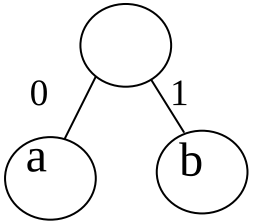
第四章：算术编码

- 算术编码：越来越流行（在很多标准中被采用）
- 适合的场合：
  - 小字母表：如二进制信源
  - 概率分布不均衡
  - 建模与编码分开

- 内容：
  - 算术编码的基本思想
  - 一些性质
  - 实现
    - 有限精度：区间缩放（浮点数/整数实现）
    - 计算复杂度：用移位代替乘法→二进制编码
  - 自适应模型
  - QM编码器：自适应二进制编码

回顾： Huffman编码

- 例1：信源的符号数目很少
  - $\begin{Bmatrix} X: & a & b \\ P(X) & 0.1 & 0.9 \end{Bmatrix}$



a=0, b=1

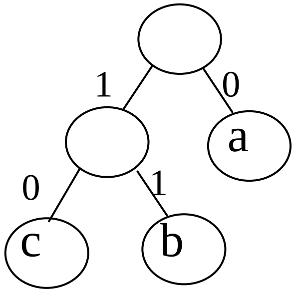
回顾：扩展的Huffman编码

- 例2：信源的符号的概率严重不对称：
  - A = {a, b, c}, P(a) = 0.95, P(b) = 0.02, P(c) = 0.03
  - H = 0.335 bits/symbol
- Huffman编码：

  a 0

  b 11

  c 10



  - l = 1.05 bits/symbol
  - 冗余（Redundancy） = l - H = 0.715 bits/sym (213%!)
- 问题：能做得更好吗？

回顾：扩展的Huffman编码

- 基本思想：
  - 考虑对两个字母序列而不是单个字母编码

Letter	Probability	Code
aa	0.9025	0
ab	0.0190	111
ac	0.0285	100
ba	0.0190	1101
bb	0.0004	110011
bc	0.0006	110001
ca	0.0285	101
cb	0.0006	110010
cc	0.0009	110000

$l = 1.222/2 = 0.611$

冗余 = 0.276 bits/symbol
（27%）

回顾：扩展的Huffman编码

- 该思想还可以继续扩展
  - 考虑长度为n的所有可能的m^n 序列 (已做了3^2)
- 理论上：考虑更长的序列能提高编码性能
- 实际上：
  - 字母表的指数增长将使得这不现实
    - 例如：对长度为3的ASCII序列：$256^3 = 2^{24} = 16M$
    - 需要对长度为n的所有序列产生码本
    - 很多序列的概率可能为0
  - 分布严重不对称是真正的大问题：
    - A = {a, b, c}, P(a) = 0.95, P(b) = 0.02, P(c) = 0.03
    - $H = 0.335$ bits/symbol
    - $l_1 = 1.05$, $l_2 = 0.611$, …
    - 当$n = 8$时编码性能才变得可接受
    - 但此时$|alphabet| = 3^8 = 6561$!!!

算术编码（Arithmetic Coding）

- 算术编码：从另一种角度对很长的信源符号序列进行有效编码
  - 对整个序列信源符号串产生一个唯一的标识（ tag ）
    - 直接对序列进行编码（不是码字的串联）：非分组码
  - 不用对该长度所有可能的序列编码
  - 标识是[0,1)之间的一个数（二进制小数，可作为序列的二进制编码）

概率复习

- 随机变量：
  - 将试验的输出映射到实数
- 用数字代替符号
  - $X(a_i) = i$, 其中 $a_i \in \mathrm{A}$ ($\mathrm{A} = \{a_i\}$, $i = 1..n$)
- 给定信源的概率模型P
  - 概率密度函数（probability density function, *pdf*）

$$\boxed{P\left(X=i\right)=P\left(a_i\right)}$$

  - 累积密度函数（cumulative density function, *cdf*）

$$\boxed{F_X\left(i\right)=\sum_{k=1}^{i}P\left(X=k\right)=\sum_{k=1}^{i}P\left(a_i\right)}$$

产生标识

- 将[0, 1)分为m个区间：

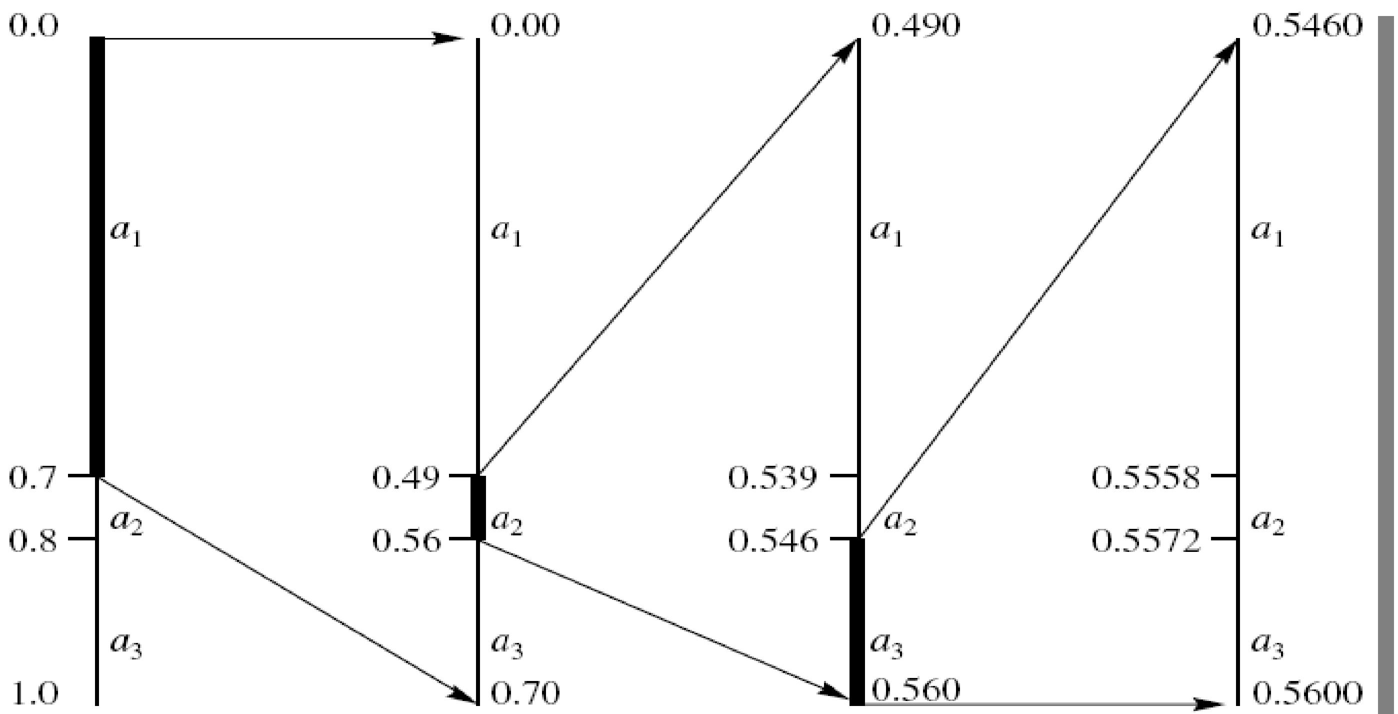
$$\left[F_X(i-1), F_X(i)\right), i=1..m, \quad F_X(0)=0$$

- 定义一一映射：
  - $a_k \Leftrightarrow [F_X(k\text{-}1), F_X(k)], k = 1..m, F_X(0) = 0$
  - $[F_X(k\text{-}1), F_X(k)]$区间内的任何数字表示 a_k
- 对2字母序列$a_k\, a_j$编码
  - 对a_k， 选择$[F_X(k\text{-}1), F_X(k)]$
  - 然后将该区间按比例分割并选取第j个区间：

$$\left[F_X(k-1)+\frac{F_X(j-1)}{F_X(k)-F_X(k-1)}, F_X(k-1)+\frac{F_X(j)}{F_X(k)-F_X(k-1)}\right]$$

产生标识：例

- 考虑对$a_1a_2a_3$编码：
  - $A = \{a_1, a_2, a_3\}$, $P = \{0.7, 0.1, 0.2)$
  - 映射：$a_1 \Leftrightarrow 1$, $a_2 \Leftrightarrow 2$, $a_3 \Leftrightarrow 3$
  - *cdf*: $F_X(1) = 0.7$, $F_X(2) = 0.8$, $F_X(3) = 1.0$



映射成实数

- A = {a_1, a_2, …, a_m}

$$\overline{T}_X(a_i) = \sum_{k=1}^{i-1} P(X=k) + \frac{1}{2}P(X=i) = F_X(i-1) + \frac{1}{2}P(X=i)$$

❖ 对公平掷骰子的例子：{1, 2, 3, 4, 5, 6}

$$P(X=k) = \frac{1}{6} \quad for\ k = 1..6$$

$$\overline{T}_X(2) = P(X=1) + \frac{1}{2}P(X=2) = 0.25$$

$$\overline{T}_X(5) = \sum_{k=1}^{4} P(X=k) + \frac{1}{2}P(X=5) = 0.75$$

词典顺序（ Lexicographic order ）

Outcome	Tag
1	$0.08\overline{33}$
3	$0.41\overline{66}$
4	$0.58\overline{33}$
6	$0.91\overline{66}$

- 字符串的词典顺序：

$$\overline{T}_X^{(n)}(\mathbf{x}_i) = \sum_{\forall \mathbf{y}:\mathbf{y} \prec \mathbf{x}_i} P(\mathbf{y}) + \frac{1}{2} P(\mathbf{x}_i)$$

  - 其中 $y \prec x$ 表示“在字母顺序中，y在x的前面”
  - n 为序列的长度

词典顺序：例

- 考虑两轮连续的骰子：
  - 输出 = {11, 12, …, 16, 21, 22, …, 26, …, 61, 62, …, 66}

$$\bar{T}_X(13) = P(x=11) + P(x=12) + \frac{1}{2}P(x=13) = \frac{5}{72} = 0.69\bar{4}$$

❖ 注意：

- 为了产生13的标识，我们不必对产生其他标识
- 但是，为了产生长度为n的字符串的标识，我们必须知道比其短的字符串的概率
  - 这与产生所有的码字一样繁重！
- 应该想办法来避免此事

区间构造

- 观察
  - 包含某个标识的区间与所有其他标识的区间不相交
- 基本思想
  - 递归：将序列的下/上界视为更短序列的界的函数
- 上述骰子的例子：
  - 考虑序列：3 2 2
  - 令$u^{(n)}$, $l^{(n)}$ 为长度为n序列的上界和下界，则

$$u^{(1)} = F_X(3),\ l^{(1)} = F_X(2)$$

$$u^{(2)} = F_X^{(2)}(32),\ l^{(2)} = F_X^{(2)}(31)$$

$$F_X^{(2)}(32) = P(\mathbf{x}=11) + \ldots + P(\mathbf{x}=16) + P(\mathbf{x}=21) + \ldots + P(\mathbf{x}=26) + P(\mathbf{x}=31) + P(\mathbf{x}=32)$$

区间构造

$$F_X^{(2)}\left(32\right)=\left[P(\mathbf{x}=11)+\ldots+P(\mathbf{x}=16)\right]+\left[P(\mathbf{x}=21)+\ldots+P(\mathbf{x}=26)\right]+P(\mathbf{x}=31)+P(\mathbf{x}=32)$$

$$\sum_{i=1}^{6}P\left(\mathbf{x}=ki\right)=\sum_{i=1}^{6}P\left(x_1=k,x_2=i\right)=P\left(x_1=k\right)\sum_{i=1}^{6}P\left(x_2=i\right)=P\left(x_1=k\right),\quad where\,\mathbf{x}=x_1x_2$$

$$F_X^{(2)}\left(32\right)=P(x_1=1)+P(x_1=2)+P(\mathbf{x}=31)+P(\mathbf{x}=32)=F_X(2)+P(\mathbf{x}=31)+P(\mathbf{x}=32)$$

$$P(\mathbf{x}=31)+P(\mathbf{x}=32)=P(x_1=3)\left(P(x_2=1)+P(x_2=2)\right)=P(x_1=3)F_X(2)$$

$$P(x_1=3)=F_X(3)-F_X(2)$$

$$F_X^{(2)}\left(32\right)=F_X(2)+\left(F_X(3)-F_X(2)\right)F_X(2)$$

$$u^{(2)}=l^{(1)}+\left(u^{(1)}-l^{(1)}\right)F_X(2)$$

区间构造

$$F_X^{(2)}(32) = F_X(2) + \left(F_X(3) - F_X(2)\right)F_X(2) \qquad F_X^{(2)}(31) = F_X(2) + \left(F_X(3) - F_X(2)\right)F_X(1)$$

$$u^{(2)} = l^{(1)} + \left(u^{(1)} - l^{(1)}\right)F_X(2) \qquad l^{(2)} = l^{(1)} + \left(u^{(1)} - l^{(1)}\right)F_X(1)$$

$$u^{(3)} = F_X^{(3)}(322), \quad l^{(3)} = F_X^{(3)}(321)$$

$$F_X^{(3)}(322) = F_X^{(2)}(31) + \left(F_X^{(2)}(32) - F_X^{(2)}(31)\right)F_X(2)$$

$$F_X^{(3)}(321) = F_X^{(2)}(31) + \left(F_X^{(2)}(32) - F_X^{(2)}(31)\right)F_X(1)$$

$$u^{(3)} = l^{(2)} + \left(u^{(2)} - l^{(2)}\right)F_X(2)$$

$$l^{(3)} = l^{(2)} + \left(u^{(2)} - l^{(2)}\right)F_X(1)$$

产生标识

- 通常，对任意序列$x = x_1 x_2 \ldots x_n$

$$\boxed{\begin{aligned} u^{(k)} &= l^{(k-1)} + \left(u^{(k-1)} - l^{(k-1)}\right) F_X\left(x_k\right) \\ l^{(k)} &= l^{(k-1)} + \left(u^{(k-1)} - l^{(k-1)}\right) F_X\left(x_k - 1\right) \end{aligned}}$$

$$\boxed{\bar{T}_X(x) = \frac{u^{(n)} + l^{(n)}}{2}}$$

只需知道信源的*cdf*，即信源的概率模型

算术编码：编码和解码过程都只涉及算术运算（加、减、乘、除）

产生标识：例

$$u^{(k)} = l^{(k-1)} + \left(u^{(k-1)} - l^{(k-1)}\right) F_X(x_k)$$
$$l^{(k)} = l^{(k-1)} + \left(u^{(k-1)} - l^{(k-1)}\right) F_X(x_k - 1)$$

- 考虑随机变量$X(a_i) = i$
  - 对序列1 3 2 1编码：

$$F_X(k) = 0, k \le 0, \quad F_X(1) = 0.8, \quad F_X(2) = 0.82, \quad F_X(3) = 1, \quad F_X(k) = 1, k > 3$$

$$l^{(0)} = 0, \quad u^{(0)} = 1$$

1

$$l^{(1)} = l^{(0)} + \left(u^{(0)} - l^{(0)}\right) F_X(0) = 0$$
$$u^{(1)} = l^{(0)} + \left(u^{(0)} - l^{(0)}\right) F_X(1) = 0.8$$

13

$$l^{(2)} = l^{(1)} + \left(u^{(1)} - l^{(1)}\right) F_X(2) = 0.656$$
$$u^{(2)} = l^{(1)} + \left(u^{(1)} - l^{(1)}\right) F_X(3) = 0.8$$

132

$$l^{(3)} = l^{(2)} + \left(u^{(2)} - l^{(2)}\right) F_X(1) = 0.7712$$
$$u^{(3)} = l^{(2)} + \left(u^{(2)} - l^{(2)}\right) F_X(2) = 0.77408$$

1321

$$l^{(4)} = l^{(1)} + \left(u^{(3)} - l^{(3)}\right) F_X(0) = 0.7712$$
$$u^{(4)} = l^{(3)} + \left(u^{(3)} - l^{(3)}\right) F_X(1) = 0.773504$$

$$\boxed{\bar{T}_X(1321) = \frac{u^{(4)} + l^{(4)}}{2} = 0.772352}$$

解码标识

- Algorithm
  - Initialize $l^{(0)} = 0$, $u^{(0)} = 1$.

1. For each i, $i = 1..n$
   - Compute $t^{*}=(\text{tag}-l^{(k-1)})/(\text{u}^{(k-1)}-l^{(k-1)})$.
2. Find the x_k: $F_X(x_k-1) \le t^* \le F_X(x_k)$.
3. Update $u^{(n)}$, $l^{(n)}$
4. If done--exit, otherwise goto 1.

解码：例

- **Algorithm**
  - Initialize $l^{(0)} = 0$, $u^{(0)} = 1$.
  1. Compute $t^*=(\text{tag}-l^{(k-1)})/(u^{(k-1)}-l^{(k-1)})$.
  2. Find the x_k: $F_X(x_k-1) \le t^* \le F_X(x_k)$.
  3. Update $u^{(k)}$, $l^{(k)}$
  4. If done--exit, otherwise goto 1.

$$\bar{T}_X(1321) = 0.772352$$

$$F_X(k) = 0, k \le 0, \quad F_X(1) = 0.8,$$
$$F_X(2) = 0.82, \quad F_X(3) = 1, \quad F_X(k) = 1, k > 3$$

$$u^{(k)} = l^{(k-1)} + \left(u^{(k-1)} - l^{(k-1)}\right) F_X(x_k)$$
$$l^{(k)} = l^{(k-1)} + \left(u^{(k-1)} - l^{(k-1)}\right) F_X(x_k - 1)$$

$$t^* = (0.772352 - 0)/(1-0) = 0.772352$$
$$F_X(0) = 0 \le t^* \le 0.8 = F_X(1)$$
$$l^{(1)} = l^{(0)} + \left(u^{(0)} - l^{(0)}\right) F_X(0) = 0$$
$$u^{(1)} = l^{(0)} + \left(u^{(0)} - l^{(0)}\right) F_X(1) = 0.8$$

⇨ **1**

$$t^* = (0.772352 - 0)/(0.8-0) = 0.96544$$
$$F_X(2) = 0.82 \le t^* \le 1 = F_X(3)$$
$$l^{(2)} = l^{(1)} + \left(u^{(1)} - l^{(1)}\right) F_X(2) = 0.656$$
$$u^{(2)} = l^{(1)} + \left(u^{(1)} - l^{(1)}\right) F_X(3) = 0.8$$

⇨ **13**

$$t^* = \frac{0.772352 - 0.656}{0.8 - 0.656} = 0.808$$
$$F_X(1) = 0.8 \le t^* \le 0.82 = F_X(2)$$
$$l^{(3)} = l^{(2)} + \left(u^{(2)} - l^{(2)}\right) F_X(1) = 0.7712$$
$$u^{(3)} = l^{(2)} + \left(u^{(2)} - l^{(2)}\right) F_X(2) = 0.77408$$

⇨ **132**

$$t^* = \frac{0.772352 - 0.7712}{0.77408 - 0.7712} = 0.4$$
$$F_X(0) = 0 \le t^* \le 0.8 = F_X(1)$$

⇨ **1321**

算术编码的唯一性和效率

- 上述产生的标识可以唯一表示一个序列，这意味着该标识的二进制表示为序列的唯一二进制编码
- 但二进制表示的精度可以是无限长：保证唯一性但不够有效
- 为了保证有效性，可以截断二进制表示，但如何保证唯一性？

- 答案：为了保证唯一性和有效性，需取小数点后l位数字作为信源序列的码字，其中

$$l(\mathbf{x}) = \left\lceil \log \frac{1}{P(\mathbf{x})} \right\rceil + 1$$

  - 注意：P(x)为最后区间的宽度，也是该符号串的概率
  - 符合概率匹配原则：出现概率较大的符号取较短的码字，而对出现概率较小的符号取较长的码字

算术编码的唯一性和效率

- 长度为n的序列的算术编码的平均码长为：

$$
\begin{aligned}
l_A &= \sum P(\mathbf{x}) l(\mathbf{x}) \\
&= \sum P(\mathbf{x}) \left[\left\lceil \log \frac{1}{P(\mathbf{x})} \right\rceil + 1 \right] \\
&< \sum P(\mathbf{x}) \left[\log \frac{1}{P(\mathbf{x})} + 1 + 1 \right] \\
&= -\sum P(\mathbf{x}) \log P(\mathbf{x}) + 2 \sum P(\mathbf{x}) \\
&= H\left(X^n\right) + 2 = nH\left(X\right) + 2
\end{aligned}
$$

$$
\boxed{nH\left(X\right) < l_{A^n} < nH\left(X\right) + 2 \quad \Rightarrow \quad H\left(X\right) < l_A < H\left(X\right) + \frac{2}{n}}
$$

算术编码的效率高：当信源符号序列很长，平均码长接近信源的熵

实现

- 迄今为止
  - 已有能工作的编码/解码算法
  - 假设实数（无限精度）
    - 最终$l^{(n)}$和$u^{(n)}$将集中到一起
  - 我们需要对字符串增量式编码
- 观测：当区间变窄时
  1. $[l^{(n)}, u^{(n)}] \subset [0, 0.5)$, 或
  2. $[l^{(n)}, u^{(n)}] \subset [0.5, 1)$, 或
  3. $l^{(n)} \in [0, 0.5), u^{(n)} \in [0.5, 1)$.
  - 先集中处理1. & 2，稍后再讨论3

实现

- 编码器：
  - 一旦我们到达1. 或2.，就可以忽略[0,1)的另一半
  - 还需要告知解码器标识所在的半区间：
    - 发送0/1 比特用来指示下上界所在区间
  - 将标识区间缩放到[0, 1):
    - E_1: [0, 0.5) => [0, 1); $E_1(x) = 2x$
    - E_2: [0. 5,1) => [0, 1); $E_2(x) = 2(x-0.5)$
  - 注意：在缩放过程中我们丢失了最高位
  - 但这不成问题—我们已经发送出去了
- 解码器
  - 根据0/1比特并相应缩放
    - 与编码器保持同步

标识产生：带缩放的例子

- 考虑随机变量$X(a_i) = i$
  - 对序列1 3 2 1编码：

$$F_X(k)=0, k\le 0,\quad F_X(1)=0.8,\quad F_X(2)=0.82,\quad F_X(3)=1,\quad F_X(k)=1, k>3$$

$$l^{(0)}=0,\quad u^{(0)}=1$$

```
Input: 1321
```

$$l^{(1)}=l^{(0)}+\left(u^{(0)}-l^{(0)}\right)F_X(0)=0$$

$$u^{(1)}=l^{(0)}+\left(u^{(0)}-l^{(0)}\right)F_X(1)=0.8$$

$$[l^{(1)},u^{(1)})\not\subset[0,0.5)$$

$$[l^{(1)},u^{(1)})\not\subset[0.5,1)$$

$$\Rightarrow get\ next\ symbol$$

```
Output:
```

```
Input: -321
```

$$l^{(2)}=l^{(1)}+\left(u^{(1)}-l^{(1)}\right)F_X(2)=0.656$$

$$u^{(2)}=l^{(1)}+\left(u^{(1)}-l^{(1)}\right)F_X(3)=0.8$$

$$[0.656,0.8]\subset[0.5,1)$$

$$l^{(2)}=2\times(0.656-0.5)=0.312$$

$$u^{(2)}=2\times(0.8-0.5)=0.6$$

```
Output: 1
```

标识产生：带缩放的例子

$$l^{(2)} = 0.312, \quad u^{(2)} = 0.6$$

Input: --**2**1

$$l^{(3)} = l^{(2)} + \left(u^{(2)} - l^{(2)}\right) F_X(1) = 0.5424$$

$$u^{(3)} = l^{(2)} + \left(u^{(2)} - l^{(2)}\right) F_X(2) = 0.54816$$

Output: 1**1**

Input: ---1

$$l^{(3)} = 2 \times (0.5424 - 0.5) = 0.0848$$

$$u^{(3)} = 2 \times (0.54816 - 0.5) = 0.09632$$

Output: 11**0**

Input: ---1

$$l^{(3)} = 2 \times 0.0848 = 0.1696$$

$$u^{(3)} = 2 \times 0.09632 = 0.19264$$

Output: 110**0**

Input: ---1

$$l^{(3)} = 2 \times 0.1696 = 0.3392$$

$$u^{(3)} = 2 \times 0.19264 = 0.38528$$

Output: 1100**0**

标识产生：带缩放的例子

Input: ---1

$$l^{(3)} = 2 \times 0.3392 = 0.6784$$

$$u^{(3)} = 2 \times 0.38528 = 0.77056$$

Output: 11000**1**

Input: ---1

$$l^{(3)} = 2 \times (0.6784 - 0.5) = 0.3568$$

$$u^{(3)} = 2 \times (0.77056 - 0.5) = 0.54112$$

Output: 110001

Input: ---**1**

$$l^{(4)} = 0.3568 + (0.54112 - 0.3568) F_X(0) = 0.3568$$

$$u^{(4)} = 0.3568 + (0.54112 - 0.3568) F_X(1) = 0.504256$$

Output: 110001

- EOT:
  - $[l^{(n)}, u^{(n)}]$区间内的任何一个数字
  - 在此选 $0.5_{10} = 0.1_2$

Output: 1100011**1**0…

❖ 注意：$0.1100011 = 2^{-1}+2^{-2}+2^{-6}+2^{-7} = 0.7734375 \in [0.7712, 0.77408]$

增量式标识解码

- 我们需要增量式解码
  - 如：网络传输
- 问题
  - 如何开始？
    - 必须有足够的信息，可以对第一个字符无歧义解码
    → 接收到的比特数应比最小标识对应的区间更小
  - 怎样继续？
    - 一旦有一个非歧义的开始，只要模拟编码器即可
  - 如何结束？

标识解码：例

- 假设码字长为6
- 输入：110001100000
  - $0.110001_2 = 0.765625_{10}$
  - 第1位：1

$$\boxed{\begin{aligned} l^{(1)} &= 0 + (1-0)\times 0 = 0 \\ u^{(1)} &= 0 + (1-0)\times 0.8 = 0.8 \end{aligned}}$$

Output: 1

Input: **110001**100000

$$t^* = (0.765625 - 0)/(0.8 - 0) = 0.9570$$

$$F_X(2) = 0.82 \le t^* \le 1 = F_X(3)$$

Output: 13

$$l^{(2)} = l^{(1)} + \left(u^{(1)} - l^{(1)}\right)F_X(2) = 0.656$$

$$u^{(2)} = l^{(1)} + \left(u^{(1)} - l^{(1)}\right)F_X(3) = 0.8$$

Input:-**100011**00000 (左移)

$$l^{(2)} = 2\times(0.656 - 0.5) = 0.312$$

$$u^{(2)} = 2\times(0.8 - 0.5) = 0.6$$

Input: -**100011**00000 (**0.546875**)

$$t^* = (0.546875 - 0.312)/(0.6 - 0.312) = 0.8155$$

$$F_X(1) = 0.8 \le t^* \le 0.82 = F_X(2)$$

Output: 132

$$l^{(3)} = l^{(2)} + \left(u^{(2)} - l^{(2)}\right)F_X(1) = 0.5424$$

$$u^{(3)} = l^{(2)} + \left(u^{(2)} - l^{(2)}\right)F_X(2) = 0.54816$$

Input: --**000110**0000 (左移)

$$l^{(3)} = 2\times(0.5424 - 0.5) = 0.0848$$

$$u^{(3)} = 2\times(0.54816 - 0.5) = 0.09632$$

Input: ---**001100**000 (左移)

标识解码：例

$l^{(3)} = 2 \times 0.0848 = 0.1696$

$u^{(3)} = 2 \times 0.09632 = 0.19264$

Input: ----**01100**00 (左移)

$l^{(3)} = 2 \times 0.1696 = 0.3392$

$u^{(3)} = 2 \times 0.19264 = 0.38528$

Input: -----**11000**0 (左移)

$l^{(3)} = 2 \times 0.3392 = 0.6784$

$u^{(3)} = 2 \times 0.38528 = 0.77056$

Input: -----**100000** (左移)

Input: -----**100000**

$l^{(4)} = 0.3568 + (0.54112 - 0.3568) F_X(0) = 0.3568$

$u^{(4)} = 0.3568 + (0.54112 - 0.3568) F_X(1) = 0.504256$

此时解码最后一个符号

$t^* = (100000)_2 = 0.5,$

$F_X(0) = 0 \le t^* \le 0.8 = F_X(1)$

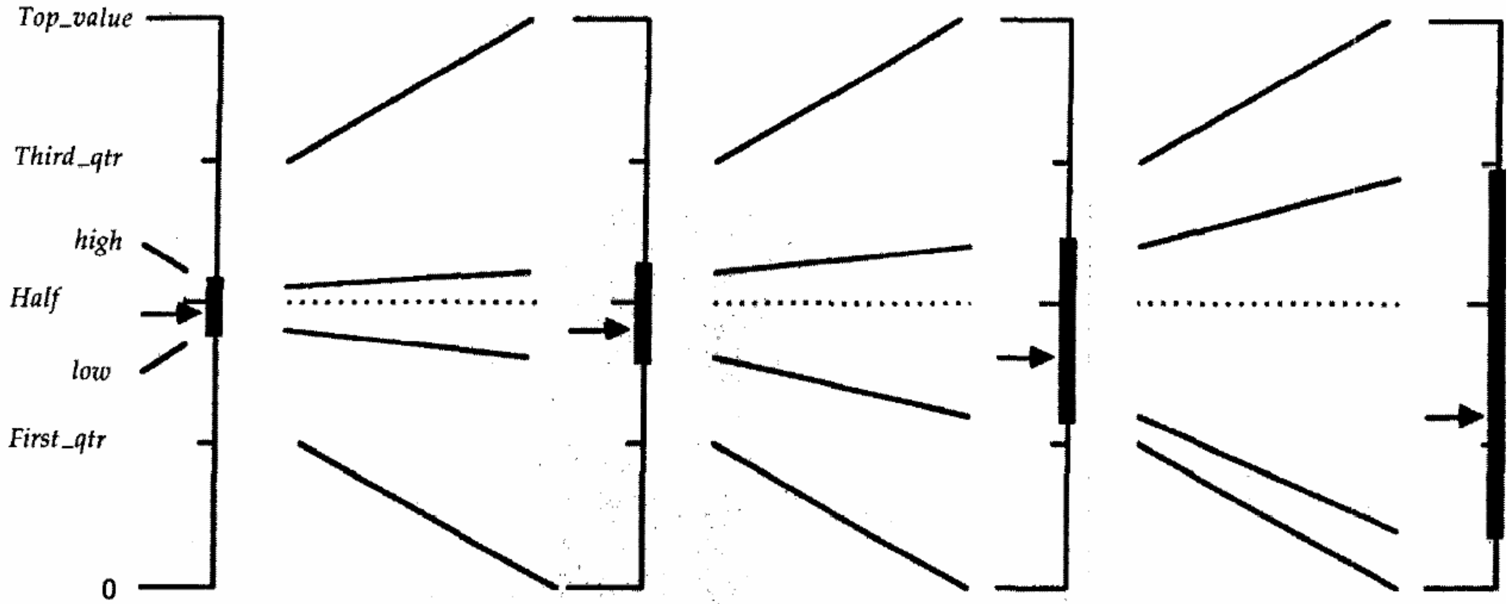
Output: 132**1**

第三种情况：E_3

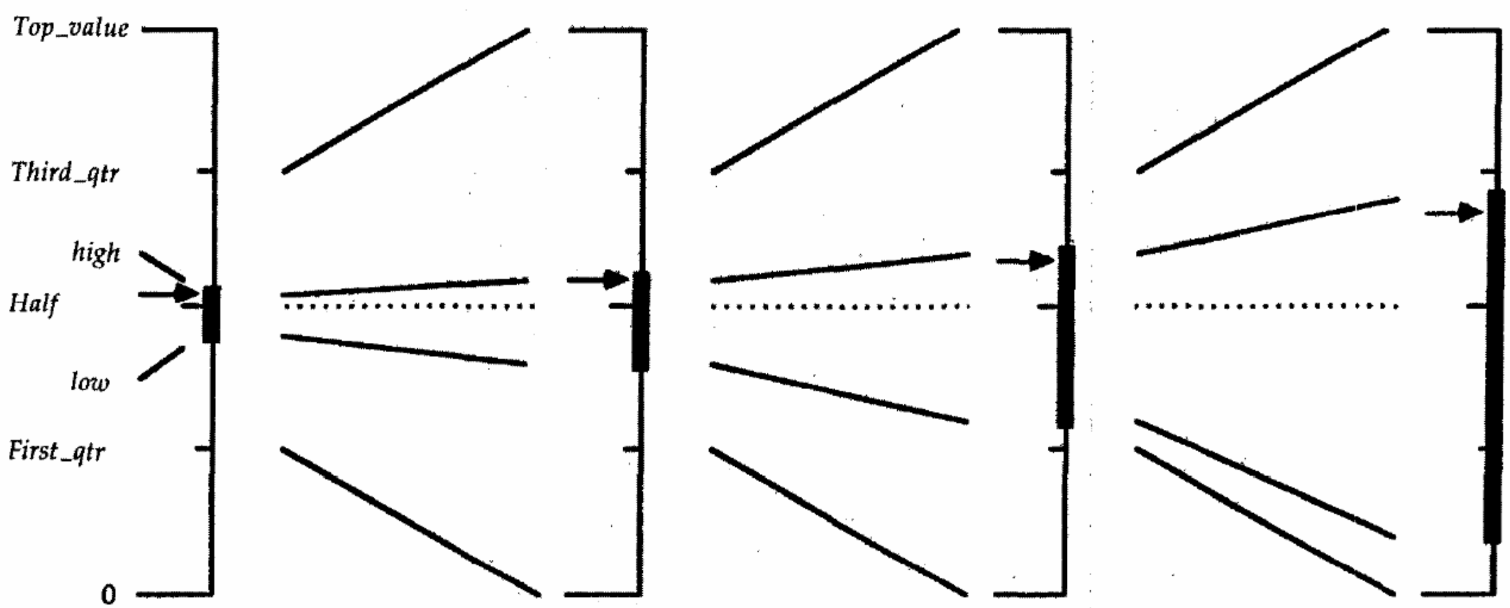
- 回忆三种情况
  1. $[l^{(n)}, u^{(n)}] \subset [0, 0.5)$: E_1: $[0, 0.5) \Rightarrow [0, 1)$; $E_1(x) = 2x$
  2. $[l^{(n)}, u^{(n)}] \subset [0.5, 1)$: E_2: $[0.5, 1) \Rightarrow [0, 1)$; $E_2(x) = 2(x-0.5)$
  3. $l^{(n)} \in [0, 0.5), u^{(n)} \in [0.5, 1)$: $E_3(x) = ???$
- E_3 缩放
  - E_3: $[0.25, 0.75) \Rightarrow [0, 1)$; $E_3(x) = 2(x-0.25)$
- 编码
  - $E_1 = 0, E_2 = 1, E_3 = ???$
  - 注意：
    - $E_3 \ldots E_3 E_1 = E_1 E_2 \ldots E_2$
    - $E_3 \ldots E_3 E_2 = E_2 E_1 \ldots E_1$
  - 规则：记录E_3 连续的次数，并在输出下一个E_2/E_1 之后发送该次数

证明请见文献： Witten, Radford, Neal, Cleary, “Arithmetic Coding for Data Compression” Communications of the ACM, vol. 30, no. 6, pp. 520-540, June 1987.

(a)


Top_value
Third_qtr
high
Half
low
First_qtr
0

(b)


Top_value
Third_qtr
high
Half
low
First_qtr
0

整数实现

$$u^{(k)} = l^{(k-1)} + \left(u^{(k-1)} - l^{(k-1)}\right) F_X(x_k)$$
$$l^{(k)} = l^{(k-1)} + \left(u^{(k-1)} - l^{(k-1)}\right) F_X(x_k - 1)$$

- 假设码字长度为n，则

$$[0,1) \rightarrow \overbrace{00\ldots0}^{n\,\text{times}} \ldots \overbrace{11\ldots1}^{n\,\text{times}} \qquad 0.5 = 1\overbrace{0\ldots0}^{n-1\,\text{times}}$$

- n_i = 长度为$TotalCount$（TC） 的序列中字符i出现的次数
- 累积计数：CC

$$CC(k) = \sum_{i=1}^{k} n_i$$

$$F_X(k) = CC(k)/TC$$

$$l^{(n)} = l^{(n-1)} + \left\lfloor \left(u^{(n-1)} - l^{(n-1)} + 1\right) \times CC(x_n - 1)/TC \right\rfloor$$
$$u^{(n)} = l^{(n-1)} + \left\lfloor \left(u^{(n-1)} - l^{(n-1)} + 1\right) \times CC(x_n)/TC \right\rfloor - 1$$

整数实现

- MSB(x) = Most Significant Bit of x
- LSB(x) = Least Significant Bit of x
- SB(x, i) = i^{th} Significant Bit of x
  - MSB(x) = SB(x, 1); LSB(x) = SB(x, m)
- E3(l, u) = (SB(l, 2) == 1 && SB(u, 2) == 0)

整数编码器

```
l=00…0, u=11…1, e3_count=0
repeat
  x=get_symbol
  l=l+ ⌊(u-l+1)×CC(x-1)/TC⌋        // lower bound update
  u=l+ ⌊(u-l+1)×CC(x)/TC⌋-1        // upper bound update
  while(MSB(u)==MSB(l) OR E3(u,l)) // MSB(u)=MSB(l)=0 → E1 rescaling
    if(MSB(u)==MSB(l))             // MSB(u)=MSB(l)=1 → E2 rescaling
      send(MSB(u))
      l = (l<<1)+0                 // shift left, set LSB to 0
      u = (u<<1)+1                 // shift left, set LSB to 1
      while(e3_count>0)
        send(!MSB(u))              // encode accumulated E3 rescalings
        e3_count--
      endwhile
    endif
    if(E3(u,l))                    // perform E3 rescaling & remember
      l = (l<<1)+0
      u = (u<<1)+1
      complement MSB(u) and MSB(l)
      e3_count++
    endif
  endwhile
until done
```

整数编码器：例

- 序列：1 3 2 1
- Count {1, 2, 3} = {40, 1, 9}
- Total count TC = 50
- Cumulative count
  - CC {0, 1, 2, 3} = {0, 40, 41, 50}
- 记住：区间的终点不会重叠
  - n = ?
  - 最小的[l(n),u(n)] =整个范围内的1/4，仍能区分0..TC 的最小区间：

=> $2^8 \times 1/4 > 50/1$

=> 最小 $n = 8$ ($2^8 = 256$)

```
l=00…0, u=11…1, e3_count=0
repeat
  x=get_symbol
  l=l+⌊(u-l+1)×CC(x-1)/TC⌋
  u=l+⌊(u-l+1)×CC(x)/TC⌋-1
  while(MSB(u)==MSB(l) OR E3(u,l))
    if(MSB(u)==MSB(l))
      send(MSB(u))
      l = (l<<1)+0
      u = (u<<1)+1
      while(e3_count>0)
        send(!MSB(u))
        e3_count--
      endwhile
    endif
    if(E3(u,l))
      l = (l<<1)+0
      u = (u<<1)+1
      complement MSB(u) and MSB(l)
      e3_count++
    endif
  endwhile
until done
```

整数编码器：例

$$l^{(0)} = 0 = (00000000)_2$$

$$u^{(0)} = 255 = (11111111)_2$$

Input: **<u>1</u>**321

$$l^{(1)} = 0 + \lfloor 256 \times 0/50 \rfloor = 0 = (00000000)_2$$

$$u^{(1)} = 0 + \lfloor 256 \times 40/50 \rfloor - 1 = 203 = (11001011)_2$$

$MSB(l) \neq MSB(u),\ E_3 = $ **false**

Output:

Input: -**<u>3</u>**21

$$l^{(2)} = 0 + \lfloor 204 \times 41/50 \rfloor = 167 = (10100111)_2$$

$$u^{(2)} = 0 + \lfloor 204 \times 50/50 \rfloor - 1 = 203 = (11001011)_2$$

$$MSB(l) = MSB(u) = 1$$

Output: **<u>1</u>**

```
l=00…0, u=11…1, e3_count=0
repeat
  x=get_symbol
  l=l+ ⌊(u-l+1)×CC(x-1)/TC⌋
  u=l+ ⌊(u-l+1)×CC(x)/TC⌋-1
  while(MSB(u)==MSB(l) OR E3(u,l))
    if(MSB(u)==MSB(l))
      send(MSB(u))
      l = (l<<1)+0
      u = (u<<1)+1
      while(e3_count>0)
        send(!MSB(u))
        e3_count--
      endwhile
    endif
    if(E3(u,l))
      l = (l<<1)+0
      u = (u<<1)+1
      complement MSB(u) and MSB(l)
      e3_count++
    endif
  endwhile
until done
```

整数编码器：例

$l^{(2)} = (10101011)_2 << 1 + 0 = (01001110)_2 = 78$

$u^{(2)} = (11001011)_2 << 1 + 1 = (10010111)_2 = 151$

$E_3 = \textbf{true}$

$l^{(2)} = \left((01001110)_2 << 1 + 0\right) \textbf{xor} \left(10000000\right) = 28$

$u^{(2)} = \left((10010111)_2 << 1 + 1\right) \textbf{xor} \left(10000000\right)_2 = 175$

e3_count = **<u>1</u>**

Input: --**<u>2</u>**1

$l^{(3)} = 28 + \lfloor 148 \times 40/50 \rfloor = 146 = (10010010)_2$

$u^{(3)} = 28 + \lfloor 148 \times 41/50 \rfloor - 1 = 148 = (10010100)_2$

$MSB(l) = MSB(u) = 1, \quad \textbf{e3\_count} = 1$

Output: 1**<u>10</u>**

```
l=00…0, u=11…1, e3_count=0
repeat
  x=get_symbol
  l=l+ ⌊(u-l+1)×CC(x-1)/TC⌋
  u=l+ ⌊(u-l+1)×CC(x)/TC⌋-1
  while(MSB(u)==MSB(l) OR E3(u,l))
    if(MSB(u)==MSB(l))
      send(MSB(u))
      l = (l<<1)+0
      u = (u<<1)+1
      while(e3_count>0)
        send(!MSB(u))
        e3_count--
      endwhile
    endif
    if(E3(u,l))
      l = (l<<1)+0
      u = (u<<1)+1
      complement MSB(u) and MSB(l)
      e3_count++
    endif
  endwhile
until done
```

整数编码器：例

Input: ---1

$$l^{(3)} = (10010010)_2 << 1 = (00100100)_2 = 36$$

$$u^{(3)} = (10010100)_2 << 1 + 1 = (00101001)_2 = 41$$

$$MSB(l) = MSB(u) = 0$$

Output: 110**<u>0</u>**

Input: ---1

$$l^{(3)} = (00100100)_2 << 1 = (01001000)_2 = 72$$

$$u^{(3)} = (00101001)_2 << 1 + 1 = (01010011)_2 = 83$$

$$MSB(l) = MSB(u) = 0$$

Output: 1100**<u>0</u>**

Input: ---1

$$l^{(3)} = (01001000)_2 << 1 = (10010000)_2 = 144$$

$$u^{(3)} = (01010011)_2 << 1 + 1 = (10100111)_2 = 167$$

$$MSB(l) = MSB(u) = 1$$

Output: 11000**<u>1</u>**

```
l=00…0, u=11…1, e3_count=0
repeat
  x=get_symbol
  l=l+ ⌊(u-l+1)×CC(x-1)/TC⌋
  u=l+ ⌊(u-l+1)×CC(x)/TC⌋-1
  while(MSB(u)==MSB(l) OR E3(u,l))
    if(MSB(u)==MSB(l))
      send(MSB(u))
      l = (l<<1)+0
      u = (u<<1)+1
      while(e3_count>0)
        send(!MSB(u))
        e3_count--
      endwhile
    endif
    if(E3(u,l))
      l = (l<<1)+0
      u = (u<<1)+1
      complement MSB(u) and MSB(l)
      e3_count++
    endif
  endwhile
until done
```

整数编码器：例

Input: ---1

$$l^{(3)} = (10010000)_2 << 1 = (00100000)_2 = 32$$

$$u^{(3)} = (10100111)_2 << 1 + 1 = (01001111)_2 = 79$$

$$MSB(l) = MSB(u) = 0$$

Output: 110001**<u>0</u>**

Input: ---1

$$l^{(3)} = (00100000)_2 << 1 = (01000000)_2 = 64$$

$$u^{(3)} = (01001111)_2 << 1 + 1 = (10011111)_2 = 159$$

$$MSB(l) \neq MSB(u), E_3 = \mathbf{true}$$

$$l^{(3)} = \left((01000000)_2 << 1 + 0\right) \mathbf{xor} \left(10000000\right) = 0$$

$$u^{(3)} = \left((10011111)_2 << 1 + 1\right) \mathbf{xor} \left(10000000\right)_2 = 191$$

e3_count = **<u>1</u>**

```
l=00…0, u=11…1, e3_count=0
repeat
  x=get_symbol
  l=l+ ⌊(u-l+1)×CC(x-1)/TC⌋
  u=l+ ⌊(u-l+1)×CC(x)/TC⌋-1
  while(MSB(u)==MSB(l) OR E3(u,l))
    if(MSB(u)==MSB(l))
      send(MSB(u))
      l = (l<<1)+0
      u = (u<<1)+1
      while(e3_count>0)
        send(!MSB(u))
        e3_count--
      endwhile
    endif
    if(E3(u,l))
      l = (l<<1)+0
      u = (u<<1)+1
      complement MSB(u) and MSB(l)
      e3_count++
    endif
  endwhile
until done
```

整数编码器：例

Input: ---**1**

$$l^{(4)} = 0 + \lfloor 192 \times 0/50 \rfloor = 0 = (00000000)_2$$

$$u^{(4)} = 0 + \lfloor 192 \times 40/50 \rfloor - 1 = 152 = (10011000)_2$$

$$MSB(l) \neq MSB(u), E_3 = \textbf{false}$$

Output: 1100010

- 结束
  - 通常，发送$l^{(4)}$: $(00000000)_2$
  - 但是 e3_count = 1，因此
  - 在发送$l^{(4)}$的第一个‘0’后发送‘1’

最后 output: 1100010**010000000**

```
l=00…0, u=11…1, e3_count=0
repeat
  x=get_symbol
  l=l+ ⌊(u-l+1)×CC(x-1)/TC⌋
  u=l+ ⌊(u-l+1)×CC(x)/TC⌋-1
  while(MSB(u)==MSB(l) OR E3(u,l))
    if(MSB(u)==MSB(l))
      send(MSB(u))
      l = (l<<1)+0
      u = (u<<1)+1
      while(e3_count>0)
        send(!MSB(u))
        e3_count--
      endwhile
    endif
    if(E3(u,l))
      l = (l<<1)+0
      u = (u<<1)+1
      complement MSB(u) and MSB(l)
      e3_count++
    endif
  endwhile
until done
```

整数解码器

```
Initialize l, u, t                          // t = first n bits
repeat
  k=0
  while( ⌊((t-l+1)×TC-1)/(u-l+1)⌋ ≥ CC(k))
    k++
  x = decode_symbol(k)
  l=l+ ⌊(u-l+1)×CC(x-1)/TC⌋
  u=l+ ⌊(u-l+1)×CC(x)/TC⌋-1
  while(MSB(u)==MSB(l) OR E3(u,l))
    if(MSB(u)==MSB(l))                      // Perform E1/E2 rescaling of l,u,t
      l = (l<<1)+0                          // add 0 to the right of l
      u = (u<<1)+1                          // add 1 to the right of u
      t = (t<<1)+next_bit
    endif
    if(E3(u,l))                             // Perform E3 rescaling of l,u,t
      l = (l<<1)+0
      u = (u<<1)+1
      t = (t<<1)+next_bit
      complement MSB(u), MSB(l), MSB(t)
    endif
  endwhile
until done
```

整数解码器：例

Input: **<u>11000100</u>**10000000

$l = 0 = (00000000)_2$

$u = 255 = (11111111)_2$

$t = 196 = (11000100)_2$

$t^* = 0 + \lfloor (197 \times 50 - 1)/256 \rfloor = 38$

$\Rightarrow k = 1 \quad \Rightarrow x = 1$

Output: **<u>1</u>**

$l = 0 + \lfloor 256 \times 0/50 \rfloor = 0 = (00000000)_2$

$u = 0 + \lfloor 256 \times 40/50 \rfloor = 203 = (11001011)_2$

$MSB(l) \neq MSB(u),\ E_3 =$ **false**

$t^* = 0 + \lfloor (197 \times 50 - 1)/203 \rfloor = 48$

$\Rightarrow k = 3 \quad \Rightarrow x = 3$

Output: 1**<u>3</u>**

```
Initialize l, u, t
repeat
  k=0
  while( ⌊(t-l+1)×CC(x-1)/TC⌋ ≥ CC(k))
    k++
  x = decode_symbol(k)
  l=l+ ⌊(u-l+1)×CC(x-1)/TC⌋
  u=l+ ⌊(u-l+1)×CC(x)/TC⌋-1
  while(MSB(u)==MSB(l) OR E3(u,l))
    if(MSB(u)==MSB(l))
      l = (l<<1)+0
      u = (u<<1)+1
      t = (t<<1)+next_bit
    endif
    if(E3(u,l))
      l = (l<<1)+0
      u = (u<<1)+1
      t = (t<<1)+next_bit
      complement MSB(u), MSB(l), MSB(t)
    endif
  endwhile
until done
```

整数解码器：例

Input: 1**10001001**0000000

$$t = (10001001)_2$$

$$l = 0 + \lfloor 204 \times 41/50 \rfloor = 167 = (10100111)_2$$

$$u = 0 + \lfloor 204 \times 50/50 \rfloor - 1 = 203 = (11001011)_2$$

$$MSB(l) = MSB(u)$$

Input: 11**00010010**000000

$$t = (00010010)_2$$

$$l = (10100111)_2 << 1 = (01001110)_2$$

$$u = (11001011)_2 << 1 + 1 = (10010111)_2$$

$MSB(l) \neq MSB(u),\ E_3 =$ **true**

Input: 110**00100100**00000

$t = (00010010)_2$ **xor** $(10000000) = (10010010)_2 = 146$

$l = ((01001110)_2 << 1)$ **xor** $(10000000) = (00011100)_2 = 28$

$u = ((10010111)_2 << 1 + 1)$ **xor** $(10000000) = (10111001)_2 = 1$

$MSB(l) \neq MSB(u),\ E_3 =$ **false**

```
Initialize l, u, t
repeat
  k=0
  while( ⌊(t-l+1)×CC(x-1)/TC⌋ ≥ CC(k))
    k++
  x = decode_symbol(k)
  l=l+ ⌊(u-l+1)×CC(x-1)/TC⌋
  u=l+ ⌊(u-l+1)×CC(x)/TC⌋-1
  while(MSB(u)==MSB(l) OR E3(u,l))
    if(MSB(u)==MSB(l))
      l = (l<<1)+0
      u = (u<<1)+1
      t = (t<<1)+next_bit
    endif
    if(E3(u,l))
      l = (l<<1)+0
      u = (u<<1)+1
      t = (t<<1)+next_bit
      complement MSB(u), MSB(l), MSB(t)
    endif
  endwhile
until done
```

整数解码器：例

$t^* = \lfloor ((146-28+1)\times 50-1)/(175-28+1) \rfloor = 40$

$\Rightarrow k=2 \quad \Rightarrow x=2$

Output: 13**2**

$l = 28 + \lfloor 148 \times 40/50 \rfloor = 146 = (10010010)_2$

$u = 28 + \lfloor 148 \times 41/50 \rfloor - 1 = 148 = (10010100)_2$

$MSB(l) = MSB(u)$，共5次

最后，$t = l = (01000000)_2$

$u = (10011111)_2$

$t^* = 0 \Rightarrow x = 1$ Output: 132**1**

- 结束
  - 已解码接收到了预定数目的字符，或
  - 接收到EOT

```
Initialize l, u, t
repeat
  k=0
  while( ⌊(t-l+1)×CC(x-1)/TC⌋ ≥ CC(k))
    k++
  x = decode_symbol(k)
  l=l+ ⌊(u-l+1)×CC(x-1)/TC⌋
  u=l+ ⌊(u-l+1)×CC(x)/TC⌋-1
  while(MSB(u)==MSB(l) OR E3(u,l))
    if(MSB(u)==MSB(l))
      l = (l<<1)+0
      u = (u<<1)+1
      t = (t<<1)+next_bit
    endif
    if(E3(u,l))
      l = (l<<1)+0
      u = (u<<1)+1
      t = (t<<1)+next_bit
      complement MSB(u), MSB(l), MSB(t)
    endif
  endwhile
until done
```

算术编码 vs. Huffman编码

- n = 序列长度
- 平均码长：
  - 算术：$H(X) \le l_A \le H(x) + 2/n$
  - 扩展Huffman：$H(X) \le l_H \le H(x) + 1/n$
- 观察
  - 二者的积渐近性质相同
    - Huffman有一个微小的边际
  - 扩展的Huffman要求巨大数量的存储和编码m^n
  - 而算术编码不用

➔ 实际应用中可以对算术编码假设较大的长度n 但 Huffman不可

➔ 我们期望算术编码表现更好（除了当概率为2的整数次幂的情况）

算术编码 vs. Huffman编码

- 增益为字母表大小和分布的函数
  - 较大的字母表（通常）较适合 Huffman
  - 不均衡的分布更适合算术编码
- 很容易将算术编码扩展到多个编码器
  - QM编码器
- 很容易将算术编码适应到统计变化模型（自适应模型、上下文模型）
  - 不必对树重新排序
  - 可以将建模与编码分开

应用：图像压缩

自适应算术编码：
对像素值

Image	Bits/pixel	Ratio Arithmetic	Ratio Huffman
Sena	6.52	1.23	1.16
Sensin	7.12	1.12	1.27
Earth	4.67	1.71	1.67
Omaha	6.84	1.17	1.14

自适应算术编码：
对像素差分

Image	Bits/pixel	Ratio Arithmetic	Ratio Huffman
Sena	3.89	2.06	2.08
Sensin	4.56	1.75	1.73
Earth	3.92	2.04	2.04
Omaha	6.27	1.28	1.26

自适应算术编码

- 统计编码技术需要利用信源符号的概率，获得这个概率的过程称为建模。不同准确度（通常也是不同复杂度）的模型会影响算术编码的效率。
- 建模的方式：
  - 静态建模：在编码过程中信源符号的概率不变。但一般来说事先知道精确的信源概率是很难的，而且是不切实际的。
  - 自适应动态建模：信源符号的概率根据编码时符号出现的频繁程度动态地进行修改。当压缩消息时，我们不能期待一个算术编码器获得最大的效率，所能做的最有效的方法是在编码过程中估算概率。
- 算术编码很容易与自适应建模相结合。

自适应算术编码

- 自适应算术编码：
  - 在编码之前，假设每个信源符号的频率相等（如都等于1），并计算累积频率
  - 从输入流中读入一个字符，并对该符号进行算术编码
  - 更新该符号的频率，并更新累积频率

  - 由于在解码之前，解码器不知道是哪个信源符号，所以概率更新应该在解码之后进行
  - 对应的，编码器也应在编码后进行概率更新

自适应算术编码

- 为了提高解码器的搜索效率，信源符号的频率、累积频率表按符号出现频率降序排列
- 在自适应算术编码中，可以用平衡二叉树来快速实现对频率和累积频率的更新
  - 平衡二叉树可用数组实现（类似Huffman编码中的堆）
  - 最可能出现的符号安排在根附近，从而平均搜索的次数最小

  - 详见《数据压缩原理与应用》第2.15节

自适应算术编码

- 例：信源符号及其出现频率为：

数组下标：	1	2	3	4	5	6	7	8	9	10
信源符号：	a_8	a_9	a_3	a_2	a_1	a_{10}	a_5	a_4	a_7	a_6
频率：	19	13	12	12	11	8	5	2	2	1
辅助变量：	41	16	8	2	1	0	0	0	0	0

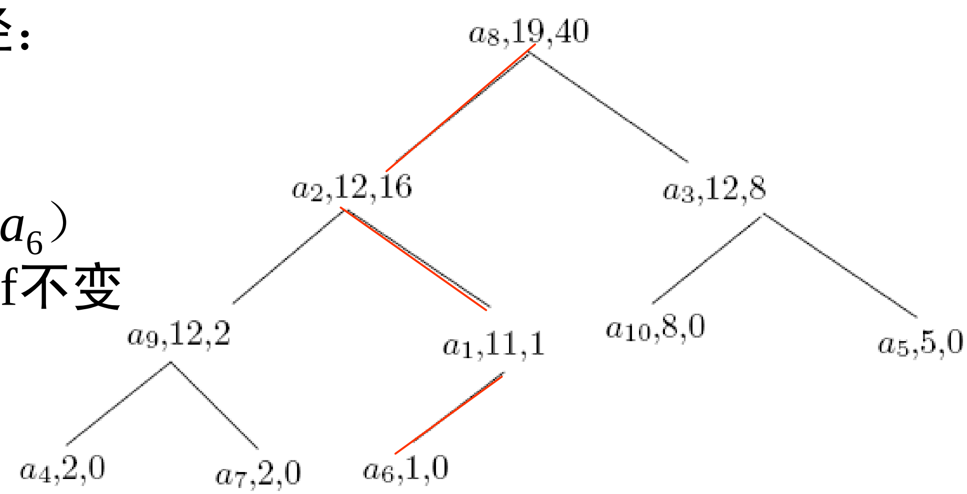
辅助变量为该节点左子树的总计数，用于计算累积频率

例：计算a_6 的累积频率

1. 得到从节点10（a_6）到根节点的路径：

   10→ 5 → 2 →1

   $a_6 \rightarrow a_1 \rightarrow a_2 \rightarrow a_8$



2. 令 af=0, 沿树枝从根节点向节点10（a_6）

   1) 取根节点的左分支到子节点a_2，af不变

   2) 取节点a_2的右分支到到节点a_1，将该节点的两个数值加到af

   af = af + 12 + 16 = 28

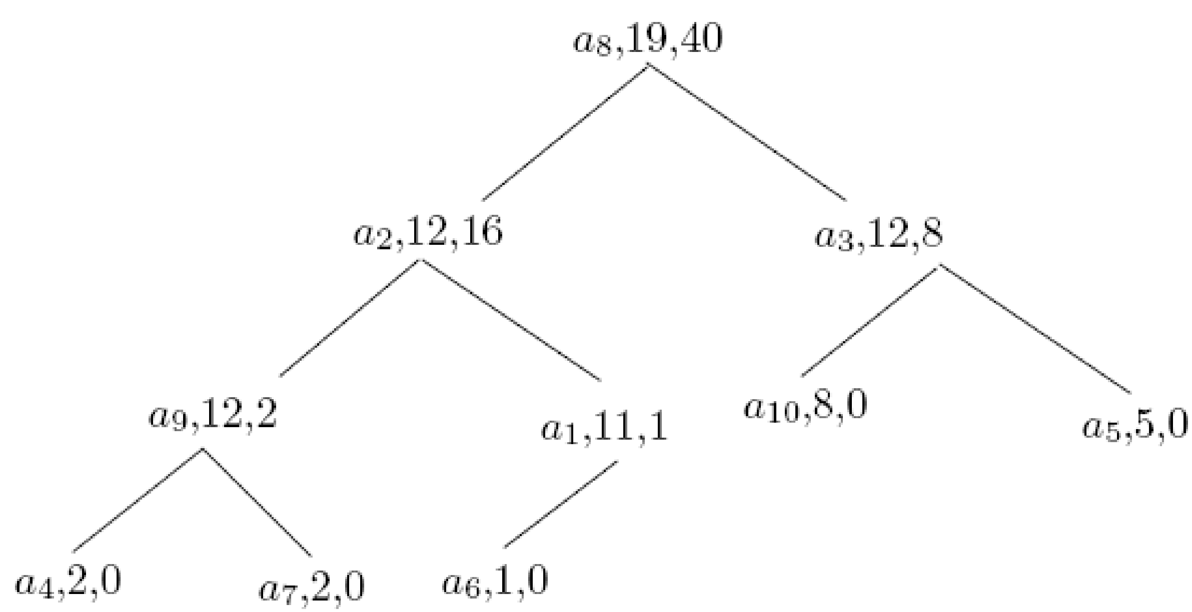
   2)取节点a_1的左分支到到节点a_6后，将其左子树计数0加至af，最后af=28为累积频率区间的起点

自适应算术编码

数组下标：	1	2	3	4	5	6	7	8	9	10
信源符号：	a_8	a_9	a_3	a_2	a_1	a_{10}	a_5	a_4	a_7	a_6
频率：	19	13	12	12	11	8	5	2	2	1
辅助变量：	41	16	8	2	1	0	0	0	0	0

频率、累积频率表：

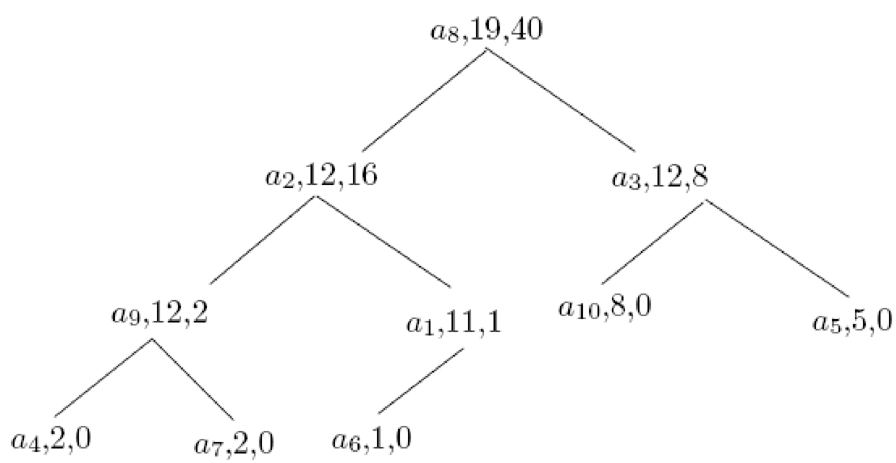
a_4	2	[0,2)
a_9	12	[2,14)
a_7	2	[14,16)
a_2	12	[16,28)
a_6	1	[28,29)
a_1	11	[29,40)
a_8	19	[40,59)
a_{10}	8	[59,67)
a_3	12	[67,79)



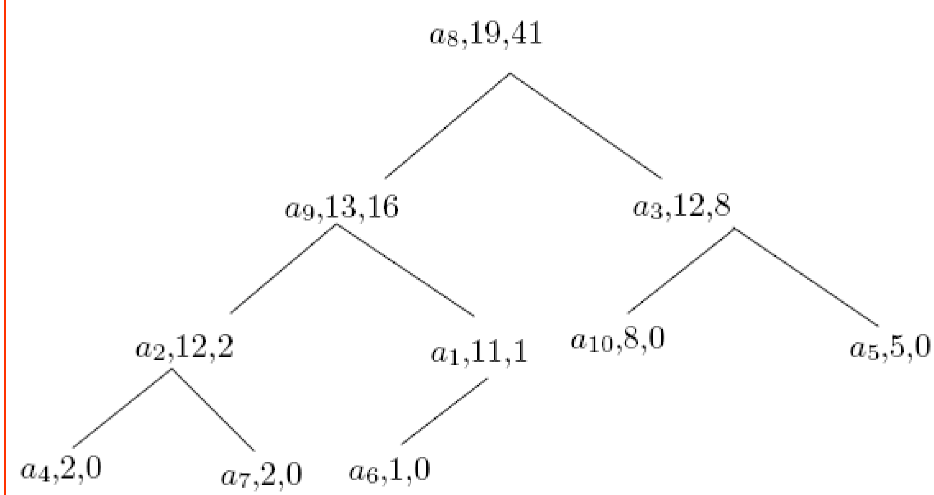
自适应算术编码

- 在平衡二叉树上更新频率：
  - 例：读进符号a_9，将其频率计数从12变为13
  - 在数组中寻找符号a_9的左边、离a_9最远的、频率小于a_9的元素，同时更新左子树计数值

a_8	a_2	a_3	a_9	a_1	a_{10}	a_5	a_4	a_7	a_6
19	12	12	12	11	8	5	2	2	1
40	16	8	2	1	0	0	0	0	0



a_8	a_9	a_3	a_2	a_1	a_{10}	a_5	a_4	a_7	a_6
19	13	12	12	11	8	5	2	2	1
41	16	8	2	1	0	0	0	0	0



总结

- 直接对序列进行编码（不是码字的串联）：非分组码
- 可证明是唯一可译码
- 渐近接近熵界
- 对不均衡的分布，比Huffman更有效
- 只产生必要的码字
- 但是实现更复杂
- 允许将建模和编码分开，容易与自适应模型和上下文模型结合
- 对错误很敏感，如果有一位发生错误就会导致整个消息译码错误

下节课内容

- 作业：
  - Sayood 3rd, pp.114-115
    - 必做：5, 6, 7, 8

- 下节课内容：
  - JPEG、JPEG2000和JBIG中的算术编码： QM编码器

History

- The idea of encoding by using cumulative probability in some ordering, and decoding by comparison of magnitude of binary fraction was introduced in Shannon's celebrated paper [1948].
- The recursive implementation of arithmetic coding was devised by Elias (another member in Fano's first information theory class at MIT).
  This unpublished result was first introduced by Abramson as a note in his book on information theory and coding [1963].
- The result was further developed by Jelinek in his book on information theory [1968].
- The growing precision problem prevented arithmetic coding from practical usage, however. The proposal of using finite precision arithmetic was made independently by Pasco [1976] and Rissanen [1976].

History

- Practical arithmetic coding was developed by several independent groups [Rissanen 1979, Rubin 1979, Guazzo 1980].
- A well-known tutorial paper on arithmetic coding appeared in [Langdon 1984].
- The tremendous efforts made in IBM lead to a new form of adaptive binary arithmetic coding known as the Q-coder [Pennebaker 1988].
- Based on the Q-coder, the activities of JPEG and JBIG combined the best features of the various existing arithmetic coders and developed the binary arithmetic coding procedure known as the QM-coder [pennebaker 1992].

Reading

- W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, Jr., R. B. Arps, “An overview of the basic principles of the Q-Coder adaptive binary arithmetic coder,” IBM J. Res. Develop., vol. 32, no. 6, November 1988.
- Witten, Radford, Neal, Cleary, “Arithmetic Coding for Data Compression” Communications of the ACM, vol. 30, no. 6, pp. 520-540, June 1987.
- Moffat, Neal, Witten, “Arithmetic Coding Revisited,” ACM Transactions on Information Systems, vol. 16, vo. 3, pp. 256–294, July 1998.

附：产生随机样本

- 产生随机样本：对分布采样
  - 均匀分布
    - 伪随机数
    - 很多统计软件包中都有此工具
      - 如在Matlab中：rand
  - 其他分布
    - 直接方法：概率积分变换
      - 通过对均匀分布的采样实现对任意分布的采样
    - 间接方法：
      - 接受/拒绝算法（重要性采样）
      - MCMC方法

概率积分变换

- 例：[概率积分变换] X有连续CDF F_X，定义随机变量Y为$Y=F_X(X)$，则Y为[0,1]上的均匀分布，即

$$\mathbb{P}(Y \le y) = y, \quad 0 \le y \le 1$$

- 对随机数产生特别有用

证明：$Y=F_X(X)$， $\therefore\quad 0\le y\le 1$

定义 F_X 的反函数为分位函数 F_X^{-1}，即

$$F_X^{-1}=\inf\left\{x:F_X\left(x\right)\ge y\right\}$$

则

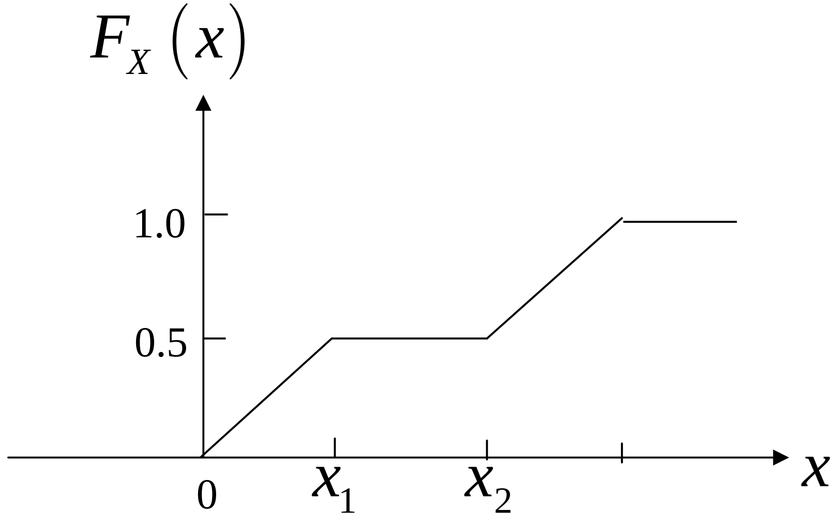
$$\mathbb{P}\left(Y\le y\right)=\mathbb{P}\left(F_X(X)\le y\right)$$

$$=\mathbb{P}\left(F_X^{-1}\left(F_X(X)\right)\le F_X^{-1}\left(y\right)\right)$$ （F_X^{-1} 为增函数）

$$=\mathbb{P}\left(X\le F_X^{-1}\left(y\right)\right)$$ （右边图示）

$$=F_X\left(F_X^{-1}\left(y\right)\right)$$ （F_X 的定义）

$$=y$$ （F_X 的连续性）



假设 $[x_1,x_2]$ 为 F_X 的平坦区域

$$\forall x\in[x_1,x_2]$$

$$F_X^{-1}\left(F_X(x)\right)=x_1$$

$$\mathbb{P}\left(X\le x\right)=\mathbb{P}\left(X\le x_1\right)$$

概率不等式仍然成立

概率积分变换

- X有连续严格递增的CDF F_X，定义随机变量Y为 $Y = F_X(X)$，则Y为[0,1]上的均匀分布，即
  $Y \sim Uniform(0,1) \quad \mathbb{P}(Y \le y) = y, \quad 0 \le y \le 1$

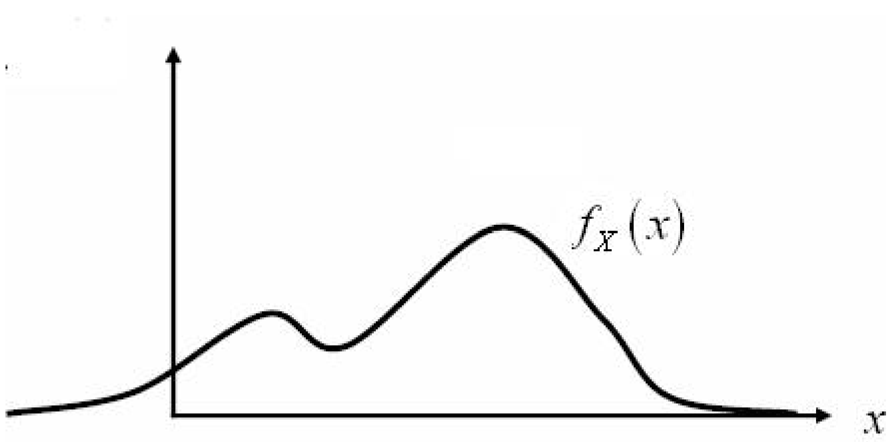
- 令 $X \sim F_X^{-1}(Y), Y \sim Uniform(0,1)$
- 则

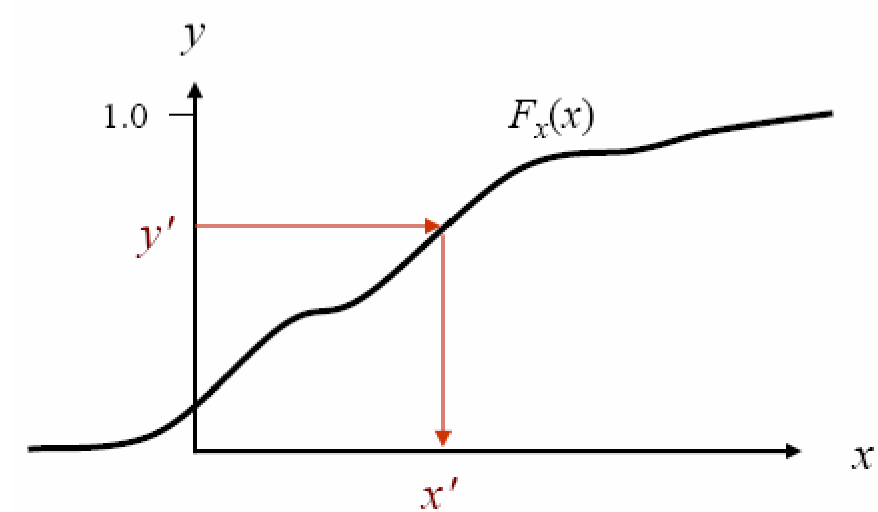
$$
\begin{aligned}
\mathbb{P}\left(X \le x\right) &= \mathbb{P}\left(F_X^{-1}\left(Y\right) \le x\right) \\
&= \mathbb{P}\left(Y \le F_X\left(x\right)\right) \\
&= F_X\left(x\right) \qquad \left(\quad \mathbb{P}\left(Y \le c\right) = c \quad \right)
\end{aligned}
$$

$$\boxed{\therefore \quad X \sim F_X}$$

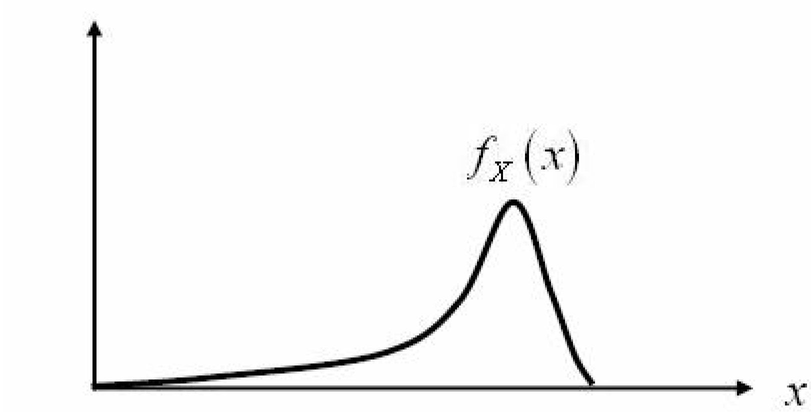
对任意分布采样

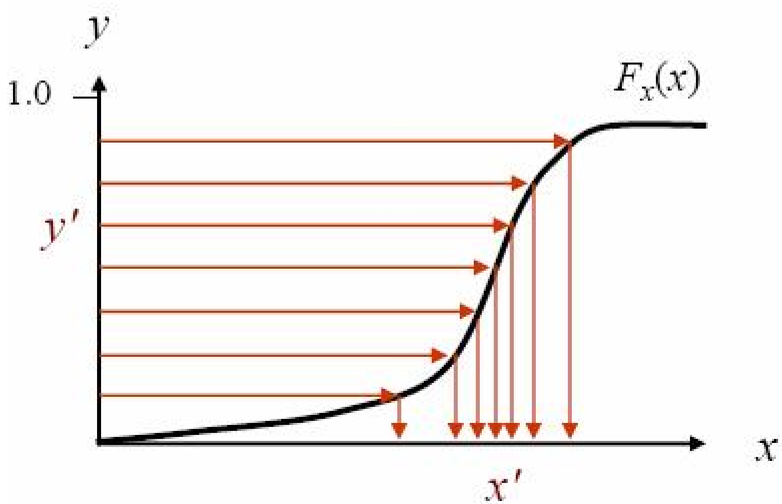
- 通过对均匀分布采样，实现对任意分布的采样
  - 从 $Uniform(0,1)$随机产生一个样本y
  - 令 $y = F_X(x)$ 其中$F_X(x)$为X的CDF
  - 计算 $x = F_X^{-1}(y)$
  - 结果 x为对 $f_X(x)$的采样





对任意分布采样





对任意分布采样

- 例：对指数分布采样

$$X \sim Exponential(\beta) \qquad f_X(x) = \frac{1}{\beta} e^{-x/\beta}$$

$$F_X(x) = \int_{-\infty}^{x} f(x)\,dx = 1 - \int_{x}^{\infty} f(x)\,dx$$

$$= 1 - \int_{x}^{\infty} \frac{1}{\beta} e^{-x/\beta} dx = 1 - e^{-x/\beta}$$

$$Y \sim Uniform(0,1)$$

$$y = F_X(x) = 1 - e^{-x/\beta}$$

$$x = F_X^{-1}(y) = -\beta \ln(1 - y)$$

变形

- 若X为离散型随机变量，其取值为 $x_1 < x_2 < \ldots < x_k$,
- 则可以通过以下方式产生随机样本 $X \sim F_X(x)$
  - 从$Uniform(0,1)$随机产生一个样本y
  - 若 $F_X(x_i) < y < F_X(x_{i+1})$，令 $x = x_{i+1}$
- 定义 $x_0 = -\infty, F_X(x_0) = 0$

- 例：为了从 $X \sim Bernoulli(p)$产生一个随机样本，从 $Y \sim Uniform(0,1)$产生一个随机样本y，则

$$X = \begin{cases} 0 & if \;\; 0 < y \le 1 - p \\ 1 & if \;\; 1 - p < y \le 1 \end{cases}$$