编码和数据压缩

Outline

- 导论
- 霍夫曼编码
- Ziv-Lempel压缩算法
- 其他压缩算法

导论

- 为什么需要压缩
  - 计算机和网络中存放着大量的书籍、音乐和电影
  - 但是它们往往会占用大量存储空间，而且需要大量时间从网络上下载
  - 通过压缩，文档会变得更“苗条”，从而使我们收集信息的速度变快，使CD、DVD等存储媒介上存放的信息更多

导论

- 为什么需要压缩
  - 为了满足上述要求，需要做两件事情
    - 压缩数据以便缩小它的体积
    - 解压缩以便人们正常阅读信息
  - 通常这两道工序由计算机在后台自动完成
  - 有时候需要用户明确指示计算机进行压缩，比如压缩成一个ZIP压缩包或GIF图像
  - 更多的时候计算机可自行判断什么时候该自动执行压缩，比如当你准备向网络传送数据前

固定长度编码

- 假设所有的编码都等长，那么编码n个不同的代码需要$log_2 n$的编码长度，这样的编码被称为*固定长度编码*
- ASCII码就是一个固定长度编码
- 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

如果字符出现的概率不同的时候，固定长度编码的空间效率还是很好的吗？

不等长编码

- 在实践应用中，字符出现的概率往往是不同的
  - 如在英语中字母E出现的概率最大，大概10%，而J，Q，Z却只有0.1%
- 直观上想，如果出现频率高的字母用短的编码代替，出现频率高的用稍长的编码表示，那么平均码长会更小

不等长编码

- 频率不等的字符

E	L	D	U	C	F	K	Z
120	42	42	37	32	24	7	2

- 可以根据字母的出现频率来编码，使得经常出现的字母的编码较短，反之不常出现的字母编码较长
- 数据压缩既能节省磁盘空间，又能提高运算速度

不等长编码

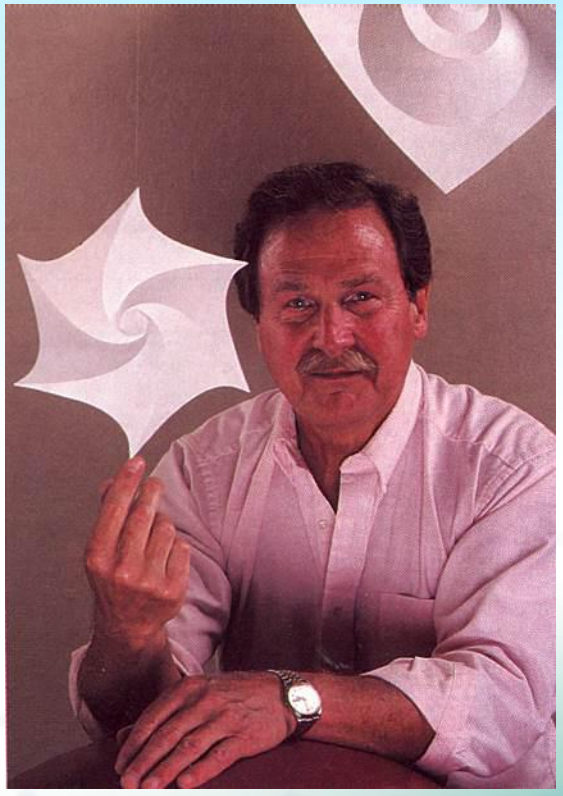
- 不等长编码是今天广泛使用的文件压缩技术的核心
- 而霍夫曼编码是最简单的文件压缩技术，它给出这种编码方法的思想

霍夫曼编码

- 小故事

1951年，霍夫曼和他在MIT信息论的同学需要选择是完成学期报告还是期末考试。导师Robert M. Fano给他们的学期报告的题目是，寻找最有效的二进制编码。由于无法证明哪个已有编码是最有效的，霍夫曼放弃对已有编码的研究，转向新的探索，最终发现了基于有序频率二叉树编码的想法，并很快证明了这个方法是最有效的。

由于这个算法，学生终于青出于蓝，超过了他那曾经和信息论创立者克劳德·香农共同研究过类似编码的导师。



前缀编码

- 如果按照下面进行编码

字符	E	L	D	U	C	F	K	Z
编码	0	1	00	01	10	11	000	001

- 读取到编码“000110”，会被译码为什么？

EEELLE　KFE　DUC　……

- 这样具有二义性的编码是无法使用的

这样的编码为什么出现这样的问题？如何避免这样的问题呢？

前缀编码

- 以上编码出现问题的原因是*编码有相同的前缀*
  - E(0)和D(00)有相同的前缀 0
  - D(00)和Z(001)有相同的前缀 00
- 因此对于不等长编码需要
  - 注意任何一个字符的编码都不是另外一个字符编码的前缀
  - 否则译码时将产生二义性

前缀编码

- 我们把编码修改为

E	L	D	U	C	F	K	Z
0	110	101	100	1110	11111	111101	111100

- 如果读到“000110”会被译码为什么？

EEEL **是唯一的！**

除了EEEL你还能想出其他的译码吗？这样的编码有什么特点？

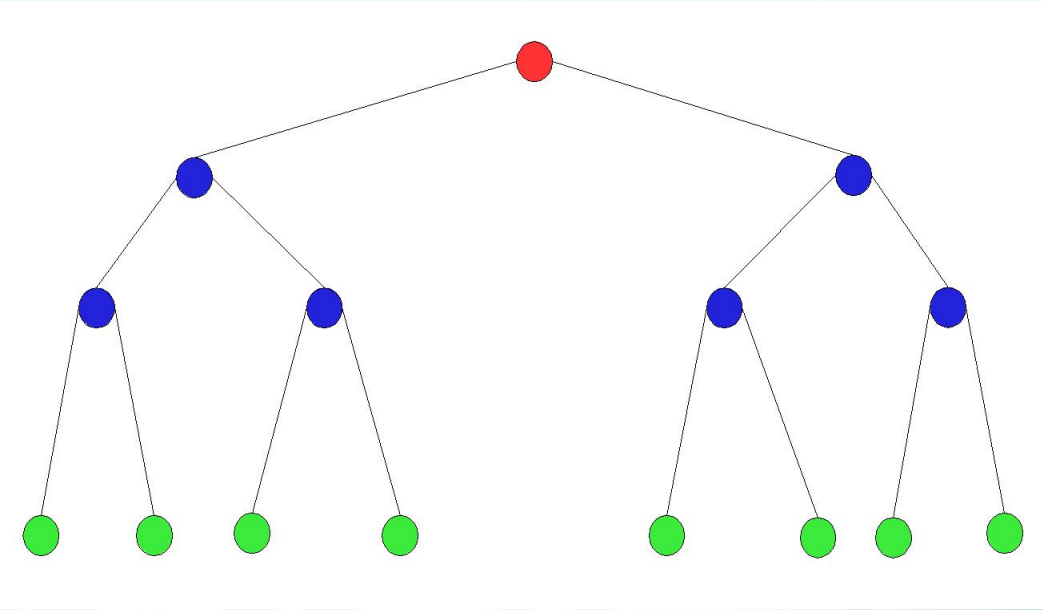
前缀编码

- 以上的编码中，任何一个字符的编码都不是另一个字符的前缀
- 这样的编码被称为***前缀编码***
- 这种前缀特性保证了代码串被译码时，不会有多种可能

那么如何设计前缀编码，并且能够使得编码的平均码长最短呢？

二叉树

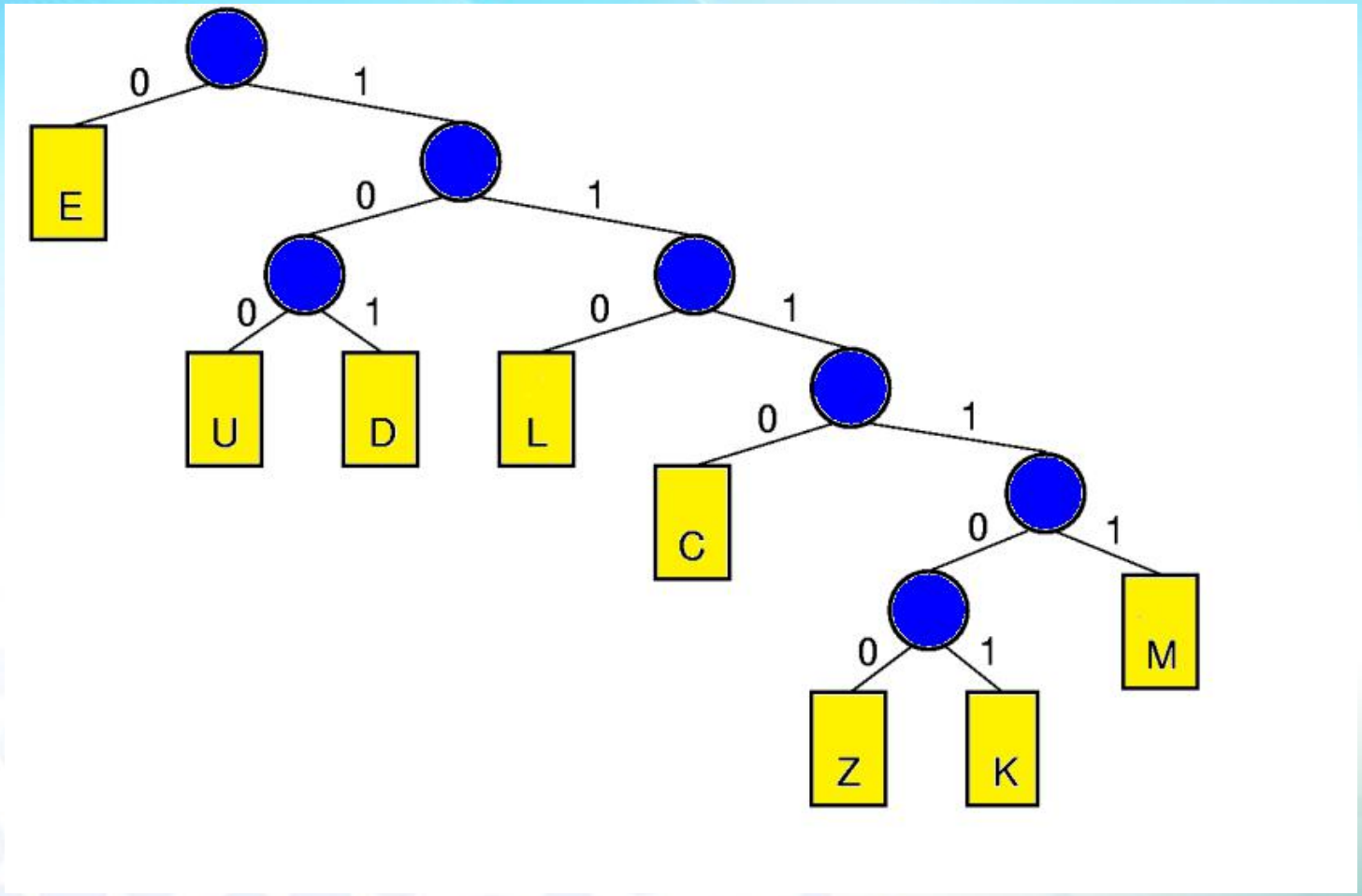
- 在计算机科学中，二叉树是每个节点最多有两个子树的有序树。
- 通常子树被称作“左子树” 和“右子树”



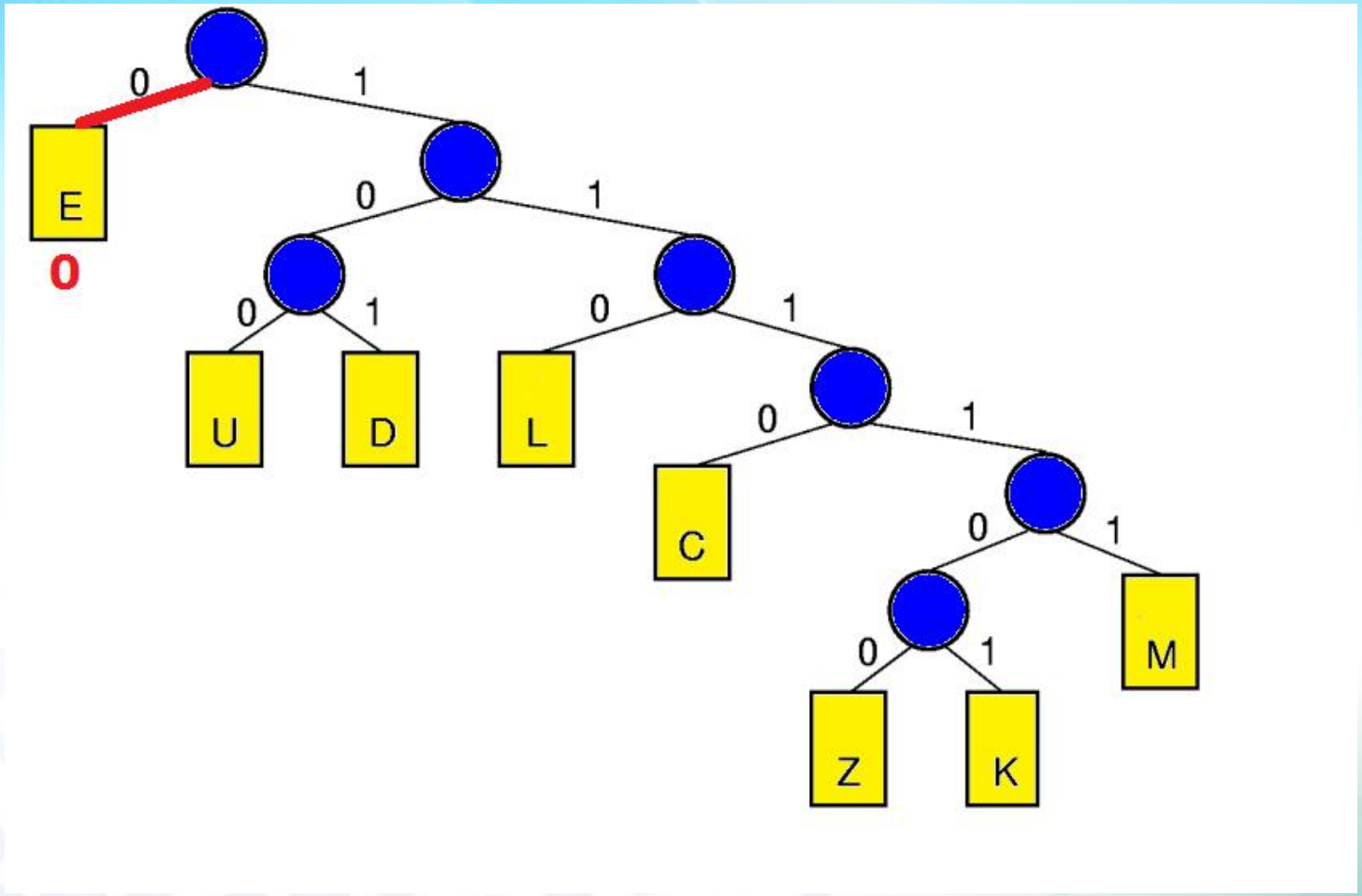
二叉树与前缀编码

- 可以利用二叉树来设计前缀编码
  - 叶结点表示字符
  - 从根结点到叶的路径中，左分支表示‘0’，右分支表示‘1’
  - 从根结点到叶结点上的路径分支所组成的字符串作为该叶结点字符的编码
  - 这样的编码一定是前缀编码

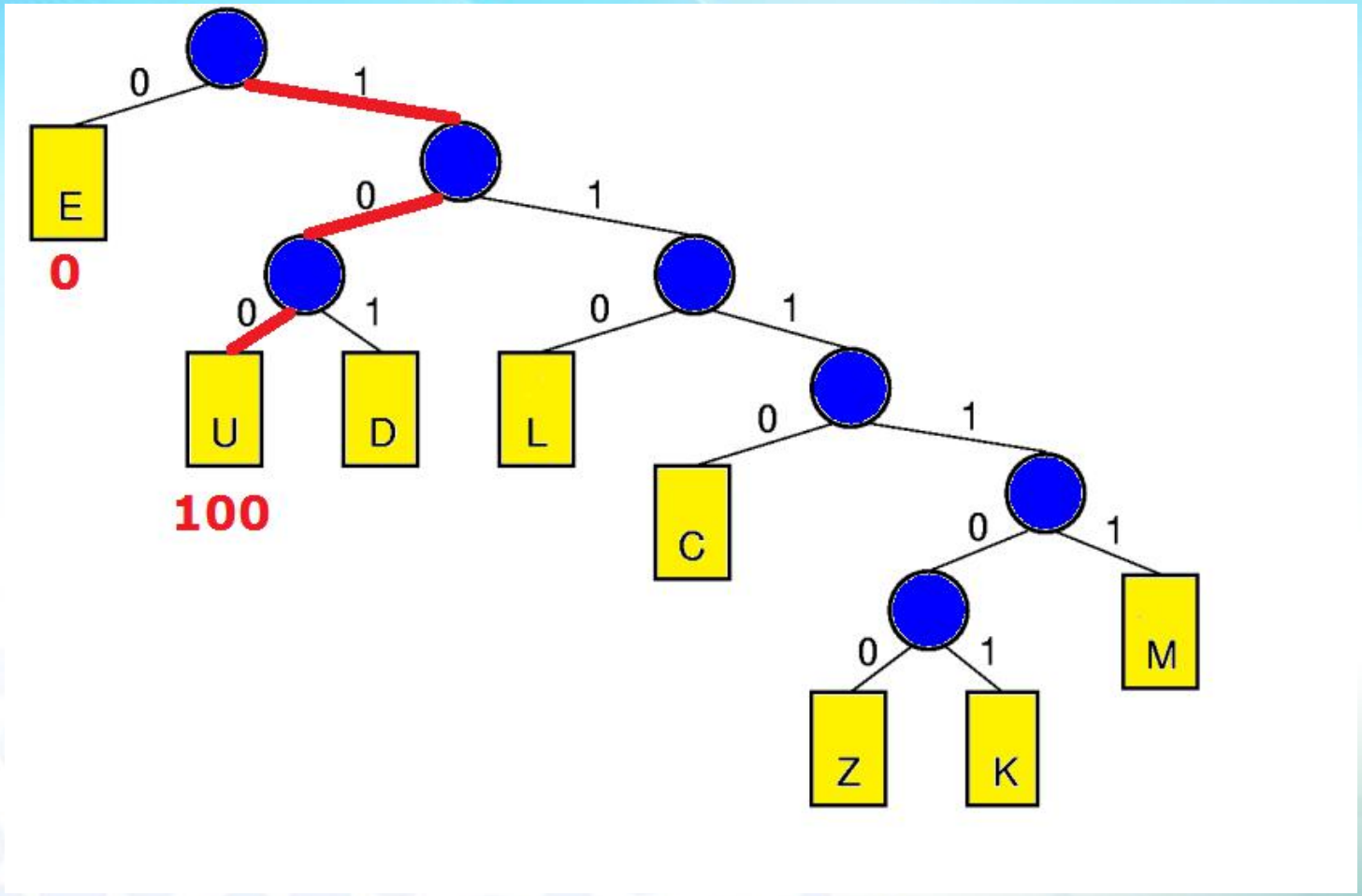
二叉树与前缀编码



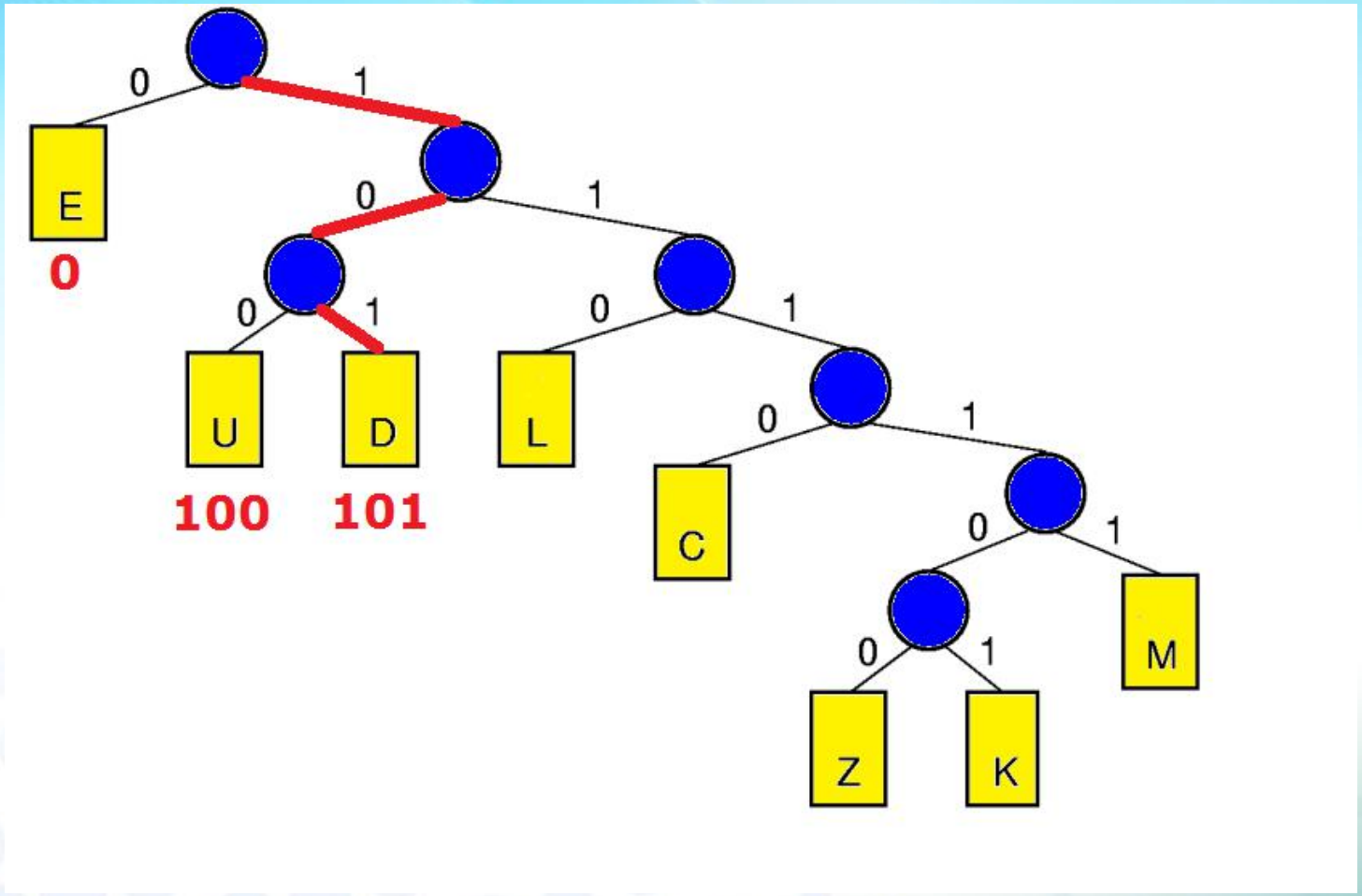
二叉树与前缀编码



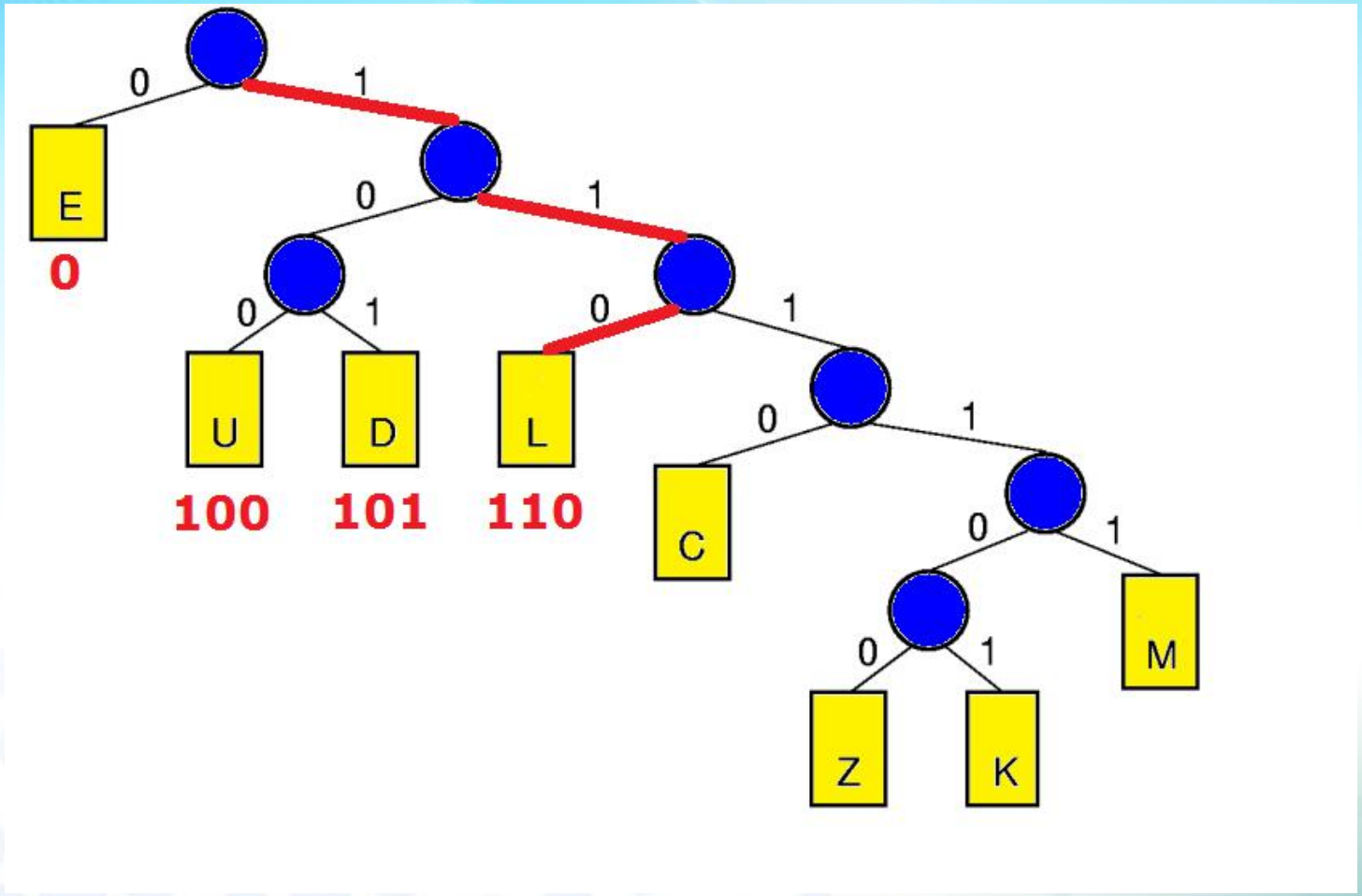
二叉树与前缀编码



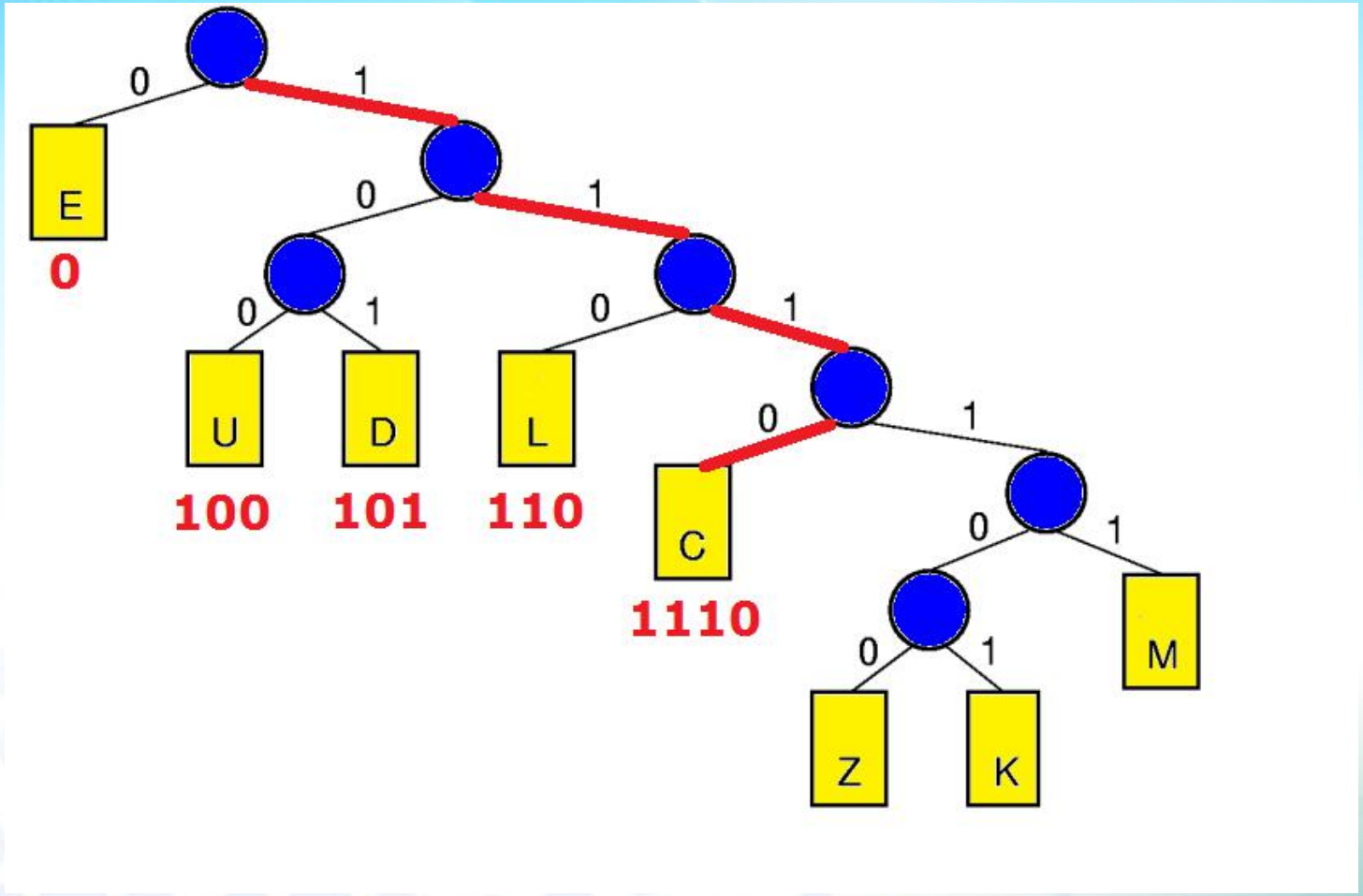
二叉树与前缀编码



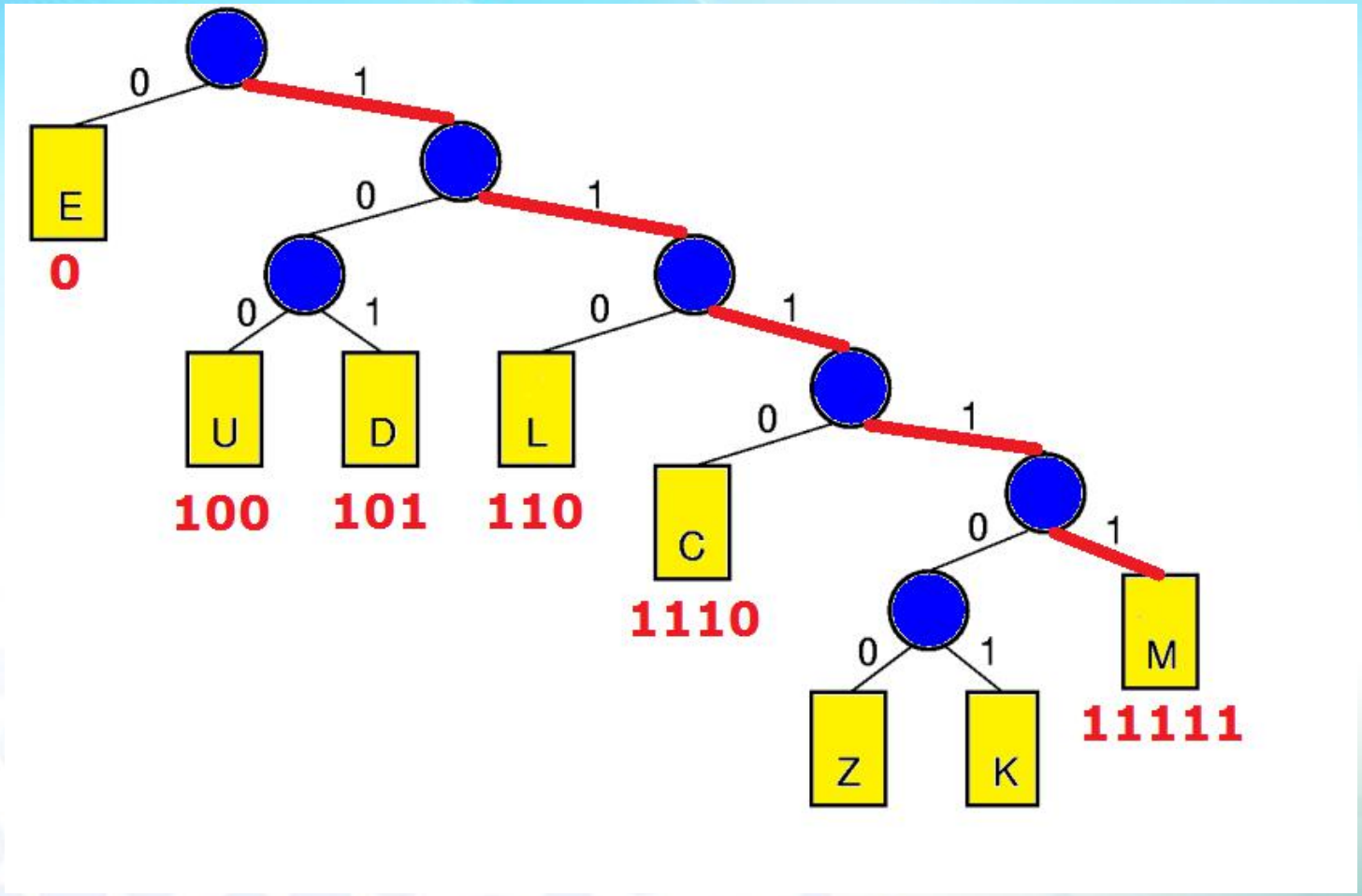
二叉树与前缀编码



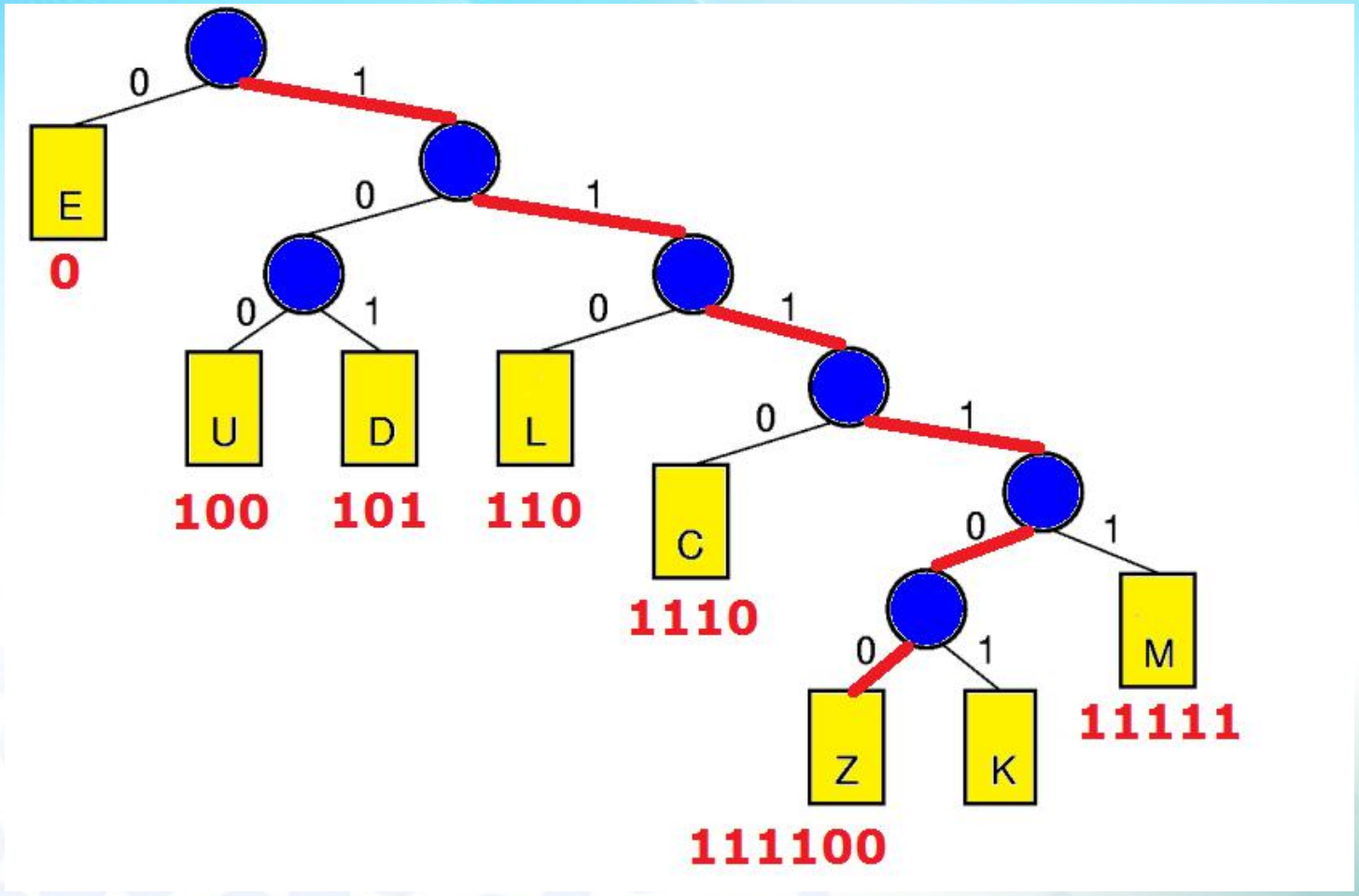
二叉树与前缀编码



二叉树与前缀编码



二叉树与前缀编码



二叉树与前缀编码

可以看到这些编码中，任何一个字符串的编码都不是另一个的前缀，因此是前缀编码

为什么二叉树得到的是前缀编码呢？

如何保证这样的编码的平均长度最小呢？

霍夫曼编码

- 霍夫曼编码将代码与字符相联系
  - 代码长度取决于对应字母的相对使用频率
  - 它是一种不等长编码
  - 如果预计的字母出现频率与实际资料显示的情况相符，那么所得代码长度将明显小于用固定长度编码获得的代码
  - 而且也可以证明，霍夫曼树获得的编码的平均长度是最短的

构建霍夫曼树

1. 首先将字符按照频率排为一列
2. 接着，拿走头两个字符，再将它们标记为霍夫曼树的树叶，将这两个树叶标记为一个分支结点的两个子女，而该结点的频率即为两树叶的频率之和。
3. 将所得的频率放回序列中适当位置，使频率的顺序保持。
4. 重复上面的步骤，直到序列中只剩下一个元素，霍夫曼树建立完毕

构建霍夫曼树

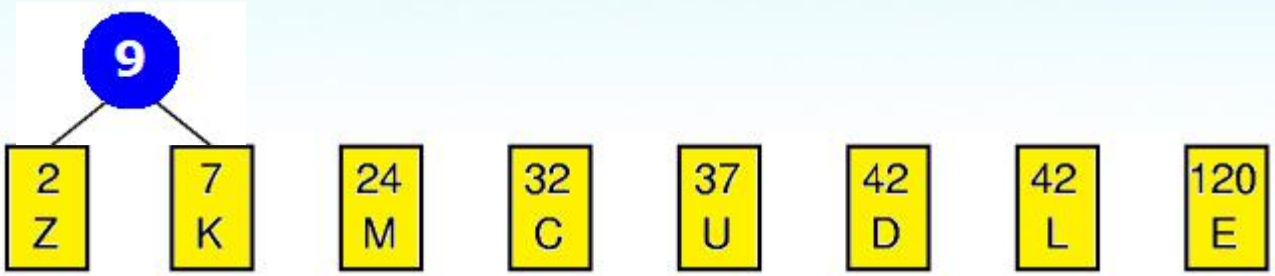
- 已知字符出现频率

E	L	D	U	C	F	K	Z
120	42	42	37	32	24	7	2

- Step 1 按照频率将字符排序

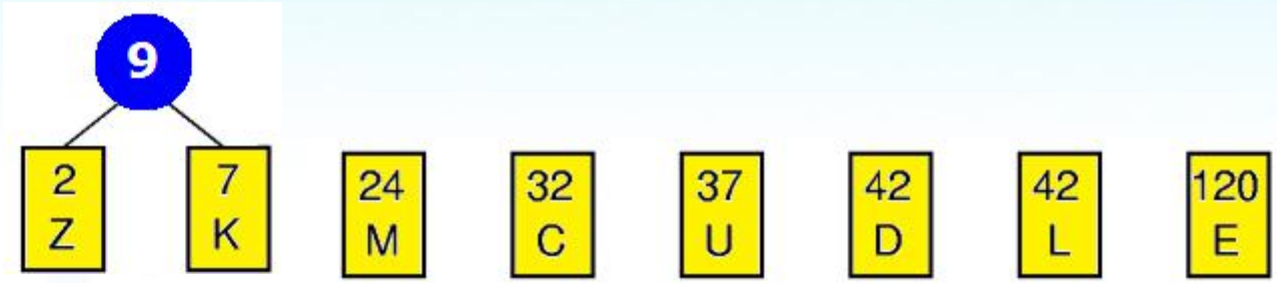
构建霍夫曼树

- Step 2 合并头两个字符，新结点的频率为两者频率之和



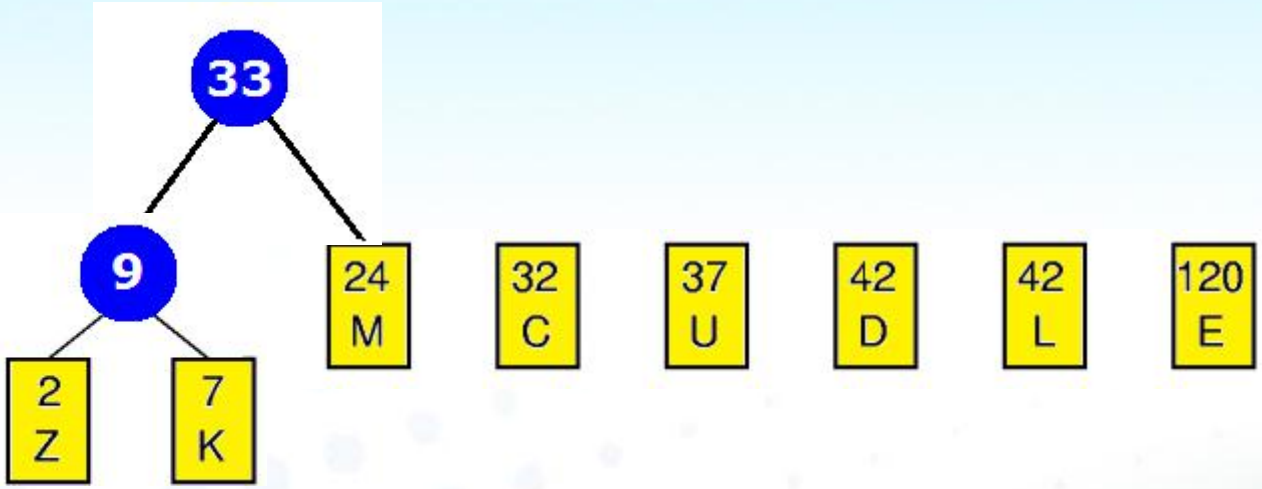
构建霍夫曼树

- Step 3 将新的频率放回序列中，使得频率顺序保持



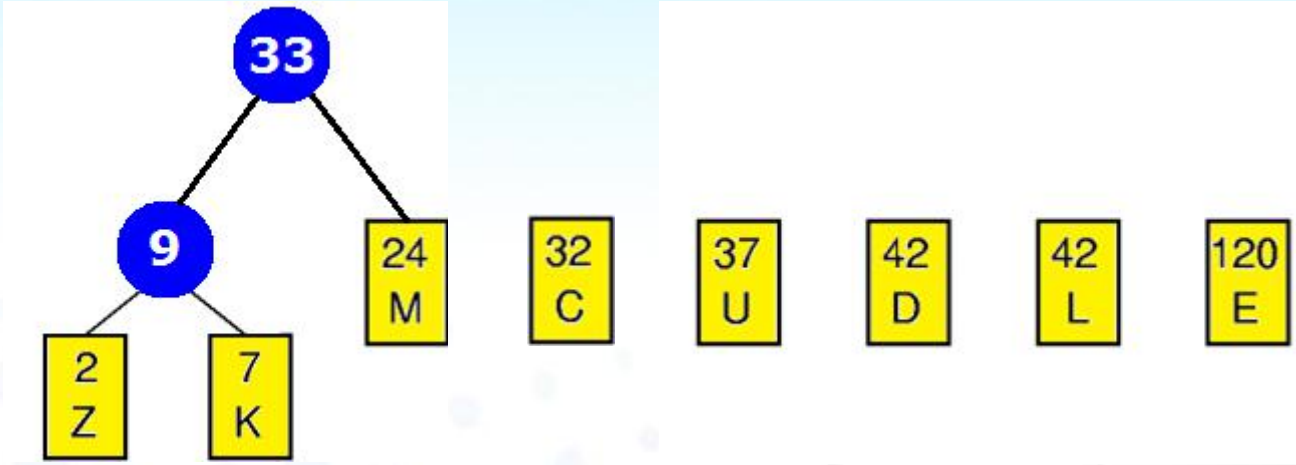
构建霍夫曼树

- Step 4 重复上面的过程



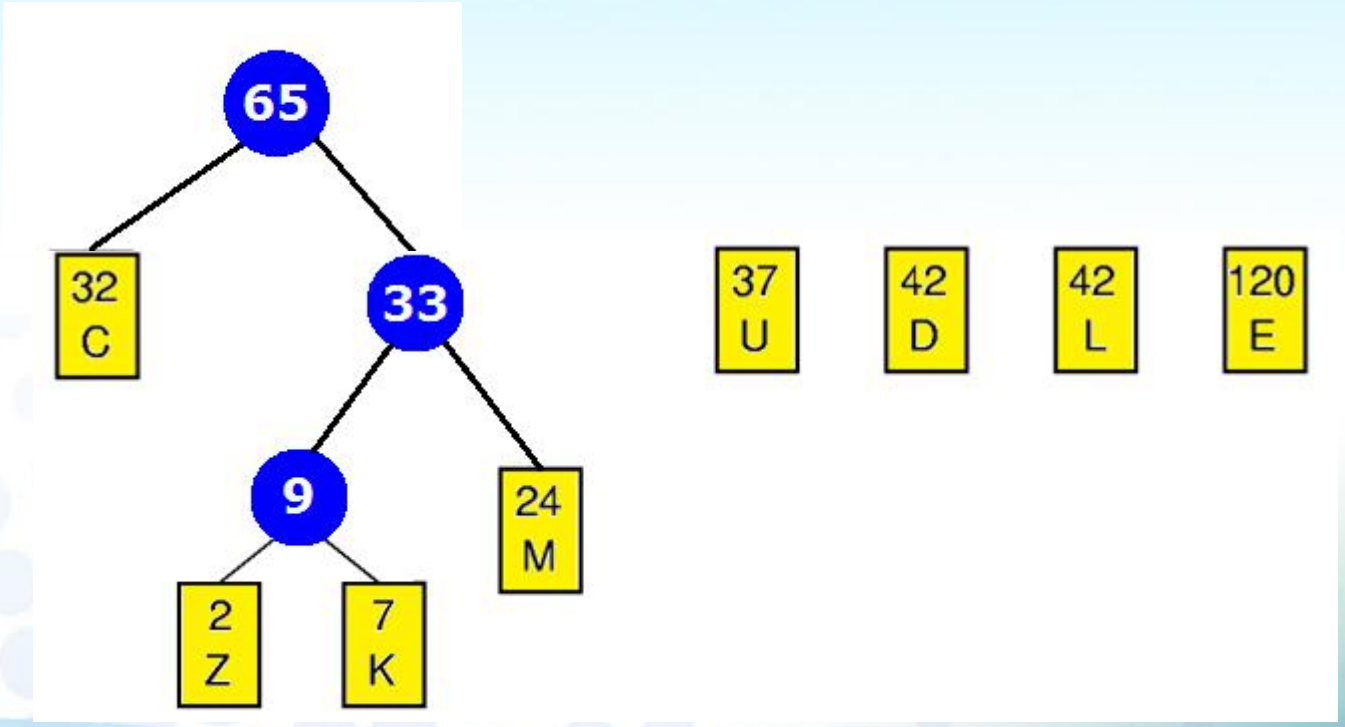
构建霍夫曼树

- Step 4 重复上面的过程



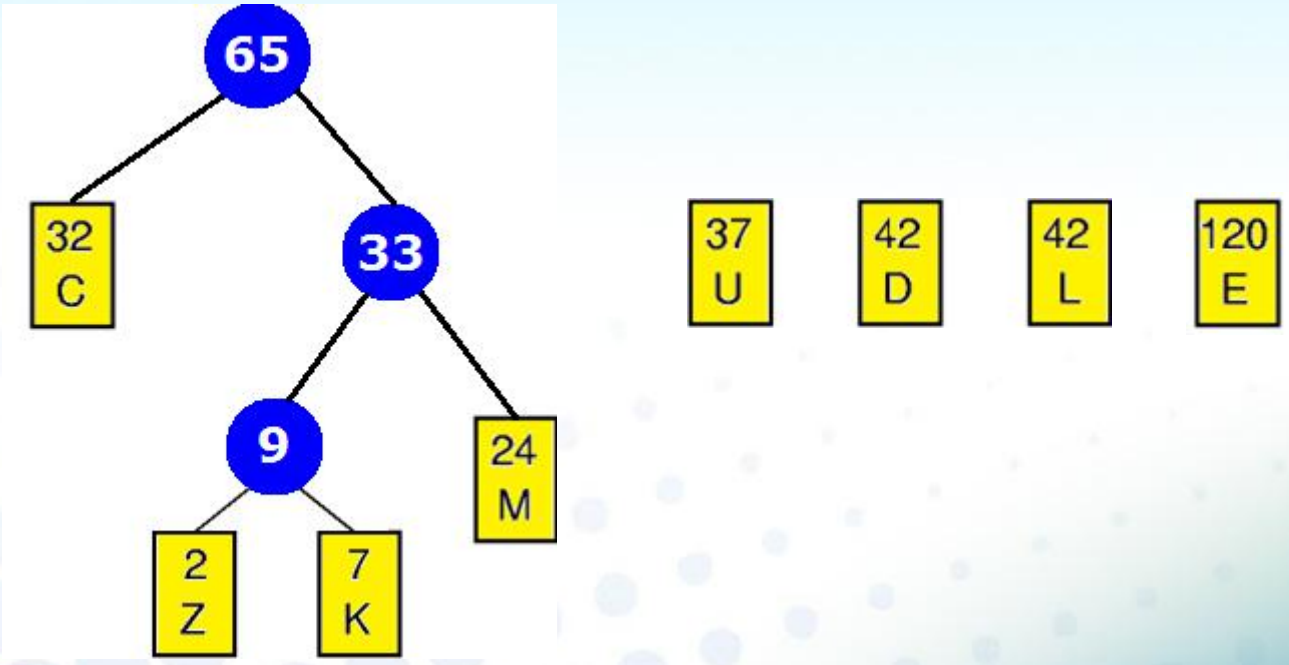
构建霍夫曼树

- Step 4 重复上面的过程



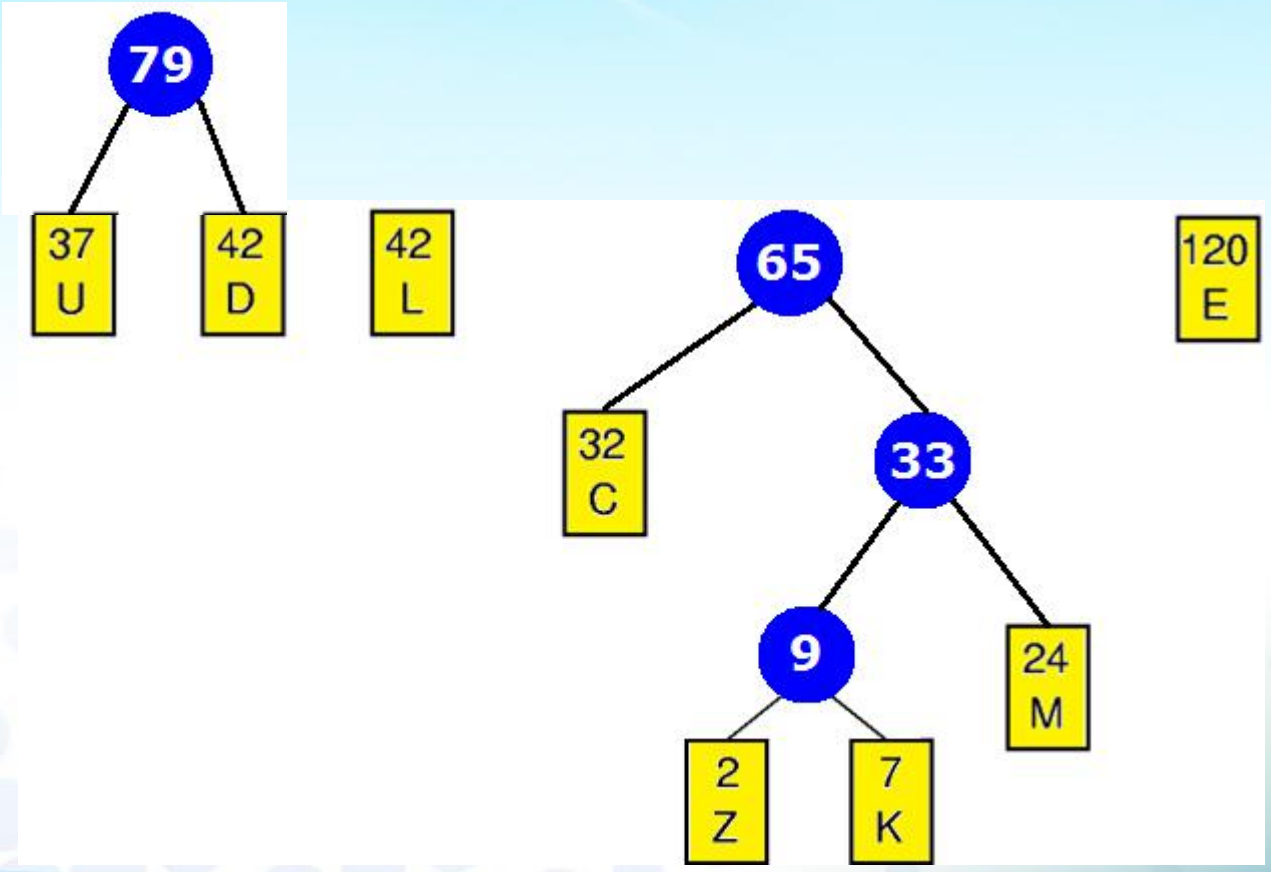
构建霍夫曼树

- Step 4 重复上面的过程



构建霍夫曼树

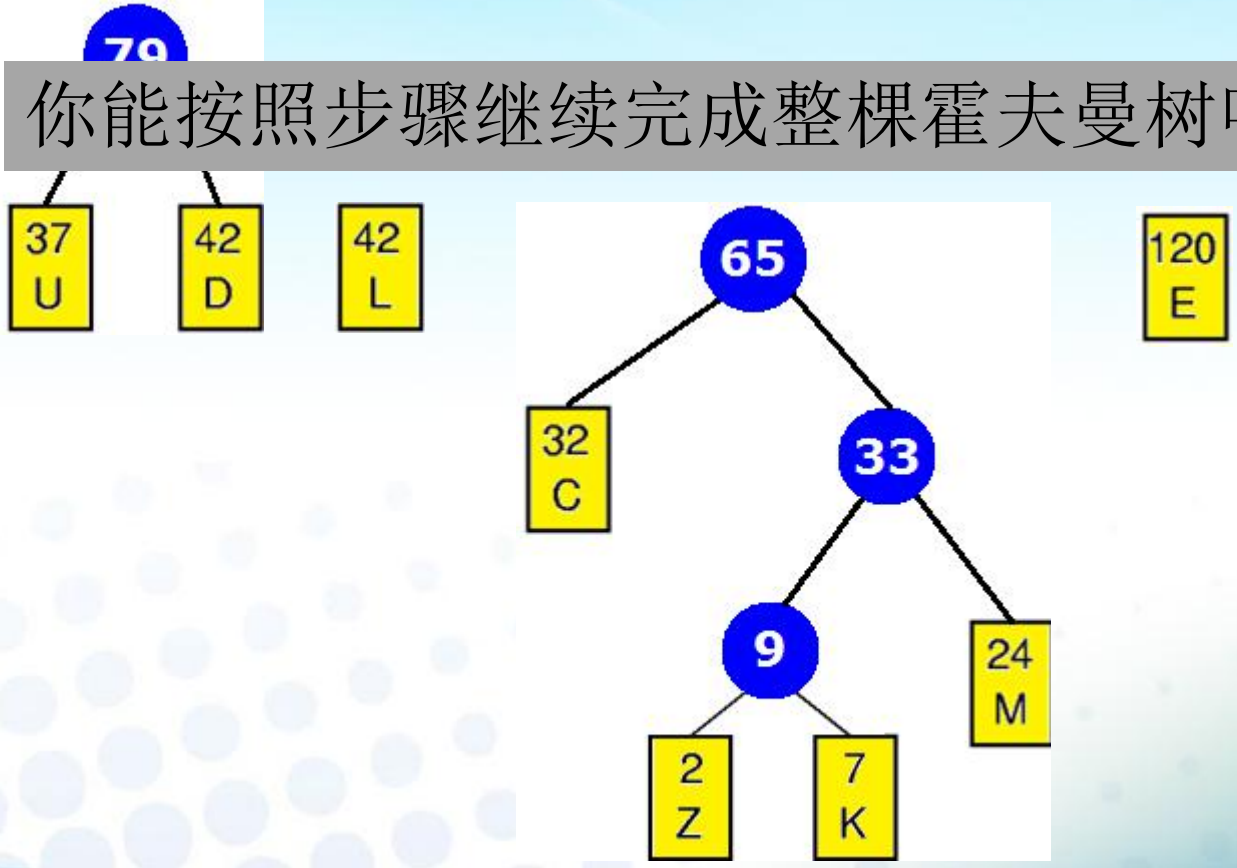
- Step 4 重复上面的过程



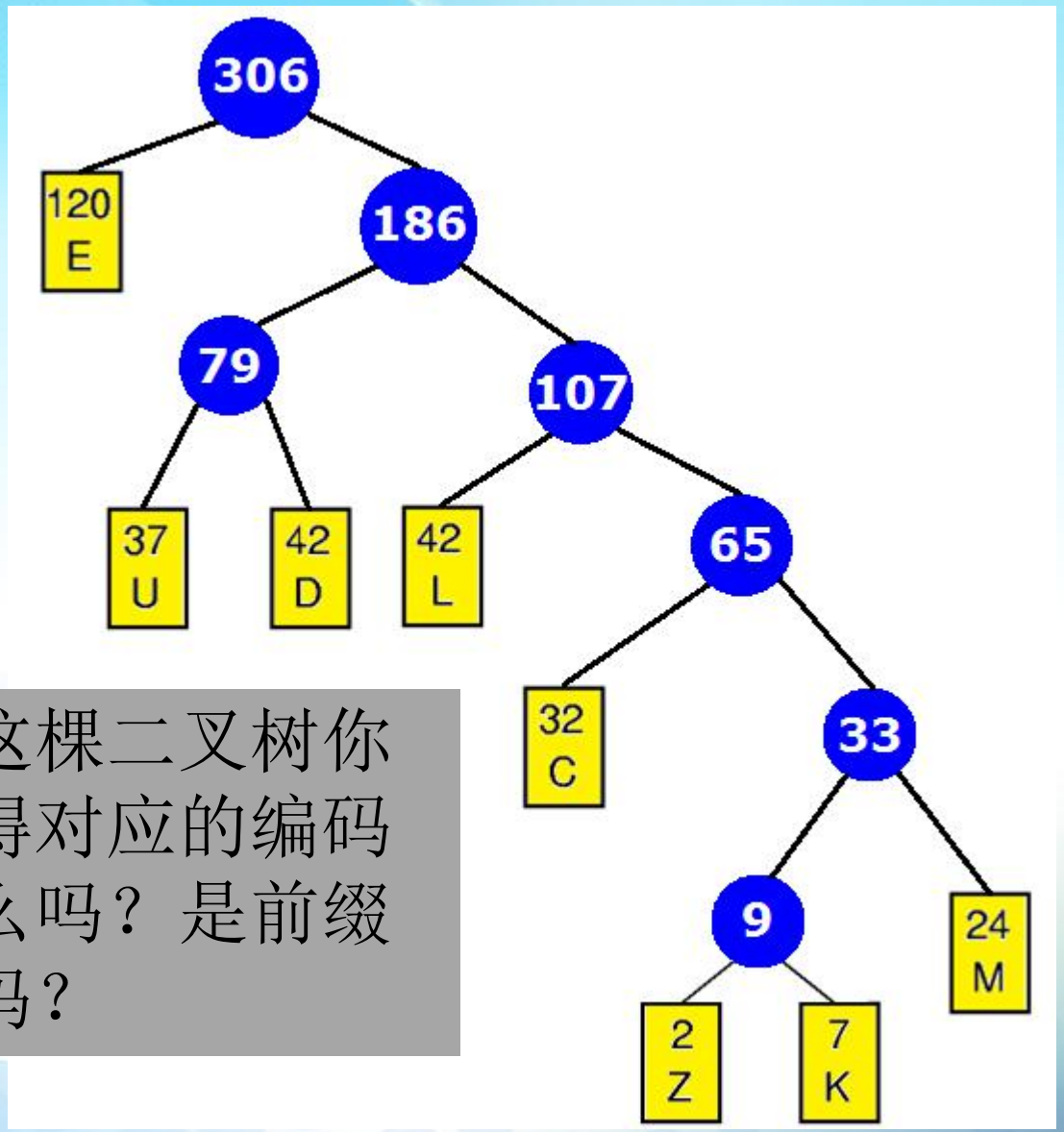
构建霍夫曼树

- Step 4 重复上面的过程

你能按照步骤继续完成整棵霍夫曼树吗？

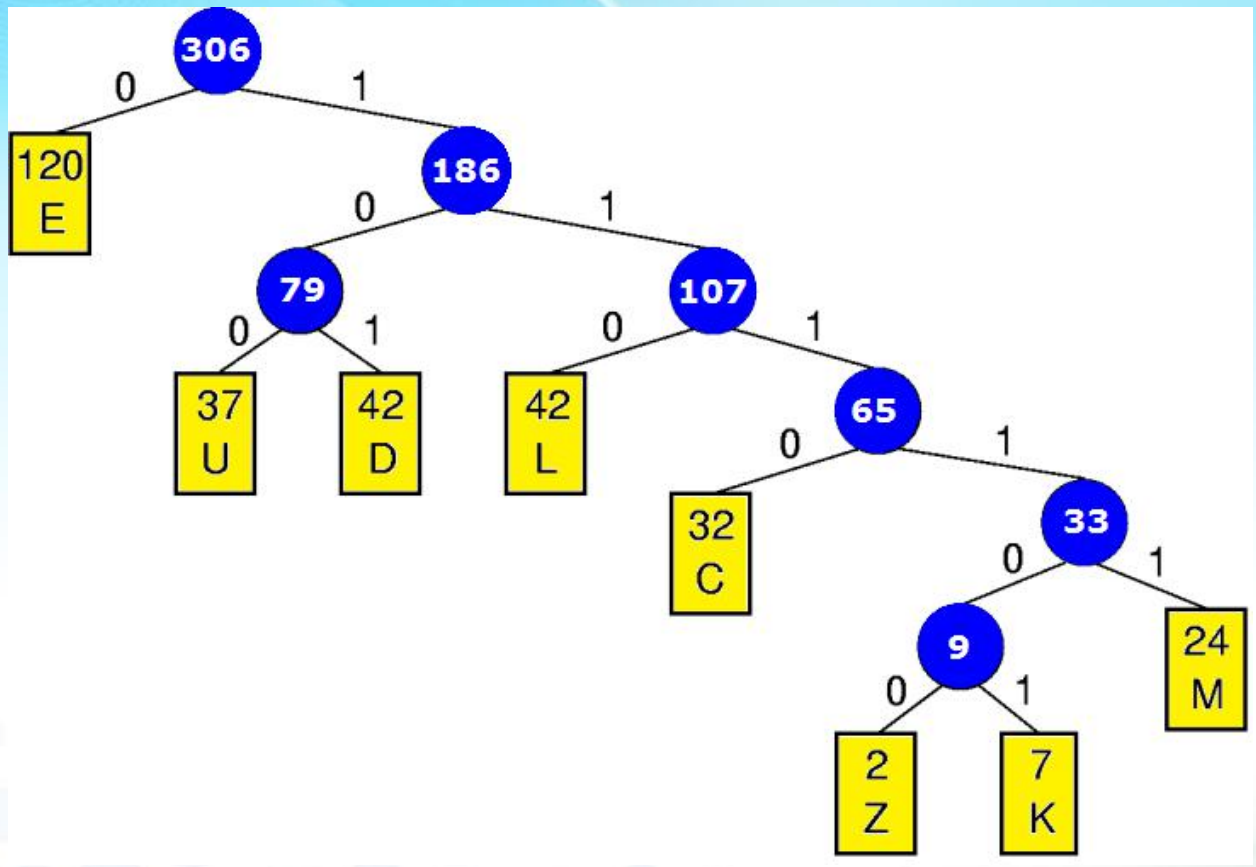


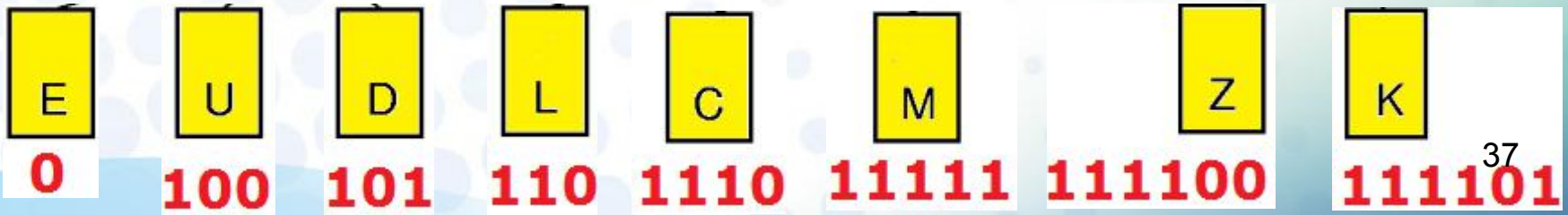
最终的霍夫曼树



根据这棵二叉树你能获得对应的编码是什么吗？是前缀编码吗？

最终的霍夫曼编码


306
0
1
120
E
186
0
1
79
107
0
1
0
1
37
U
42
D
42
L
65
0
1
32
C
33
0
1
9
24
M
0
1
2
Z
7
K


E
0
U
100
D
101
L
110
C
1110
M
11111
Z
111100
K
111101

比较霍夫曼编码

- 根据出现频率获得了霍夫曼编码

字符	E	U	D	L	C	M	Z	K
编码	0	100	101	110	1110	11111	111100	111101
频数	120	42	42	37	32	24	7	2
概率	0.392	0.137	0.137	0.121	0.105	0.078	0.023	0.007

平均码长 = 1x0.392+3x0.137+3x0.137+… = 2.567 bits

- 固定长度编码编码8个字符则至少需要3bits

平均码长 = 固定码长 = 3 bits

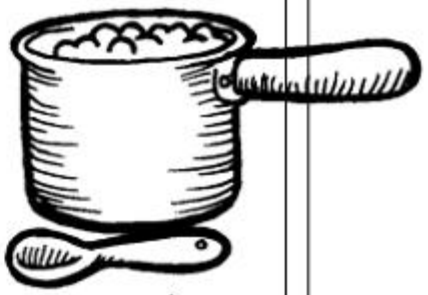
Ziv-Lempel压缩算法

- 之前我们介绍过了游程压缩算法，这里我们会介绍称为Ziv-Lempel的经典压缩算法

Ziv-Lempel压缩算法

- 小游戏——文字的解压缩
  - 下面是一首缺词少字的诗歌，你能完整地恢复它的原貌吗？
  - 依照空格旁箭头的指示，复制箭头指示的内容寻找缺失的字母或词语

Pease porridge hot,
Pease porridge cold,
Pease porridge in the pot,
Nine days old.



Some like it hot,
Some like it cold,
Some like it in the pot,
Nine days old.

文字的压缩

- 下面学习一下文本的压缩

The Rain
Pitter patter
Pitter patter
Listen to the rain
Pitter patter
Pitter patter
On the window pane

文字的压缩

- 在这首诗中，你能找出两个或更多重复的字母、单词或词组吗？

这个游戏中可以忽略字母的大小写，但现实中计算机系统进行压缩时是不能忽略大小写的。

The Rain

Pitter patter

Pitter Patter

Listen to the rain

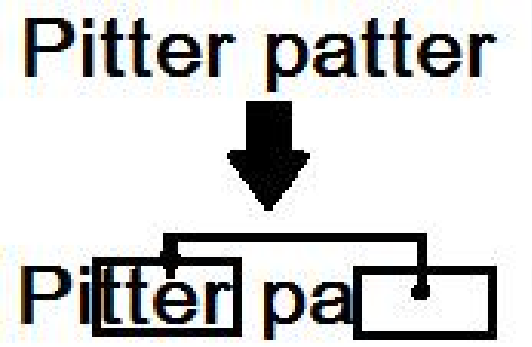
Pitter Patter

Pitter Patter

On the window pane

文字的压缩

- 你能用下面所示的格子来代替这首诗中重复的单词吗？



The Rain

Pitter patter

Pitter patter

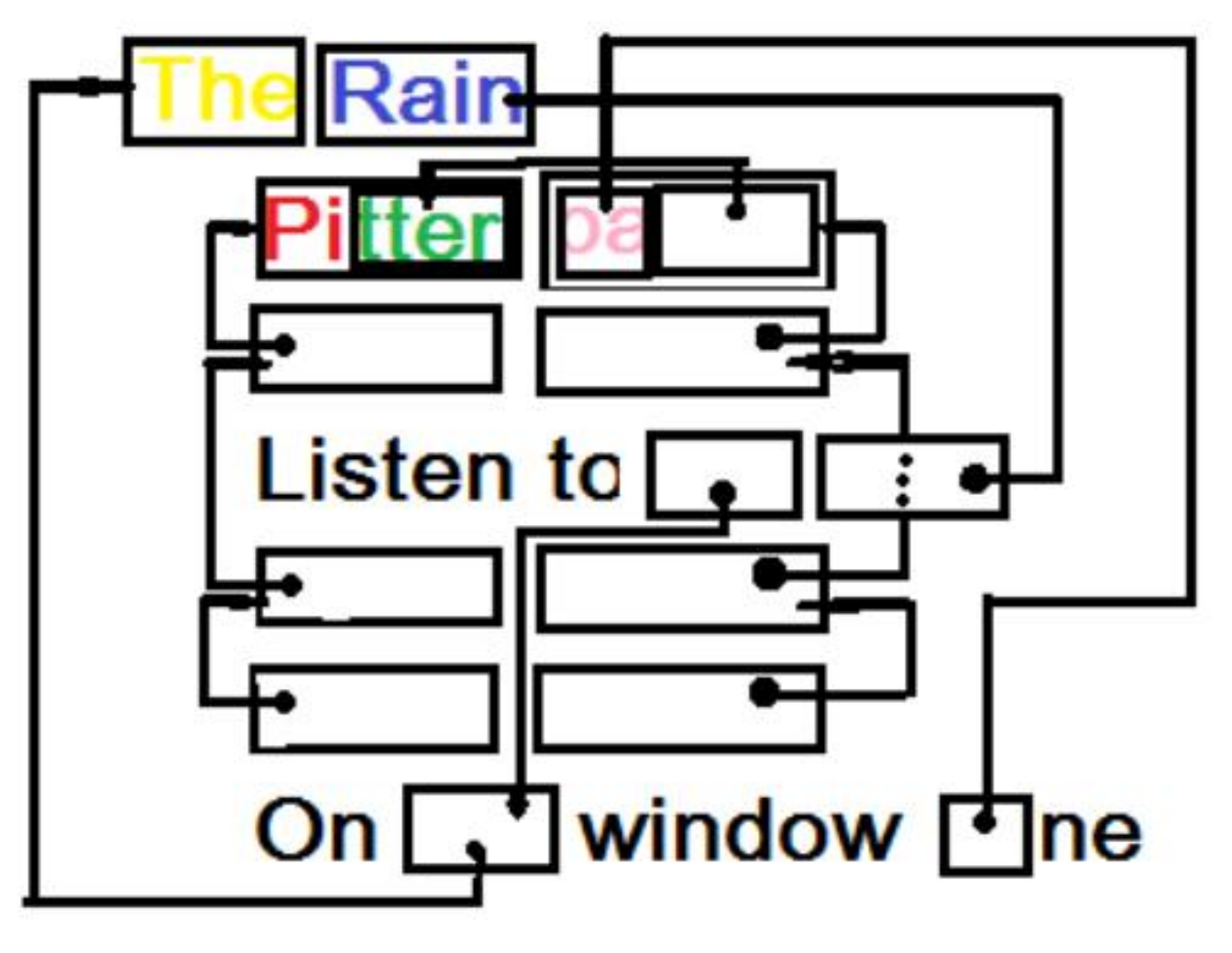
Listen to the rain

Pitter patter

Pitter patter

On the window pane

文字的压缩



文字的压缩

- 你所做的正是计算机压缩数据的过程
  - 现在文中查找出现过的重复字母组合，如果发现这种字母组合在前文也出现过
  - 这个字母组合会被移除并用指针代替，并且指针将指向该字母组合之前出现的位置
  - 而解压文档文档时，计算机处理的过程就好比是在第一个游戏中做的那样

文字的压缩

- 原诗中一共包含多少个字母以及空格？

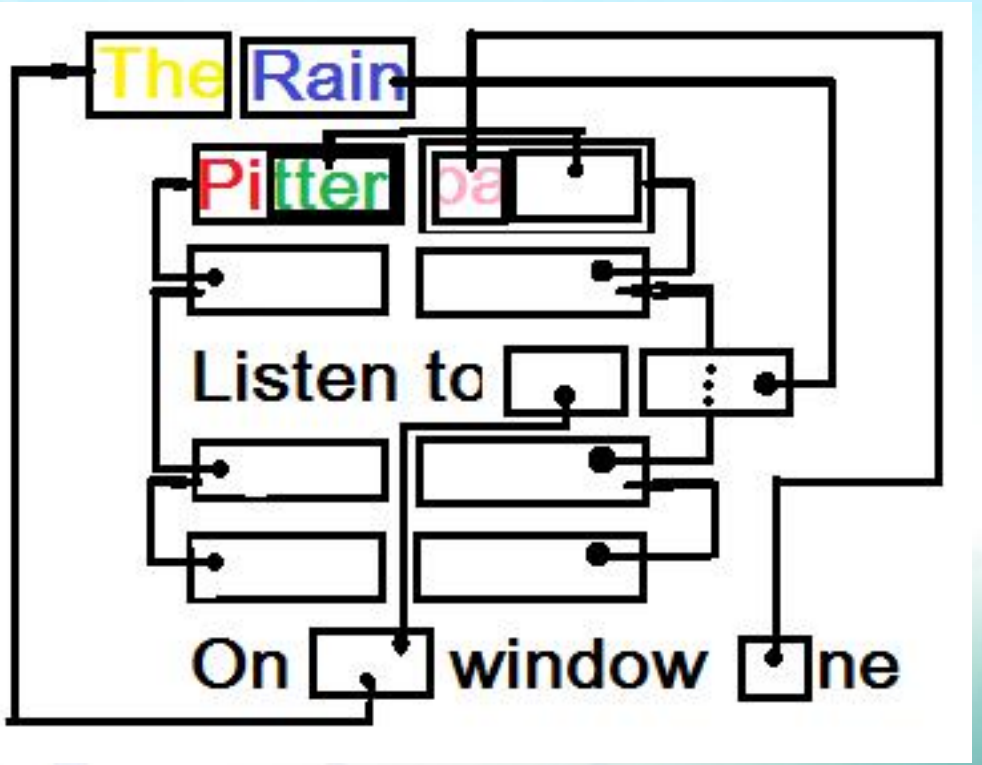
  85个字母和11个空格

- 压缩后的诗中包含多少个字母以及空格

  33个字母和22个空格

- 这首诗压缩后比压缩前缩小了多少？

  缩小了41个位置空间



文字的压缩

- 其实上面的比较有些问题，因为压缩后的诗需要额外的信息来指示被移除的字母
- 你所画出的一个箭头，在计算机中都将相应地记录
  - 这个位置返回到原始单词的距离
  - 包含字母的个数
  - 字格中的符号等

文字的压缩

- 计算机如何表示我们的箭头和方格呢？
  - 计算机使用两个数字来分别表示它们

  - 括号中数字的意义是
    - 其中7表示回退7个字符后开始复制 遇到第一个‘t’
    - 其中4代表需要复制4个字母 复制‘tter’

文字的压缩

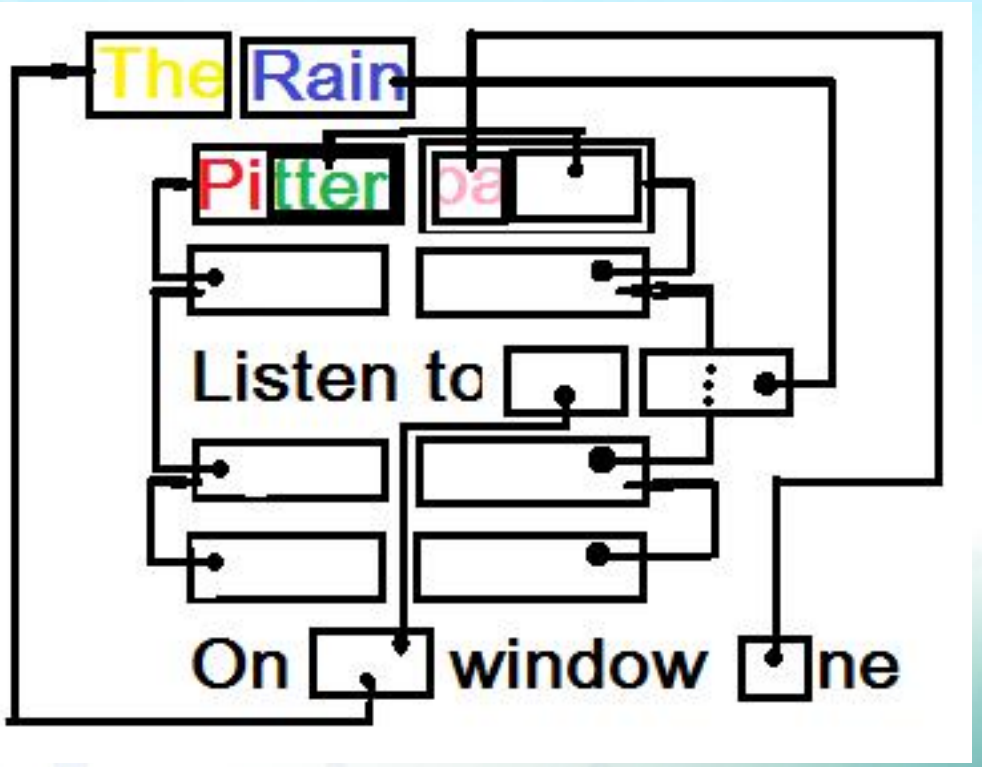
- 原诗中有多少个箭头？

11个箭头

- 在这里表示全部箭头一共需格外占用多大空间呢？

计算机中表示箭头往往需要用到两个字符的长度

22个字符空间



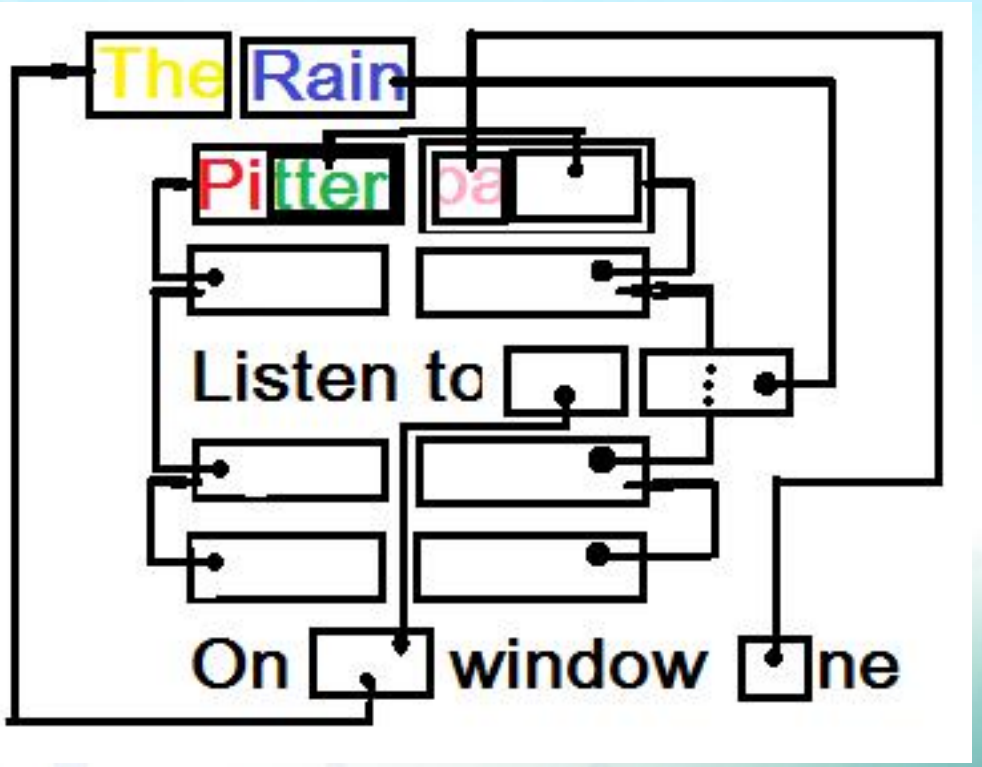
文字的压缩

- 如果把箭头占用的算进去，压缩后的诗需要多少字符空间？

66个字符空间

- 这样一来，压缩后的诗比原来少了多少字符呢？

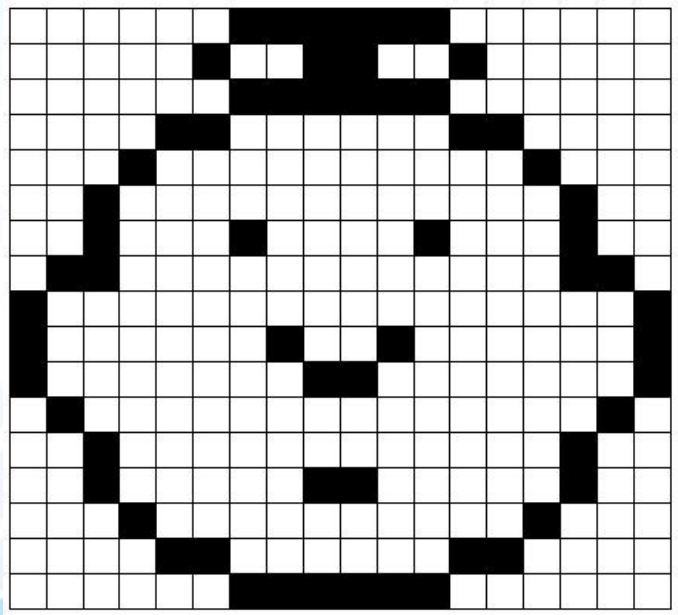
30个字符空间

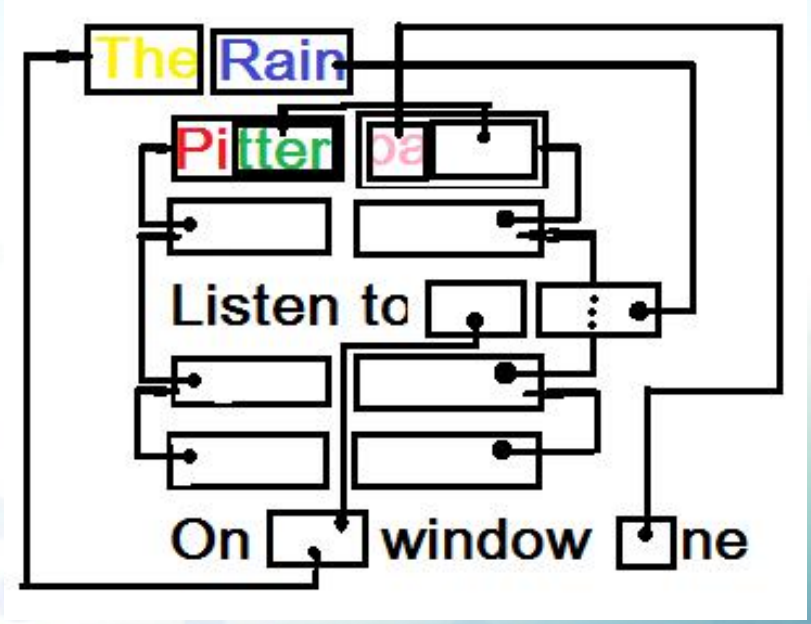


文字的压缩

如果用像素代替文字，你觉得还能

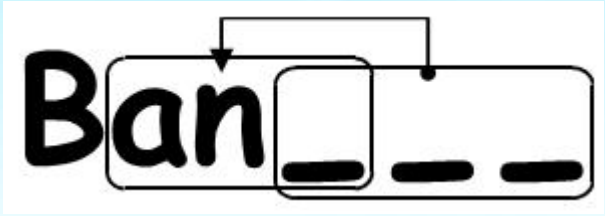
也是可以的，因为这种方式本质是寻找文档中的重复的模式，因此只要有重复的模式都可以用这种方式压缩，无论是图像还是文字





有趣的编码

- 你能解决下面这个问题吗？应该如何编码？



有趣的编码

- 这种情况我们可以写为

Ban(2,3)

- 2表示回退两个字符到第一个‘a’

Ban---

- 3表示要复制3个字符，但是却只有两个字符可用

Banana

有趣的编码

- 如果没有这种“自指向代码”如何编码banana呢？
  - ban(2,2)a ？
  - bana(2,3) ？
- 但是无论这两种哪一种都比之前的ban(2,3)都多用了一个字符
- 因此有时候自指向代码可以帮助节省空间
  - 一个极端例子

bananananananan → ban(2,10)

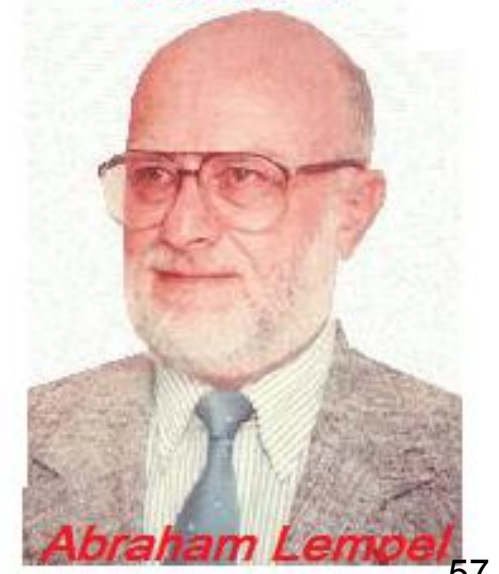
文字的压缩

- 指针的前面还需要一个标识来说明它被作为指针使用，还是仅仅被作为字符使用
- 故指针一般需要占用两个字符
- 而指针处通常对应的原字符长度平均为4个
- 所有压缩后的数据体积大约会缩小到原始体积的一半

Ziv-Lempel算法的故事

这里谈到的压缩方式称为“Ziv-Lempel”压缩，有Jacob Ziv和Abraham Lempel这两名计算机科学家共同于20世纪70年代发明。

不幸的是，有人将他们论文的署名顺序弄错，本该称为“ZL”压缩，却被错误地称为“LZ”压缩。由于这个错误流传太广以至于直到今天我们还依旧沿用这个错误的名字“LZ”压缩。



Ziv-Lempel算法的应用

- LZ算法拥有多种不同的演变算法
  - LZ77, LZ78, LZW ……
- 一些LZ算法的演变算法被应用到
  - GIF图像格式一部分
  - 大名鼎鼎的WinZip  和RAR  都是源于LZ算法

其他的压缩算法

- 游程压缩算法，LZ压缩算法，都是无损压缩算法
  - 无损压缩是指压缩后的文件，可以被完全还原，而不会影响文件的内容
- 还有更多的有损压缩算法
  - mp3
  - rmvb
  - jpeg

其他的压缩算法

- 有损压缩利用了人类对图像或声波中的某些频率成分不敏感的特性，允许压缩过程中损失一定的信息
- 虽然不能完全恢复原始数据，但是所损失的部分对理解原始图像的影响很小，却换来了大得多的压缩比
- 因此有损压缩广泛应用于语音，图像和视频数据的压缩。

压缩前，5M

压缩后，412.6k

作业

- 用LZ算法压缩以下儿歌
- twinkle, twinkle, little star.
- how i wonder what you are.
- up above the world so high.
- like a diamond in the sky.
- twinkle, twinkle, little star.
- how i wonder what you are.