即算法中确定的 dist[t]确实是当前从源点到顶点 t 的最短路径长度。为此，只要考察算法在添加 t 到 S 中后，dist[t]的值所起的变化就行了。将添加 t 之前的 S 称为老 S。当添加了 t 之后，可能出现一条到顶点 j 的新的特殊路径。如果这条新路径是先经过老 S 到达顶点 t，然后从 t 经一条边直接到达顶点 j，则这条路径的最短长度是 dist[t]+C[t][j]。这时，如果 dist[t]+C[t][j]<dist[j]，则算法中用 dist[t]+C[t][j]作为 dist[j]的新值。如果这条新路径经过老 S 到达 t 后，不是从 t 经一条边直接到达 j，而是先回到老 S 中某个顶点 x，最后才到达顶点 j，那么由于 x 在老 S 中，因此 x 比 t 先加入 S，故从源点到 x 的路径长度比从源点到 t，再从 t 到 x 的路径长度小。于是当前 dist[j]的值小于从源点经 x 到 j 的路径长度，也小于从源点经 t 和 x，最后到达 j 的路径长度。因此，在算法中不必考虑这种路径。可见，无论算法中 dist[t]的值是否有变化，它总是关于当前顶点集 S 到顶点 t 的最短路径长度。

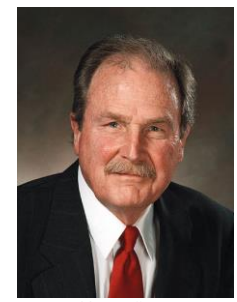
P43

目录

- 贪心法基础
- Dijkstra最短路径算法
- Huffman编码
- Huffman编码—《导论》表述
- 作业

David A. Huffman

- Huffman编码是一种用于无损数据压缩的熵编码(权编码)算法。
  - 由David A. Huffman于1952年在麻省理工攻读博士时所发明的，论文题为《一种构建极小冗余编码的方法》(A Method for the Construction of Minimum-Redundancy Codes)



David A. Huffman
1925.8.9-1999.10.7

哈夫曼编码(Huffman Coding)—概述

- Huffman编码使用变长编码表对源符号进行编码，其中变长编码表是通过一种评估来源符号出现机率的方法得到的
  - 出现机率高的字母使用较短的编码，出现机率低的则使用较长的编码，这使编码之后的字符串的平均长度降低，从而达到无损压缩数据的目的。

哈夫曼编码(Huffman Coding)—概述

(1) 固定长度编码方法。

假设所有字符的编码都等长，则表示 n 个不同的字符需要 $\log n$ 位，ASCII 码就是固定长度的编码。如果每个字符的使用频率相等的话，固定长度编码是空间效率最高的方法。但在信息的实际处理过程中，每个字符的使用频率有着很大的差异，现在的计算机键盘中的键的不规则排列，就是源于这种差异。

(2) 不等长度编码方法。

不等长编码方法是今天广泛使用的文件压缩技术，其思想是：利用字符的使用频率来编码，使得经常使用的字符编码较短，不常使用的字符编码较长。这种方法既能节省磁盘空间，又能提高运算与通信速度。

P43

哈夫曼编码(Huffman Coding)—概述

(1) 固定长度编码方法。

假设所有字符的编码都等长，则表示 n 个不同的字符需要 $\log n$ 位，ASCII 码就是固定长度的编码。如果每个字符的使用频率相等的话，固定长度编码是空间效率最高的方法。但在信息的实际处理过程中，每个字符的使用频率有着很大的差异，现在的计算机键盘中的键的不规则排列，就是源于这种差异。

(2) 不等长度编码方法。

不等长编码方法是今天广泛使用的文件压缩技术，其思想是：利用字符的使用频率来编码，使得经常使用的字符编码较短，不常使用的字符编码较长。这种方法既能节省磁盘空间，又能提高运算与通信速度。

P43

哈夫曼编码(Huffman Coding)—概述

但采用不等长编码方法要注意一个问题：任何一个字符的编码都不能是其他字符编码的前缀(即前缀码特性)，否则译码时将产生二义性。那么如何来设计前缀编码呢？我们自然地想到利用二叉树来进行设计。具体做法是：约定在二叉树中用叶子结点表示字符，从根结点到叶子结点的路径中，左分支表示“0”，右分支表示“1”。那么，从根结点到叶子结点的路径分支所组成的字符串作为该叶子结点字符的编码，可以证明这样的编码一定是前缀编码，这棵二叉树即为编码树。

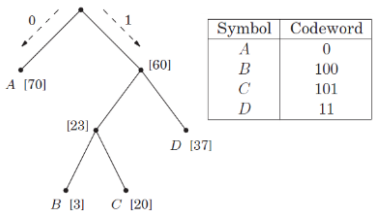
剩下的问题是怎样保证这样的编码树所得到的编码总长度最小？哈夫曼提出了解决该问题的方法，由此产生的编码方案称为哈夫曼算法。

P43

Huffman编码—前缀码示例

- 任一无前缀编码都可以表示为一棵满二叉树

Symbol	Frequency
A	70 million
B	3 million
C	20 million
D	37 million



Symbol	Codeword
A	0
B	100
C	101
D	11

P57，图5-10

Huffman编码—构造思想

该算法的基本思想是以字符的使用频率作为权构建一棵哈夫曼树，然后利用哈夫曼树对字符进行编码，俗称哈夫曼编码。具体来讲，是将所要编码的字符作为叶子结点，该字符在文件中的使用频率作为叶子结点的权值，以自底向上的方式、通过执行 $n-1$ 次的“合并”运算后构造出最终所要求的树，即哈夫曼树，它的核心思想是让权值大的叶子离根最近。

那么，在执行合并运算的过程中，哈夫曼算法采取的贪心策略是每次从树的集合中取出双亲为 0 且权值最小的两棵树作为左、右子树，构造一棵新树，新树根结点的权值为其左右孩子结点权之和，将新树插入到树的集合中。依照该贪心策略，执行 $n-1$ 次合并后最终可构造出哈夫曼树。

P44

Huffman编码—算法设计

步骤 1：确定合适的数据结构。由于哈夫曼树中没有度为 1 的结点，则一棵有 n 个叶子结点的哈夫曼树共有 $2n-1$ 个结点；构成哈夫曼树后，为求编码需从叶子结点出发走一条从叶子到根的路径；而译码则需从根出发走一条从根到叶子的路径。对每个结点而言，既需要知道双亲的信息，又需要知道孩子结点的信息，因此数据结构的选择要考虑这方面的情况。

步骤 2：初始化。构造 n 棵结点为 n 个字符的单结点树集合 $F=\{T_1,T_2,\cdots,T_n\}$，每棵树中只有一个带权的根结点，权值为该字符的使用频率。

步骤 3：如果 F 中只剩下一棵树，则哈夫曼树构造成功，转步骤 6；否则，从集合 F 中取出双亲为 0 且权值最小的两棵树 T_i 和 T_j，将它们合并成一棵新树 Z_k，新树以 T_i 为左儿子，T_j 为右儿子(反之也可以)。新树 Z_k 的根结点的权值为 T_i 与 T_j 的权值之和。

步骤 4：从集合 F 中删去 T_i、T_j，加入 Z_k。

步骤 5：重复步骤 3 和 4。

步骤 6：从叶子结点到根结点逆向求出每个字符的哈夫曼编码(约定左分支表示字符“0”，右分支表示字符“1”)。则从根结点到叶子结点路径上的分支字符组成的字符串即为叶子字符的哈夫曼编码。算法结束。

P44

目录

Huffman编码—示例

赫夫曼编码可以很有效地压缩数据：通常可以节省 20%～90%的空间，具体压缩率依赖于数据的特性。我们将待压缩数据看做字符序列。根据每个字符的出现频率，赫夫曼贪心算法构造出字符的最优二进制表示。

假定我们希望压缩一个 10 万个字符的数据文件。图 16-3 给出了文件中所出现的字符和它们的出现频率。也就是说，文件中只出现了 6 个不同字符，其中字符 a 出现了 45 000 次。

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

图 16-3　一个字符编码问题。一个 100 000 个字符的文件，只包含 a～f 6 个不同字符，出现频率如上表所示。如果为每个字符指定一个 3 位的码字，我们可以将文件编码为 300 000 位的长度。但使用上表所示的变长编码，我们可以仅用 224 000 位编码文件

《导论》中文3版P245

Huffman编码—示例

我们有很多方法可以表示这个文件的信息。在本节中，我们考虑一种**二进制字符编码**(或简称**编码**)的方法，每个字符用一个唯一的二进制串表示，称为**码字**。如果使用**定长编码**，需要用 3 位来表示 6 个字符：a=000，b=001，…，f=101。这种方法需要 300 000 个二进制位来编码文件。是否有更好的编码方案呢？

变长编码(variable-length code)可以达到比定长编码好得多的压缩率，其思想是赋予高频字符短码字，赋予低频字符长码字。图 16-3 显示了本例的一种变长编码：1 位的串 0 表示 a，4 位的串 1100 表示 f。因此，这种编码表示此文件共需

$$(45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4) \cdot 1\,000 = 224\,000 \text{ 位}$$

与定长编码相比节约了 25%的空间。实际上，我们将看到，这是此文件的最优字符编码。

《导论》中文3版P245

Huffman编码—前缀码

我们这里只考虑所谓**前缀码**(prefix code)⊖，即没有任何码字是其他码字的前缀。虽然我们这里不会证明，但与任何字符编码相比，前缀码确实可以保证达到最优数据压缩率，因此我们只关注前缀码，不会丧失一般性。

任何二进制字符码的编码过程都很简单，只要将表示每个字符的码字连接起来即可完成文件压缩。例如，使用图 16-3 所示的变长前缀码，我们可以将 3 个字符的文件 abc 编码为 0 · 101 · 100=0101100，“ · ”表示连结操作。

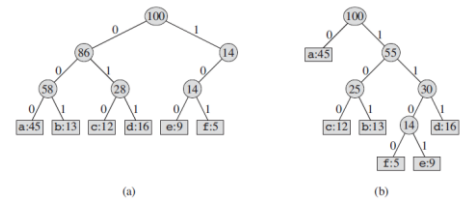
前缀码的作用是简化解码过程。由于没有码字是其他码字的前缀，编码文件的开始码字是无歧义的。我们可以简单地识别出开始码字，将其转换回原字符，然后对编码文件剩余部分重复这种解码过程。在我们的例子中，二进制串 001011101 可以唯一地解析为 0 · 0 · 101 · 1101，解码为 aabe。

⊖ 可能“无前缀码”是一个更好的名字，但在相关文献中，“前缀码”是一致认可的标准术语。

《导论》中文3版P245

Huffman编码—二叉树表示

解码过程需要前缀码的一种方便的表示形式，以便我们可以容易地截取开始码字。一种二叉树表示可以满足这种需求，其叶结点为给定的字符。字符的二进制码字用从根结点到该字符叶结点的简单路径表示，其中 0 意味着“转向左孩子”，1 意味着“转向右孩子”。图 16-4 给出了两个编码示例的二叉树表示。注意，编码树并不是二叉搜索树，因为叶结点并未有序排列，而内部结点并不包含字符关键字。



《导论》中文3版P246

Huffman编码—二叉树表示

文件的最优编码方案总是对应一棵满(full)二叉树，即每个非叶结点都有两个孩子结点(参见练习 16.3-2)。前文给出的定长编码实例不是最优的，因为它的二叉树表示并非满二叉树，如图 16-4(a)所示：它包含以 10 开头的码字，但不包含以 11 开头的码字。现在我们可以只关注满二叉树了，因此可以说，若 C 为字母表且所有字符的出现频率均为正数，则最优前缀码对应的树恰有 $|C|$ 个叶结点，每个叶结点对应字母表中一个字符，且恰有 $|C|-1$ 个内部结点(参见练习 B.5-3)。

给定一棵对应前缀码的树 T，我们可以容易地计算出编码一个文件需要多少个二进制位。对于字母表 C 中的每个字符 c，令属性 $c.freq$ 表示 c 在文件中出现的频率，令 $d_T(c)$ 表示 c 的叶结点在树中的深度。注意，$d_T(c)$ 也是字符 c 的码字的长度。则编码文件需要

$$B(T) = \sum_{c \in C} c.freq \cdot d_T(c) \qquad (16.4)$$

个二进制位，我们将 $B(T)$ 定义为 T 的**代价**。

《导论》中文3版P246

Huffman编码—算法

赫夫曼设计了一个贪心算法来构造最优前缀码，被称为**赫夫曼编码**(Huffman code)。与 16.2 节中我们的观察一致，它的正确性证明也依赖于贪心选择性质和最优子结构。接下来，我们并不是先证明这些性质成立然后再设计算法，而是先设计算法。这样做可以帮助我们明确算法是如何做出贪心选择的。

在下面给出的伪代码中，我们假定 C 是一个 n 个字符的集合，而其中每个字符 $c \in C$ 都是一个对象，其属性 $c.freq$ 给出了字符的出现频率。算法自底向上地构造出对应最优编码的二叉树 T。它从 $|C|$ 个叶结点开始，执行 $|C|-1$ 个“合并”操作创建出最终的二叉树。算法使用一个以属性 $freq$ 为关键字最小优先队列 Q，以识别两个最低频率的对象将其合并。当合并两个对象时，得到的新对象的频率设置为原来两个对象的频率之和。

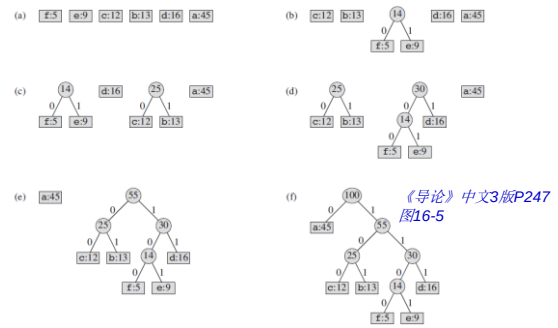
《导论》中文3版P246

Huffman编码—算法伪代码

```
1. n = |C|
2. Q = C
3. for i=1 to n-1
4.     allocate a new node z
5.     z.left = x = EXTRACT-MIN(Q)
6.     z.right = y = EXTRACT-MIN(Q)
7.     z.freq = x.freq + y.freq
8.     INSERT(Q, z)
9. return EXTRACT-MIN(Q)
```

《导论》中文3版P247

Huffman编码—算法示例



《导论》中文3版P247 图16-5

Huffman编码—算法复杂度分析

为了分析赫夫曼算法的运行时间，我们假定 Q 是使用最小二叉堆实现的(参见第 6 章)。对一个 n 个字符的集合 C，我们在第 2 行用 BUILD-MIN-HEAP 过程(参见 6.3 节)将 Q 初始化，花费时间为 $O(n)$。第 3～8 行的 **for** 循环执行了 $n-1$ 次，且每个堆操作需要 $O(\lg n)$的时间，所以循环对总时间的贡献为 $O(n\lg n)$。因此，处理一个 n 个字符的集合，HUFFMAN 的总运行时间为 $O(n\lg n)$。如果将最小二叉堆换为 van Emde Boas 树(参见第 20 章)，我们可以将运行时间减少为 $O(n\lg\lg n)$。

《导论》中文3版P248

Huffman编码—算法正确性证明

引理 16.2 令 C 为一个字母表，其中每个字符 $c\in C$ 都有一个频率 $c.freq$。令 x 和 y 是 C 中频率最低的两个字符。那么存在 C 的一个最优前缀码，x 和 y 的码字长度相同，且只有最后一个二进制位不同。

引理 16.3 令 C 为一个给定的字母表，其中每个字符 $c\in C$ 都定义了一个频率 $c.freq$。令 x 和 y 是 C 中频率最低的两个字符。令 C'为 C 去掉字符 x 和 y，加入一个新字符 z 后得到的字母表，即 $C'=C-\{x,y\}\cup\{z\}$。类似 C，也为 C'定义 $freq$，不同之处只是 $z.freq=x.freq+y.freq$。令 T'为字母表 C'的任意一个最优前缀码对应的编码树。于是我们可以将 T'中叶结点 z 替换为一个以 x 和 y 为孩子的内部结点，得到树 T，而 T 表示字母表 C 的一个最优前缀码。

定理 16.4 过程 HUFFMAN 会生成一个最优前缀码。

证明 由引理 16.2 和引理 16.3 即可得。

《导论》中文3版P248-9

作业

1. 描述贪心法的贪心选择性质和最优子结构性质
2. 描述贪心算法的基本过程
3. 给出Dijkstra算法的伪代码，说明其复杂度
4. 证明Dijkstra算法满足贪心法的贪心选择性质和最优子结构性质
5. 解释什么是前缀码
6. 给出Huffman算法的伪代码，说明其复杂度

目录