

مروری بر نظریه‌ی اطلاعات  
کدگذاری بی‌اتلاف

## فشرده‌سازی اطلاعات

۰۱-۷۰۲-۱۰-۱۴۰

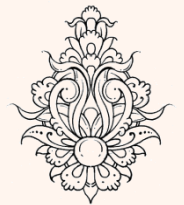
## بخش نخست



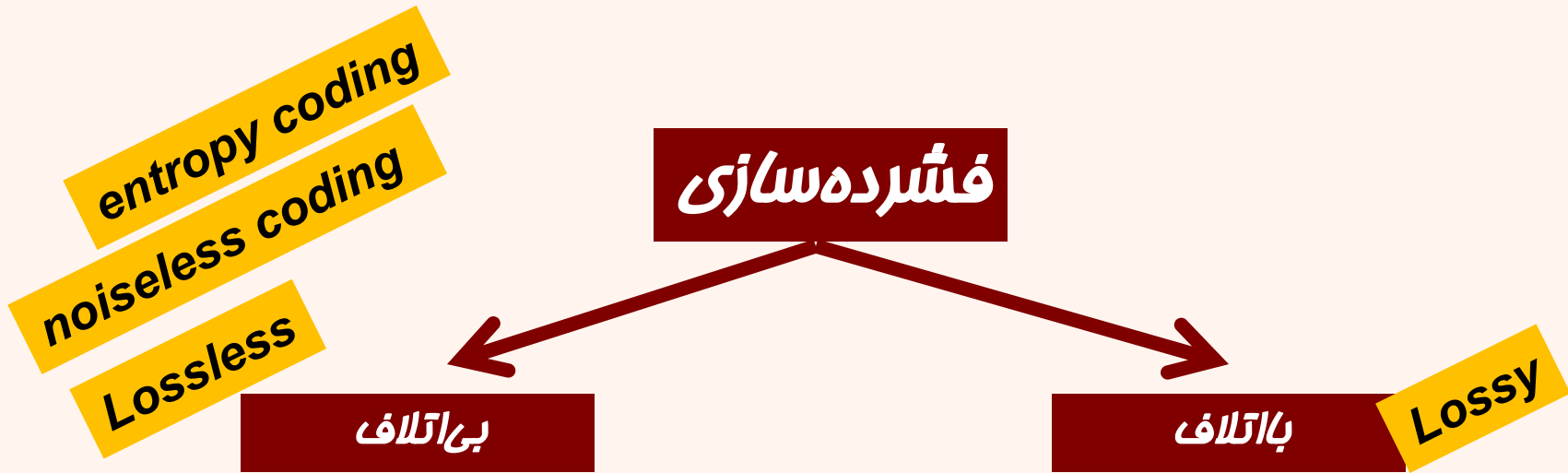
دانشگاه شهید بهشتی  
پژوهشکده‌ی فضای مجازی  
زمستان ۱۳۹۷  
احمد محمودی ازناوه

# فهرست مطالب

- انواع فشرده‌سازی
- اندازه‌گیری اطلاعات
- مفهوم آنتروپی
- نیازمندی‌های یک کدگذار
- نامساوی Kraft-McMillan
- کدهای پیشوندی
- کدگذاری Shannon-Fano
- کدگذاری Huffman
- کدگذاری Unary
- کدگذاری Golomb
- کدگذاری محاسباتی

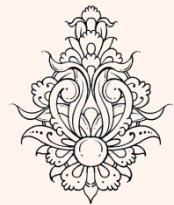


# انواع فشرده‌سازی



داده‌ی فشرده‌شده پس از بازیابی کاملاً با داده‌ی اصلی یکسان است.

داده‌ی فشرده‌شده پس از بازیابی با داده‌ی اصلی یکسان **نیست**.



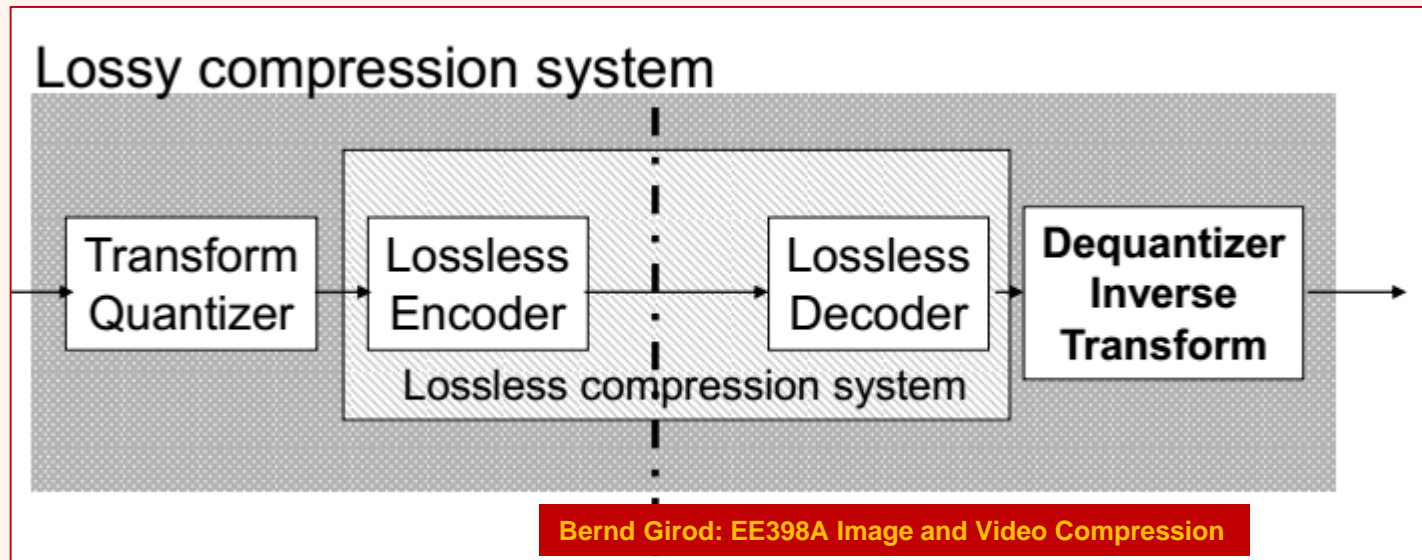
**فشرده‌سازی متن**  
تصاویر پزشکی-تصاویر نظامی

**نرخ فشرده‌گی بالا**

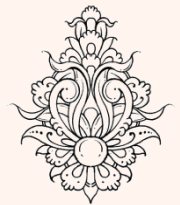


# فشرده‌سازی بی‌اتلاف

- تقریباً در بطن همه‌ی سیستم‌های فشرده‌سازی «**بی‌اتلاف**» یک سیستم «**بی‌اتلاف**» قرار دارد:



- از این رو ابتدا به بررسی سیستم‌های «فشرده‌سازی بی‌اتلاف» می‌پردازیم.



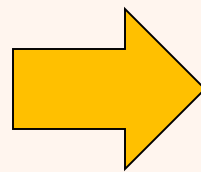
# اندازه‌گیری اطلاعات

• اگر تعدادی نماد در دست باشد، به وسیله‌ی رابطه‌ی *Shannon* کمینه‌ی تعداد بیت‌ها برای ارسال نمادهای مورد نظر محاسبه می‌گردد.



• اگر  $M$  تعداد نمادها باشد:

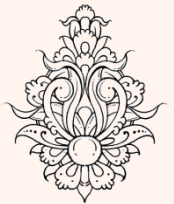
$$Ent = -\sum_{i=1}^M \rho_i \log_2^{\rho_i} \quad 0 \leq \rho_i \leq 1$$



*Average Numbit = Ent*

$$0 \leq Ent \leq \log_2^M$$

• اگر داشته باشیم: *AAAAABBCDE*



$$A = 0.5 \quad B = 0.2 \quad C = 0.1 \quad D = 0.1 \quad E = 0.1$$

$$Ent = -[(0.5 \log_2^{0.5} + 0.2 \log_2^{0.2} + (0.1 \log_2^{0.1}) \times 3)]$$

$$Ent = -[-0.5 + (-0.46438) + (-0.9965)]$$

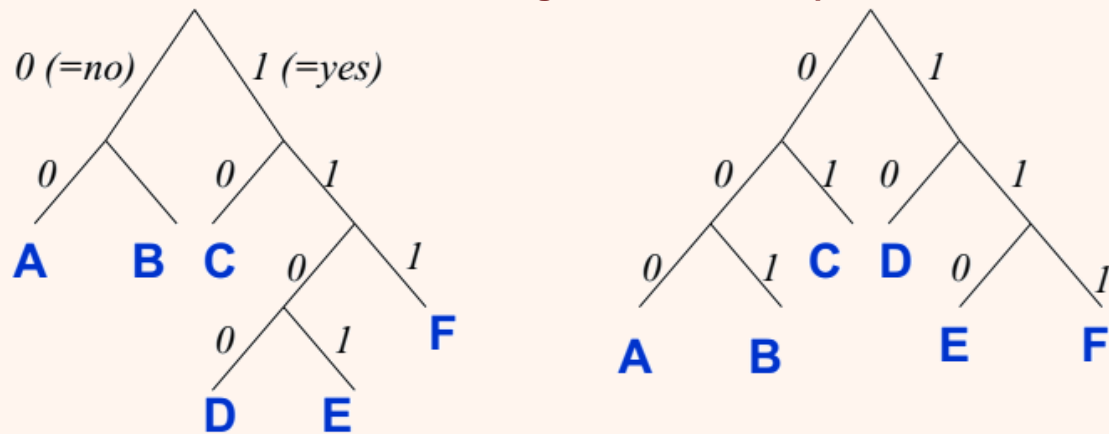
$$Ent = -[-1.9] = 1.9$$



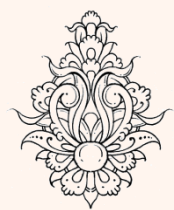
# یک مثال ساده

- یک مسابقه‌ی «بیست سؤالی» را در نظر بگیرید:
- یک خروجی از یک مجموعه‌ی محدود در نظر گرفته می‌شود:  $\{A, B, C, D, E, F\}$
- کدام استراتژی برای رسیدن به پاسخ بهتر است؟

Bernd Girod: EE398A Image and Video Compression



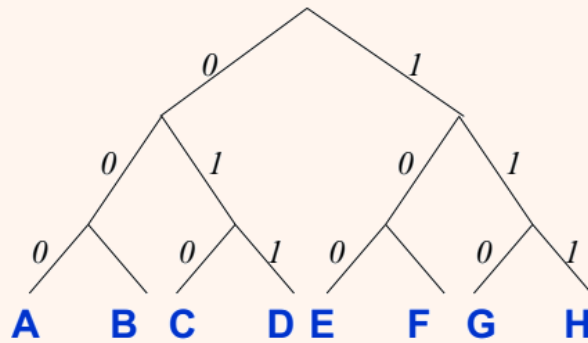
**نمونه‌ی انتخاب با یک رشته بیتی قابل نمایش است.**



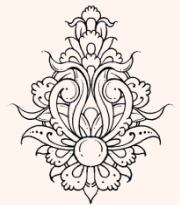
# نمایش با طول ثابت

طول متوسط این رشته‌ی **بیتی** توصیف‌کننده برای  $M$  نماد:

$$l_{av} = \log_2 M$$



در صورتی که احتمال وقوع هر نماد  $1/M$  باشد، این شیوه‌ی کدگذاری بهینه است.

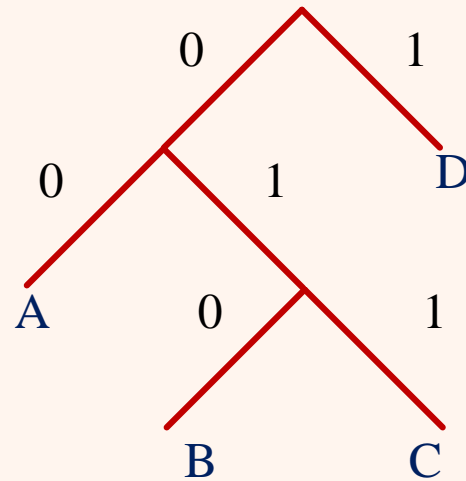


# نمایش با طول متغیر

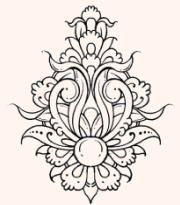
- در صورتی که احتمال انتخاب نمادها یکسان نباشد:



## Binary Code Trees



– **بهتر است** نمادهایی که احتمال رخداد بالاتری دارند، با رشته‌ی کوتاه‌تری توصیف شوند و بالعکس

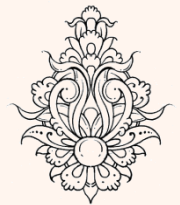




# کدهای با طول متغیر

- با افزودن یک بیت، میزان اطلاعات قابل نمایش دو برابر می‌شود (رابطه‌ی نمایی).
- اگر میزان اطلاعات یک متغیر تصادفی دو برابر شود (احتمال وقوع نصف شود)، به یک بیت بیشتر برای نمایش احتیاج دارد (رابطه‌ی لگاریتمی).
- در این حالت تعداد **بیت** برای نمایش یک نماد با احتمال وقوع  $p$ :

$$\log_2 \left( \frac{1}{p} \right)$$



# آنتروپی یک متغیر تصادفی

- فرض کنیم یک متغیر تصادفی شامل تعداد محدودی الفبا باشد:

$$A_x = \{\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{M-1}\}$$

$$f_X(x) = P(X = x); PMF$$

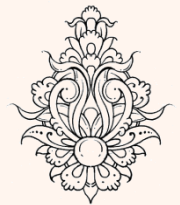
- اطلاعات متناسب با  $X=x$

$$h_X(x) = -\log_2 f_X(x)$$

- آنتروپی، مقدار متوسط (امید ریاضی) اطلاعات نمادهاست:

$$H(X) = E[h_X(X)] = -\sum_{x \in A_x} f_X(x) \log_2 f_X(x)$$

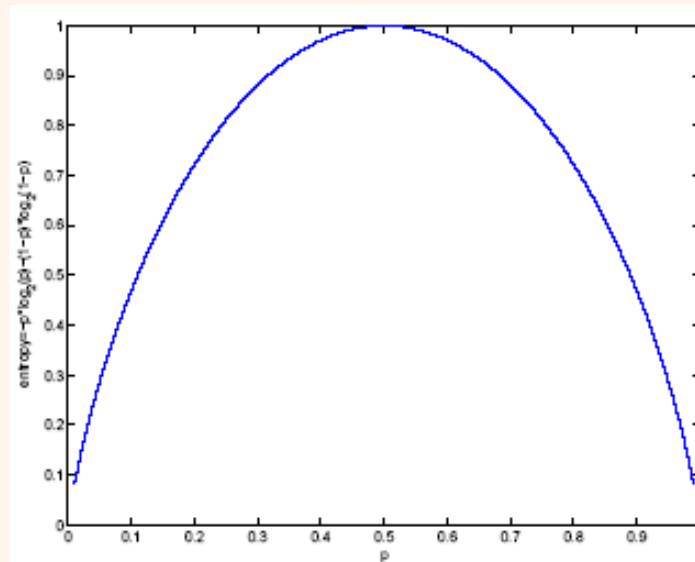
واحد اندازه گیری: بیت



# آنتروپی برای دو دسته

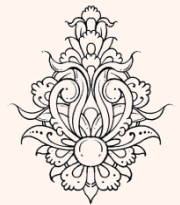
– آنتروپی اطلاعات: مقدار اتفاقی تا چه حد تصادفی است.

$$H(X) = -p \log_2 p - (1-p) \log_2 (1-p)$$

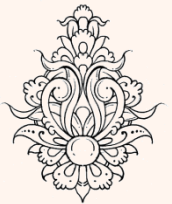


$$\rho = R - H(X) \geq 0; \quad R: \text{average length code}$$

میزان افزونگی



- با توجه به اینکه اطلاعات موجود یا نمادهای تولید شده در یک پیام دارای احتمال وقوع یکسان نیستند، می‌توان اطلاعات با احتمال وقوع کمتر را با تعداد بیت بیشتر و اطلاعات با احتمال وقوع بیشتر را با تعداد بیت کمتر **کد** کرد.



- مجموعه نمادهایی (حروف) که باید ذخیره (ارسال) شوند، «الفبا» نامیده می‌شود.

Alphabet

Letter

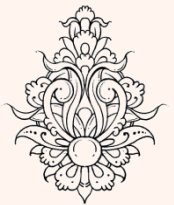
- به رشته‌ی بی‌تی که به هر «نماد» (حرف) نسبت داده می‌شود، «کلمه‌کد» گفته می‌شود.

Codeword

– هر کلمه‌کد دارای معنی واحد می باشد و طبق یک قاعده خاص با یک کد متناظر می باشد.

Code

- به مجموعه‌ی «کلمه‌کدها»، «کد» گفته می‌شود.
- به فرآیند انتساب «کد» به حروف الفبا «کدگذاری» گفته می‌شود.



# نیازمندی‌های یک کدگذار

- یک کد خوب باید شرایط زیر را داشته باشد:

**Non-singular**

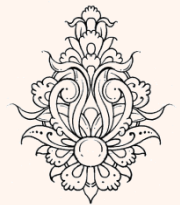
کد اختصاص یافته به هر نماد یکتا باشد.

**Uniquely decodable**

دنبالهی نمادهای کد شده با هر ترتیبی قابل کدگشایی باشد.

**Instantaneously decodable**

دنبالهی نمادهای کد شده دریافت شده بدون توجه به مابقی داده‌ها (هنوز دیده نشده) قابل کدگشایی باشد.



# مثال ۱

## کدگذاری نمادها

$\alpha_i$	Code word
$\alpha_0$	0
$\alpha_1$	01
$\alpha_2$	10
$\alpha_3$	11

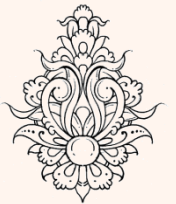
$\alpha_0, \alpha_2, \alpha_3, \alpha_0, \alpha_1$

0 10 11 0 0

$\alpha_1, \alpha_0, \alpha_3, \alpha_0, \alpha_1$

01 0 11 0 0

کدگذاری به صورت یکتا ممکن نیست!



# مثال ۲

$\alpha_i$	$p(\alpha_i)$	Code A	Code B	Code C	Code D	Code E
$\alpha_0$	0.5	0	0	0	00	0
$\alpha_1$	0.25	10	01	01	01	10
$\alpha_2$	0.125	11	010	011	10	110
$\alpha_3$	0.125	11	011	111	110	111
	$l_{av}$	1.5	1.75	1.75	2.125	1.75

**non-singular**



**uniquely decodable**



**instantaneously decodable**





# تمرین

## کد یک

$\alpha_i$	Code word
$\alpha_0$	0
$\alpha_1$	10
$\alpha_2$	110
$\alpha_3$	111

## کد دو

$\alpha_i$	Code word
$\alpha_0$	0
$\alpha_1$	01
$\alpha_2$	100
$\alpha_3$	011

با توجه به کدهای فوق رشته‌ی بیتی زیر را کدگشایی کنید.

0 0 1 1 0 1 0 1 1 0 1 0 0

## کد یک

0|0|1|1|0|1|0|1|1|0|1|0|0

$\alpha_0 \alpha_0 \alpha_2 \alpha_1 \alpha_2 \alpha_1 \alpha_0$

## کد دو

0|0|1|1|0|1|0|1|1|0|1|0|0

$\alpha_0 \alpha_3 \alpha_1 \alpha_3 \alpha_0 \alpha_2$

به صورت یکتا کدگشایی نمی‌شود!

$\alpha_0 \alpha_3 \alpha_1 \alpha_3 \alpha_1 \alpha_0 \alpha_0$

در صورتی که یک کلمه کد پیشوند دیگری باشد، به پسوند کلمه کد دو

**Dangling Suffix**

«پسوند معلق» می‌گویند.



# شیوهی تشخیص قابلیت کدگشایی

Sardinas-Patterson algorithm(1953)

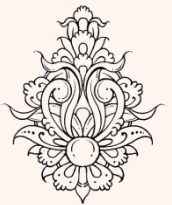
- یک لیست از کلمه‌کدها ایجاد کنید.
- کلمه‌کدها را دوبه‌دو بررسی کنید که آیا یکی پیشوند دیگری است؟
- در این صورت، پسوند معلق را به لیست بیفزایید(چنانچه پیش از این اضافه نکرده باشید).

– این فرآیند را تا زمانی ادامه دهید که:

- به یک پسوند معلق برسید که کلمه‌کد باشد.
- پسوند معلق دیگری نباشد.

قابل کدگشایی نیست.

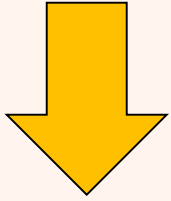
به صورت یکتا قابل کدگشایی است.



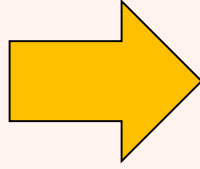
Sardinas, A. A. and G. W. Patterson (1953). A necessary and sufficient condition for unique decomposition of coded messages. PROCEEDINGS OF THE INSTITUTE OF RADIO ENGINEERS.

# مثال

{0, 01, 11}

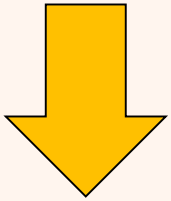


{0, 01, 11, 1}

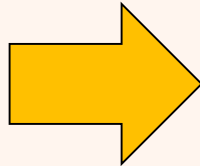


قابل کدگشایی به صورت یکتاست

{0, 01, 10}

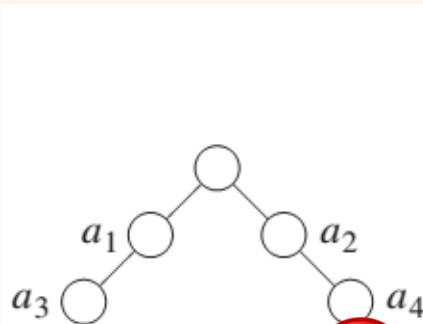


{0, 01, 10, 1}

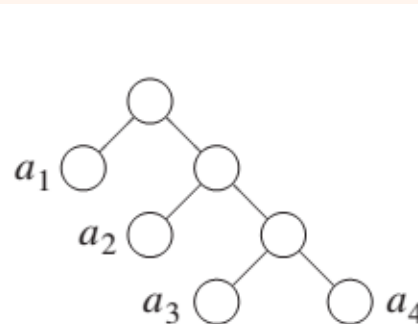


پسوند معلق یک کلمه کد است، در نتیجه قابل کدگشایی نیست.

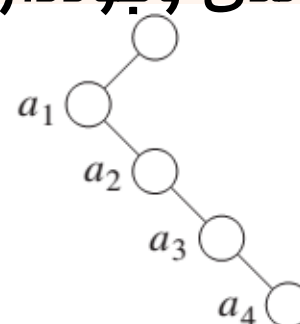
- به کدی که هیچ کلمه‌کدی از آن پیشوند دیگری نباشد، «کد پیشوندی» می‌گویند.
- با توجه نمونه‌ی تشخیص قابلیت کدگشایی، چنین کدی همواره قابل کدگشایی خواهد بود.
- به ازای هر کد پیشوندی یک درخت دودویی می‌توان رسم کرد که به هر برگ یک کلمه‌کد نسبت داده شده است.
- پیشوندی بودن کد باعث ایجاد هیچ محدودیتی نمی‌شود. ثابت می‌شود به ازای هر کد قابل کدگشایی، یک کد پیشوندی وجود دارد.



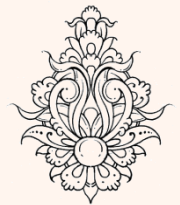
Code 2

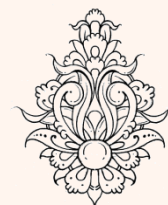
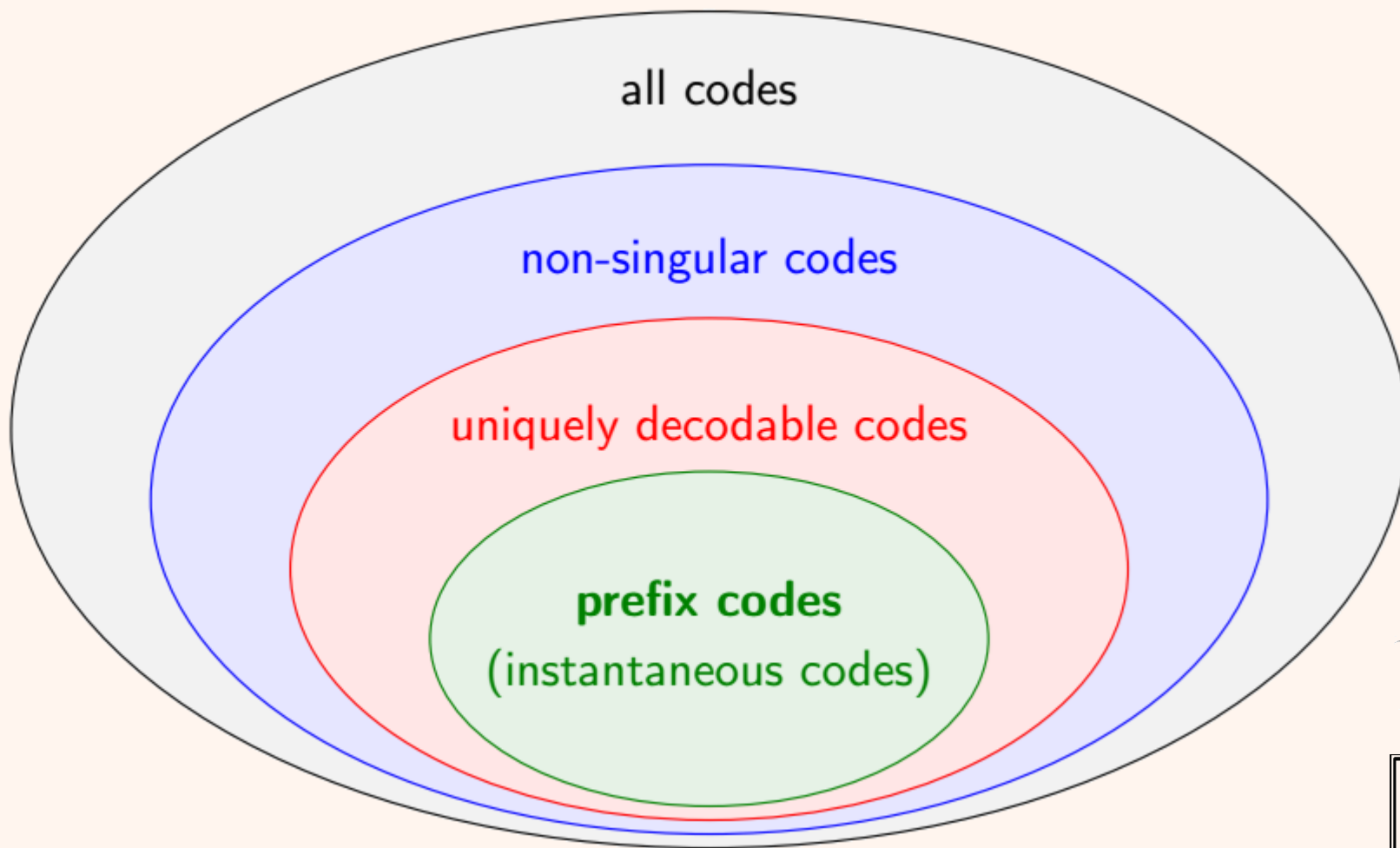


Code 3



Code 4





# Kraft-McMillan inequality

- شرط لازم برای این که یک کد به صورت یکتا قابل کدگشایی باشد:

$$\sum_{i=0}^{M-1} 2^{-l(\alpha_i)} \leq 1$$

اثبات در (2012) Sayood, K. آمده است.

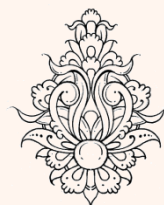
قضیه ۱

- چنانچه یک کد در شرایط فوق صدق کند، یک کد پیشوندی با کلمه‌کد با اندازه‌های مورد نظر وجود خواهد داشت.

$$l_0 \leq l_1 \leq \dots \leq l_{M-1}$$

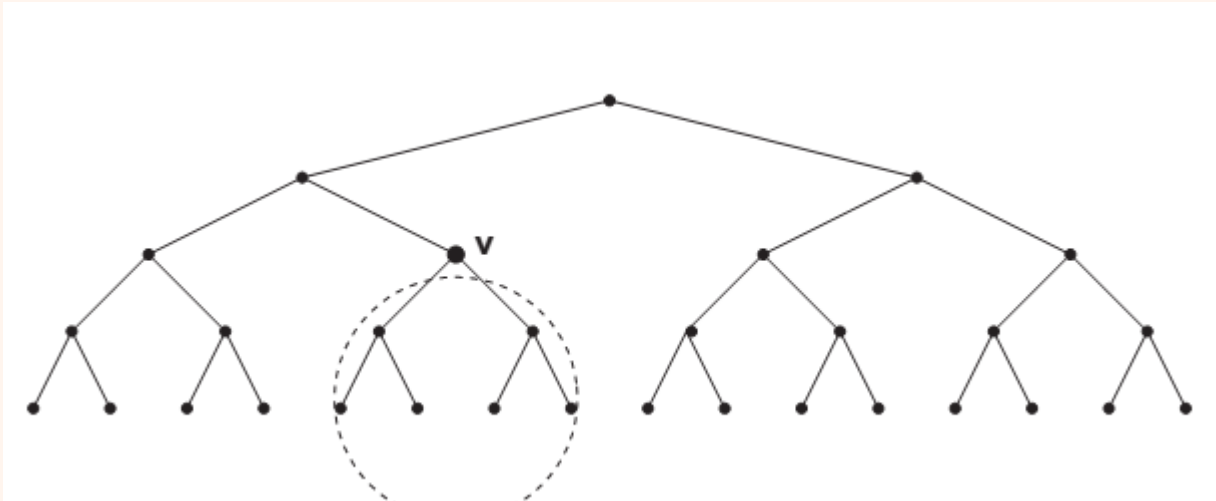
اندازه‌ی کلمه‌کدها

$$\sum_{i=0}^{M-1} 2^{-l_i} \leq 1$$



یک درخت دودویی کامل با عمق  $l$  ایجاد می‌کنیم:

$$l = \max \{l_1, l_2, \dots, l_{M-1}\}$$



در صورت حذف یک زیردرخت از عمق  $k$ ،  $2^{l-k}$  گرهی برگ حذف می‌شوند.

مجموع برگ‌هایی که برای تشکیل کد باید حذف شوند:

$$\sum_{i=0}^{M-1} 2^{l-l_i} = 2^l \sum_{i=0}^{M-1} 2^{-l_i} \leq 2^l$$



ژانسیکا

در نتیجه تعداد گره‌های برگ برای سافت پنین کدی کفایت می‌کند ✨

# Shannon–Fano coding

- در این شیوه بر اساس احتمال وقوع نمادها پس از مرتب کردن به دو دسته‌ی تقریباً با احتمال مساوی تقسیم می‌شوند.
- به یک دسته بیت صفر و به دسته‌ی دیگر بیت یک اختصاص می‌یابد.
- این تقسیم‌بندی به همین شیوه ادامه می‌یابد.
- این شیوه‌ی کدگذاری **بهینه** نیست.

$a_i$	$p(a_i)$	1	2	3	4	Code	
$a_1$	0.36	0	00			00	
$a_2$	0.18		01			01	
$a_3$	0.18	1	10			10	
$a_4$	0.12		11	110		110	
$a_5$	0.09			111	1110		1110
$a_6$	0.07				1111		1111

*Fano, R. M. (1949). The transmission of information, Massachusetts Institute of Technology, Research Laboratory of Electronics Cambridge, Mass, USA.*

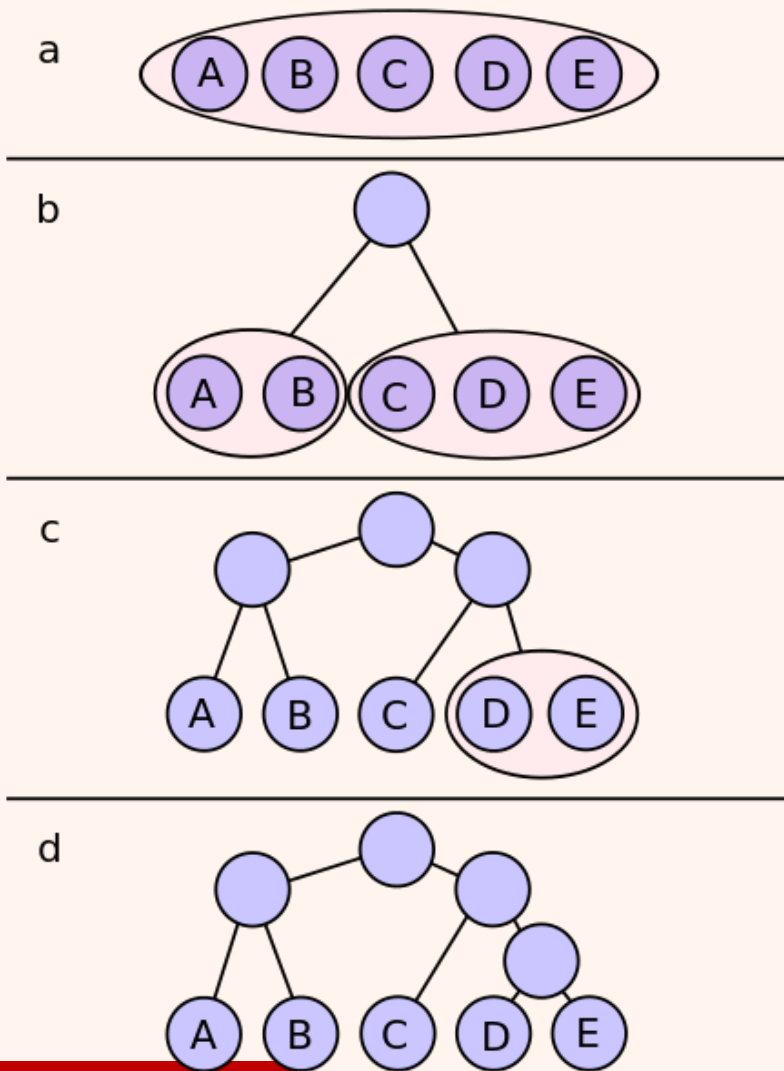
wikipedia





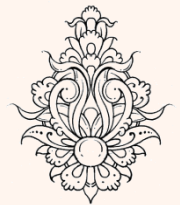
# Shannon–Fano coding

- در این شیوه (بر خلاف روش Huffman) درخت از سمت ریشه رشد می‌کند.

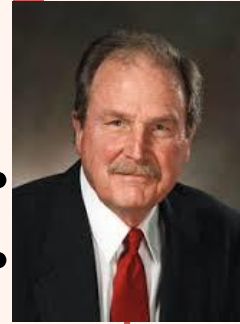


wikipedia

فشرده‌سازی



# کدگذاری Huffman

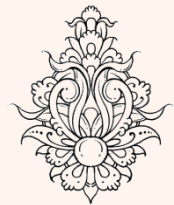


• در سال ۱۹۵۲ توسط David Huffman مطرح شد.  
• این شیوه prefix است و متوسط طول کد در آن بهینه است.  
• در این شیوه ابتدا اصولی برای کدهای بهینه مطرح و سپس بر اساس آن شیوهی کدگذاری به دست می‌آید:

- کد اختصاص یافته به هر نماد یکتاست.
- اطلاعات با احتمال وقوع کمتر را با تعداد بیت بیشتر و اطلاعات با احتمال وقوع بیشتر را با تعداد بیت کمتر کد شوند.
- دنباله‌ی نمادهای کد شده دریافت شده بدون توجه به مابقی داده‌ها (هنوز دیده نشده) قابل کدگشایی باشد.
- دو نماد با کمترین احتمال با طول یکسان کد می‌شوند.
- کلمه‌کدهای دو نماد با کمترین احتمال تنها در بیت آخر متفاوت هستند.

$$P(1) \geq P(2) \geq \dots \geq P(N-1) \geq P(N)$$

$$L(1) \leq L(2) \leq \dots \leq L(N-1) = L(N)$$



# مثال

Letter	Probability	Codeword
$a_2$	0.4	$c(a_2)$
$a_1$	0.2	$c(a_1)$
$a_3$	0.2	$c(a_3)$
$a_4$	0.1	$c(a_4)$
$a_5$	0.1	$c(a_5)$

- دو نماد  $\alpha_4$  و  $\alpha_5$  کمترین احتمال وقوع را دارند، پس تنها در یک بیت اختلاف خواهند داشت:

$$c(a_4) = \alpha_1 * 0$$

$$c(a_5) = \alpha_1 * 1$$

Letter	Probability	Codeword
$a_2$	0.4	$c(a_2)$
$a_1$	0.2	$c(a_1)$
$a_3$	0.2	$c(a_3)$
$a'_4$	0.2	$\alpha_1$

- به همین ترتیب خواهیم داشت:

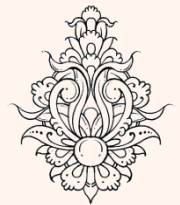
$$c(a_3) = \alpha_2 * 0$$

$$c(a'_4) = \alpha_2 * 1$$

$$\alpha_1 = \alpha_2 * 1$$

$$c(a_4) = \alpha_2 * 10$$

$$c(a_5) = \alpha_2 * 11$$



# مثال

Letter	Probability	Codeword
$a_2$	0.4	$c(a_2)$
$a'_3$	0.4	$\alpha_2$
$a_1$	0.2	$c(a_1)$

$$c(a'_3) = \alpha_3 * 0$$

$$c(a_1) = \alpha_3 * 1$$

$$c(a_3) = \alpha_3 * 00$$

$$c(a_4) = \alpha_3 * 010$$

$$c(a_5) = \alpha_3 * 011$$

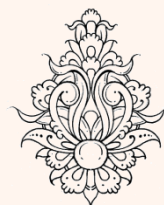
Letter	Probability	Codeword
$a''_3$	0.6	$\alpha_3$
$a_2$	0.4	$c(a_2)$

$$c(a''_3) = 0$$

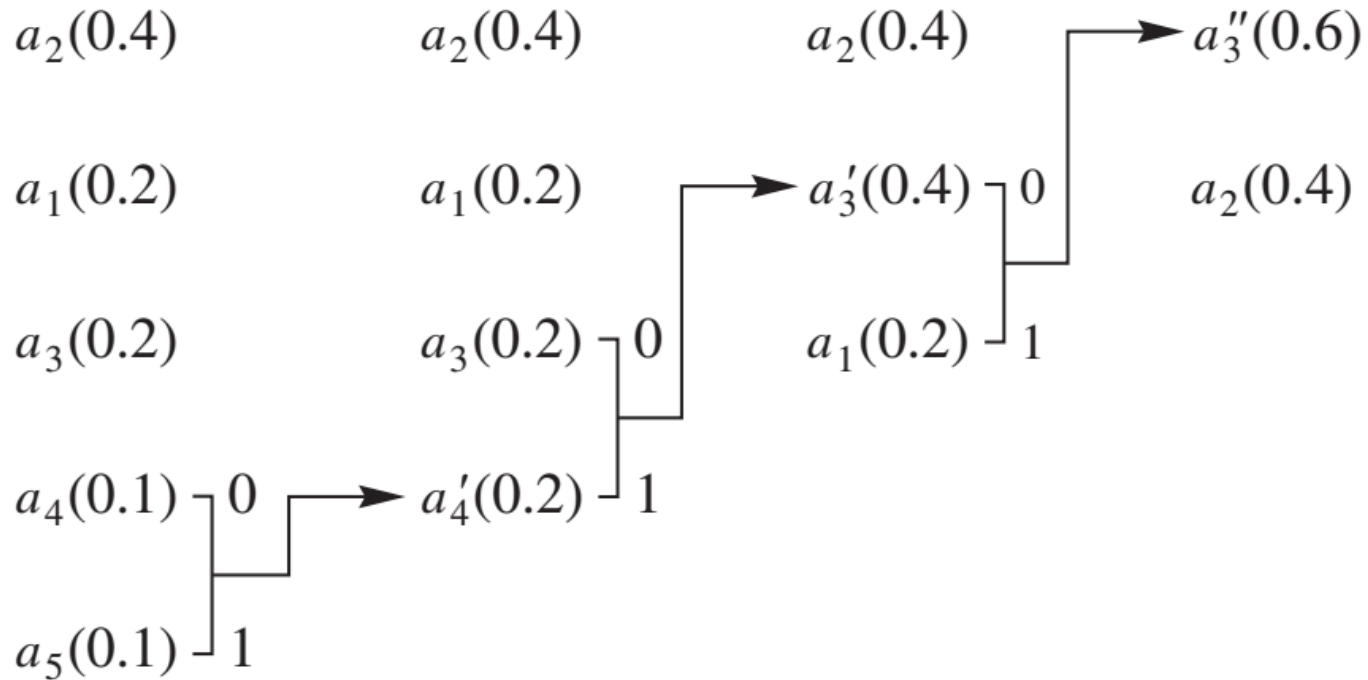
$$c(a_2) = 1$$

$$\alpha_2 = \alpha_3 * 0$$

Letter	Probability	Codeword
$a_2$	0.4	1
$a_1$	0.2	01
$a_3$	0.2	000
$a_4$	0.1	0010
$a_5$	0.1	0011

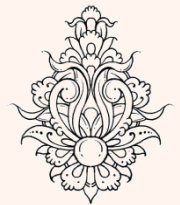


# مثال (ادامه...)



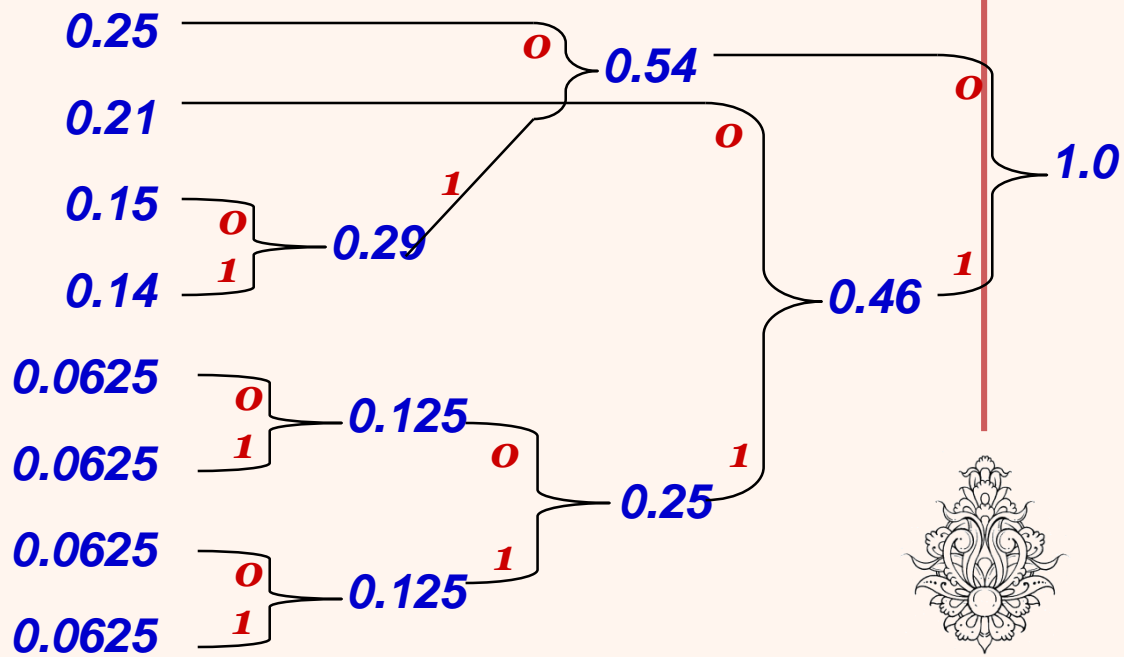
$$l = .4 \times 1 + .2 \times 2 + .2 \times 3 + .1 \times 4 + .1 \times 4 = 2.2 \text{ bits/symbol}$$

the redundancy is 0.078 bits/symbol.

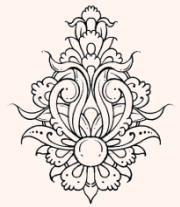


# مثال ۲

000	00	S0
001	10	S1
010	010	S2
011	011	S3
100	1100	S4
101	1101	S5
110	1110	S6
111	1111	S7



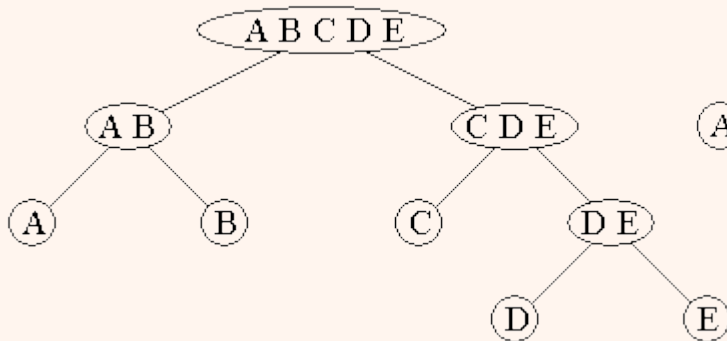
**Huffman**  
(trace from root)



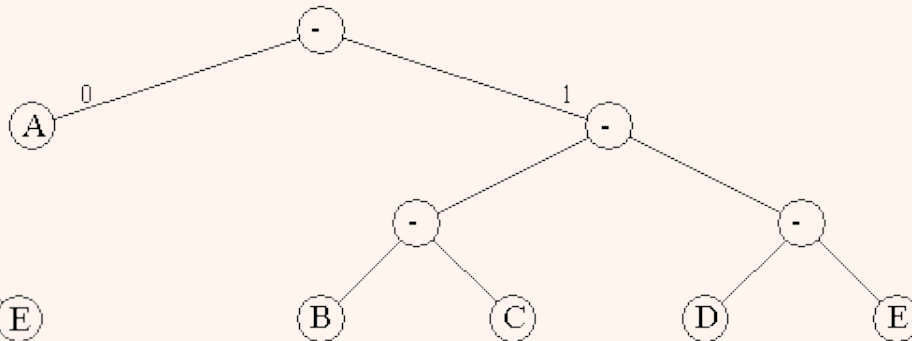
# تمرین

- جدول نمادهای زیر را به روش Huffman و Shannon-Fano کد کنید.

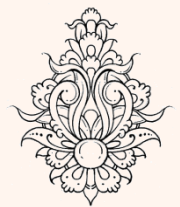
$\alpha_i$	frequency
A	24
B	12
C	10
D	8
E	8



**Shannon-Fano**



**Huffman**



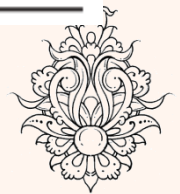
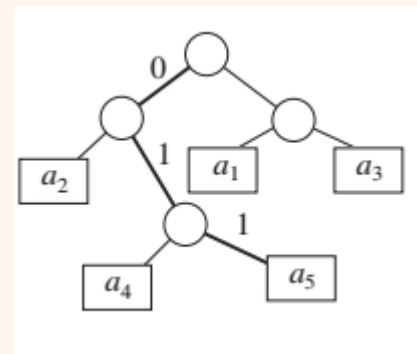
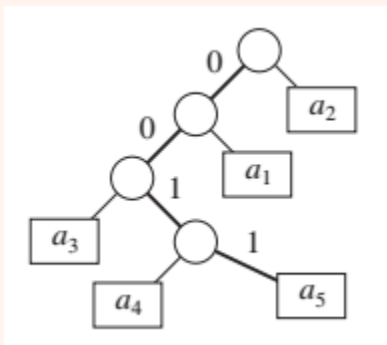
# Minimum Variance Huffman Codes

Letter	Probability	Codeword
$a_2$	0.4	$c(a_2)$
$a_1$	0.2	$c(a_1)$
$a_3$	0.2	$c(a_3)$
$a'_4$	0.2	$\alpha_1$

Letter	Probability	Codeword
$a_2$	0.4	$c(a_2)$
$a'_4$	0.2	$\alpha_1$
$a_1$	0.2	$c(a_1)$
$a_3$	0.2	$c(a_3)$

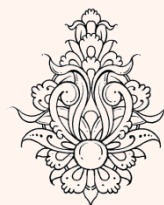
Letter	Probability	Codeword
$a_2$	0.4	1
$a_1$	0.2	01
$a_3$	0.2	000
$a_4$	0.1	0010
$a_5$	0.1	0011

Letter	Probability	Codeword
$a_1$	0.2	10
$a_2$	0.4	00
$a_3$	0.2	11
$a_4$	0.1	010
$a_5$	0.1	011





- با تخییر در ترتیب اجرای الگوریتم Huffman کدهای متفاوتی به دست می‌آید، بسته به این که نمادها در هر مرحله چگونه مرتب شوند.
- هرچند متوسط نرخ بیت در دو حالت با هم تفاوتی ندارد، اما واریانس طول کلمه‌کدها متفاوت خواهد بود.
- در بیشتر کاربردها با وجود این که می‌توان از کدهای با طول متغیر استفاده کرد، ولی پهنای باند ثابت است.
- به عنوان مثال در صورتی که ما قصد داشته باشیم ۱۰,۰۰۰ نماد در ثانیه ارسال کنیم و پهنای باند ۲۲,۰۰۰bps باشد، امکان ارسال بیش از ۲۲,۰۰۰ بیت بر ثانیه وجود نخواهد داشت.



# Minimum Variance Huffman Codes

- چنانچه در یک بازه‌ی کوتاه دو نماد  $\alpha_4$  و  $\alpha_5$  پیوسته ارسال شود:
- برای کد اول، داده‌ی ارسالی محدود  $40,000\text{bps}$  خواهد بود (۱۸۰۰۰ بیت باید بافر شوند).
- برای کد دوم، داده‌ی ارسالی محدود  $30,000\text{bps}$  خواهد بود (۸۰۰۰ بیت باید بافر شوند).
- در صورتی که به جای این دو نماد از  $\alpha_2$  استفاده شود، داده‌ی ارسالی در کد اول نرخ  $10,000$  بیت بر ثانیه خواهد داشت.
- در تمامی این حالات گیرنده انتظار دریافت  $22,000$  بیت بر ثانیه را خواهد داشت.

برای به دست آوردن کد Huffman با کمترین واریانس، حرف ترکیبی در مرتب‌سازی بالاتر از حروف تکی با احتمال یکسان قرار می‌گیرند.



- در صورتی که خصوصیات آماری داده در حال تغییر باشد، برای کدگذاری، هر بار کلمه‌کدها باید محاسبه شده و همراه با داده‌ها ارسال (ذخیره) گردد:

Letter	Probability	Codeword
$a_2$	0.4	1
$a_1$	0.2	01
$a_3$	0.2	000
$a_4$	0.1	0010
$a_5$	0.1	0011

– مثال:

- به عنوان مثال کلمه‌کدها به ترتیب الفبایی مرتب شده و ارسال شوند.

[2, 01, 1, 1, 3, 000, 4, 0010, 4, 0011]

- هرچند چنین کاری برای الفبای کوچک شدنی است، اما در صورتی که تعداد نمادها افزایش یابد، حتی ممکن است اثر فشرده‌سازی را خنثی کند.



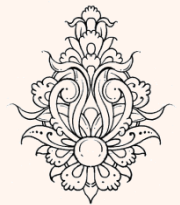
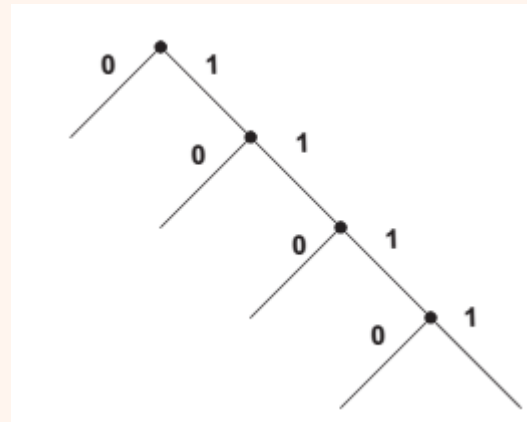


# Canonical Huffman Codes

- برای چیرگی بر این مشکل نوعی خاصی از کدگذاری پیشنهاد می‌شود:
  - نمادها به ترتیب الفبایی مرتب شوند.
  - کلمه‌کدهای کوتاه‌تر، پیش از کلمه‌کدهای بلند در ساختار درخت قرار گیرند.

{2, 1, 3, 4, 4}

Letter	Probability	Codeword
$a_2$	0.4	1
$a_1$	0.2	01
$a_3$	0.2	000
$a_4$	0.1	0010
$a_5$	0.1	0011



# افزونگی کدگذاری Huffman

$$\rho = R - H(X) \geq 0; \quad R: \text{average length code}$$

افزونگی کد Huffman همیشه کمتر از یک بیت به ازای هر نماد است:

$$0 \leq \rho < 1$$

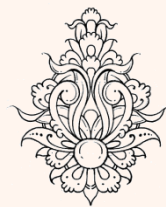
به عبارت دیگر

$$H(X) \leq R < H(X) + 1$$

نامساوی سمت چپ که  
بر اساس رابطه‌ی  
Shannon برقرار است.

نامساوی سمت راست:

$$R = \sum_{i=1}^M \rho_i \lceil -\log_2^{\rho_i} \rceil < \sum_{i=1}^M \rho_i (1 - \log_2^{\rho_i}) = H(X) + 1$$

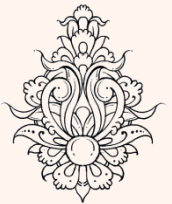


# سایر شیوه‌های کدگذاری

- کدگذاری Huffman منجر به متوسط نرخ بیت کمینه می‌شود، اما در مورد حداکثر طول کلمه‌کد هیچ محدودیتی ندارد.

– در برخی کاربردها به دلیل جلوگیری از پیچیدگی سخت‌افزار و یا محدودیت طول ثبات‌ها نوعی از کدگذاری مطرح می‌شود که هرچند بهینه نیست، اما طول کلمه‌کدها دارای محدودیت است.

## Length- Limited Huffman Codes



# سایر شیوه‌های کدگذاری (ادامه...)

- در شرایطی که الفبای کوچکی داشته باشیم و احتمال وقوع نمادها متوازن نباشد، کد Huffman از کارایی کمتری برخوردار خواهد بود:

$$A = \{a_1, a_2, a_3\}$$

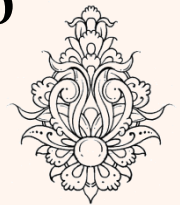
$$p(a_1) = 0.8 \quad p(a_2) = 0.02 \quad p(a_3) = 0.18$$

$$Ent = 0.816$$

Letter	Codeword
$a_1$	0
$a_2$	11
$a_3$	10

$$l_{av} = 1.2 \text{ bits/symbol}$$

نرخ بیت ۴۷٪ بیش از آنتروپی است





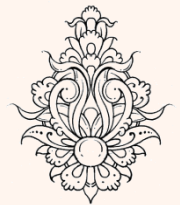
# سایر شیوه‌های کدگذاری (ادامه...)

- در چنین شرایطی پیشنهاد می‌شود به جای کد کردن نمادها، مجموعه‌ای از نمادها را کد کنیم.

$$\mathcal{A}^{(n)} = \{\overbrace{a_1 a_1 \dots a_1}^{n \text{ times}}, a_1 a_1 \dots a_2, \dots, a_1 a_1 \dots a_m, a_1 a_1 \dots a_2 a_1, \dots, a_m a_m \dots a_m\}$$

Letter	Probability	Code
$a_1 a_1$	0.64	0
$a_1 a_2$	0.016	10101
$a_1 a_3$	0.144	11
$a_2 a_1$	0.016	101000
$a_2 a_2$	0.0004	10100101
$a_2 a_3$	0.0036	1010011
$a_3 a_1$	0.1440	100
$a_3 a_2$	0.0036	10100100
$a_3 a_3$	0.0324	1011

$$H(X) \leq R < H(X) + \frac{1}{n}$$



افزایش اندازه‌ی الفبا می‌تواند منجر به ناکارایی کدگذاری شود.

- میزان اطلاعات یک دنباله به طول  $n$  (با الفبایی شامل  $m$  نماد) با فرض مستقل بودن و یکسان بودن توزیع نمادها

$$H(\mathbf{x}^n) = nH(x)$$

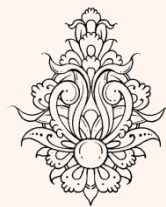
- اثبات:

$$H(\mathbf{x}^n) = -\sum_{i_1=1}^m \sum_{i_2=1}^m \cdots \sum_{i_n=1}^m P(a_{i_1}, a_{i_2}, \dots, a_{i_n}) \log [P(a_{i_1}, a_{i_2}, \dots, a_{i_n})]$$

- با در نظر گرفتن iid بودن منبع:

$$H(\mathbf{x}^n) = -\sum_{i_1=1}^m \sum_{i_2=1}^m \cdots \sum_{i_n=1}^m P(a_{i_1})P(a_{i_2}) \cdots P(a_{i_n}) \log [P(a_{i_1}, a_{i_2}, \dots, a_{i_n})]$$

$$H(\mathbf{x}^n) = -\sum_{i_1=1}^m \sum_{i_2=1}^m \cdots \sum_{i_n=1}^m P(a_{i_1})P(a_{i_2}) \cdots P(a_{i_n}) \sum_{j=1}^n \log P(a_{i_j})$$



$$H(\mathbf{x}^n) = - \sum_{i_1=1}^m P(a_{i_1}) \log P(a_{i_1})$$

$$\left\{ \sum_{i_2=1}^m \cdots \sum_{i_n=1}^m P(a_{i_2}) \cdots P(a_{i_n}) \right\}$$

$$- \sum_{i_2=1}^m P(a_{i_2}) \log P(a_{i_2})$$

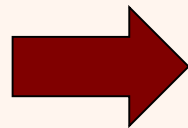
$$\left\{ \sum_{i_1=1}^m \sum_{i_3=1}^m \cdots \sum_{i_n=1}^m P(a_{i_1}) P(a_{i_2}) \cdots P(a_{i_n}) \right\}$$

⋮

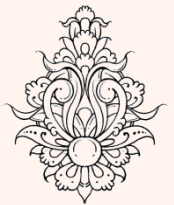
$$- \sum_{i_n=1}^m P(a_{i_n}) \log P(a_{i_n})$$

$$\left\{ \sum_{i_1=1}^m \sum_{i_3=1}^m \cdots \sum_{i_{n-1}=1}^m P(a_{i_1}) P(a_{i_2}) \cdots P(a_{i_{n-1}}) \right\}$$

برابر با یکی

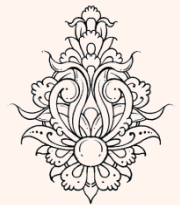


$$H(\mathbf{x}^n) = nH(x)$$



# پیاده‌سازی کد Huffman

- کدگذاری به سادگی و با کمک یک «جدول مراجعه» انجام می‌شود.
- کدگذاری، با چنین شیوه‌ای به یک جدول بزرگ  
– متناسب با بزرگ‌ترین کلمه‌کد  
می‌توان از ساختار درخت دودیی استفاده کرد.  
– پیمایش درخت زمان‌بر است
- معمولاً از ترکیب این دو استفاده می‌شود.

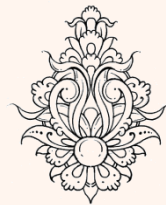


# سایر شیوه‌های کدگذاری (ادامه...)

- در صورتی که مشخصات آماری نمادها در طول زمان تغییر کند، استفاده از کدگذاری وفقی Huffman پیشنهاد می‌شود.

## Adaptive Huffman Coding

- به علت حجم محاسبات بالا که در مورد استفاده قرار می‌گیرد.



# Unary coding

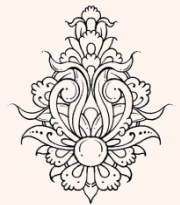
• این شیوهی کدگذاری برای اعداد طبیعی پیشنهاد شده است:

– برای کد کردن عدد  $n$ ، به تعداد  $n$  عدد یک و در پایان صفر قرار می‌دهیم.

– در صورتی که  $p(n) = 1/2^n$  این شیوه بهینه است. (معادل Huffman)

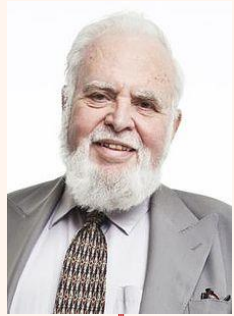
– با وجود این که تابع احتمال فوق در مورد محدودی بهینه است، اما پیاده‌سازی آن بسیار ساده است.

n	Unary code
0	0
1	10
2	110
3	1110
4	11110
5	111110



# کدگذاری Golomb

با پارامتر  $m$



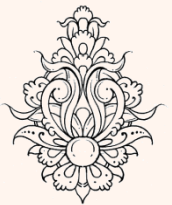
- ابتدا خارج قسمت و باقیمانده نسبت به  $m$  محاسبه می‌شود:

$$q = \left\lfloor \frac{n}{m} \right\rfloor \quad r = n - qm$$

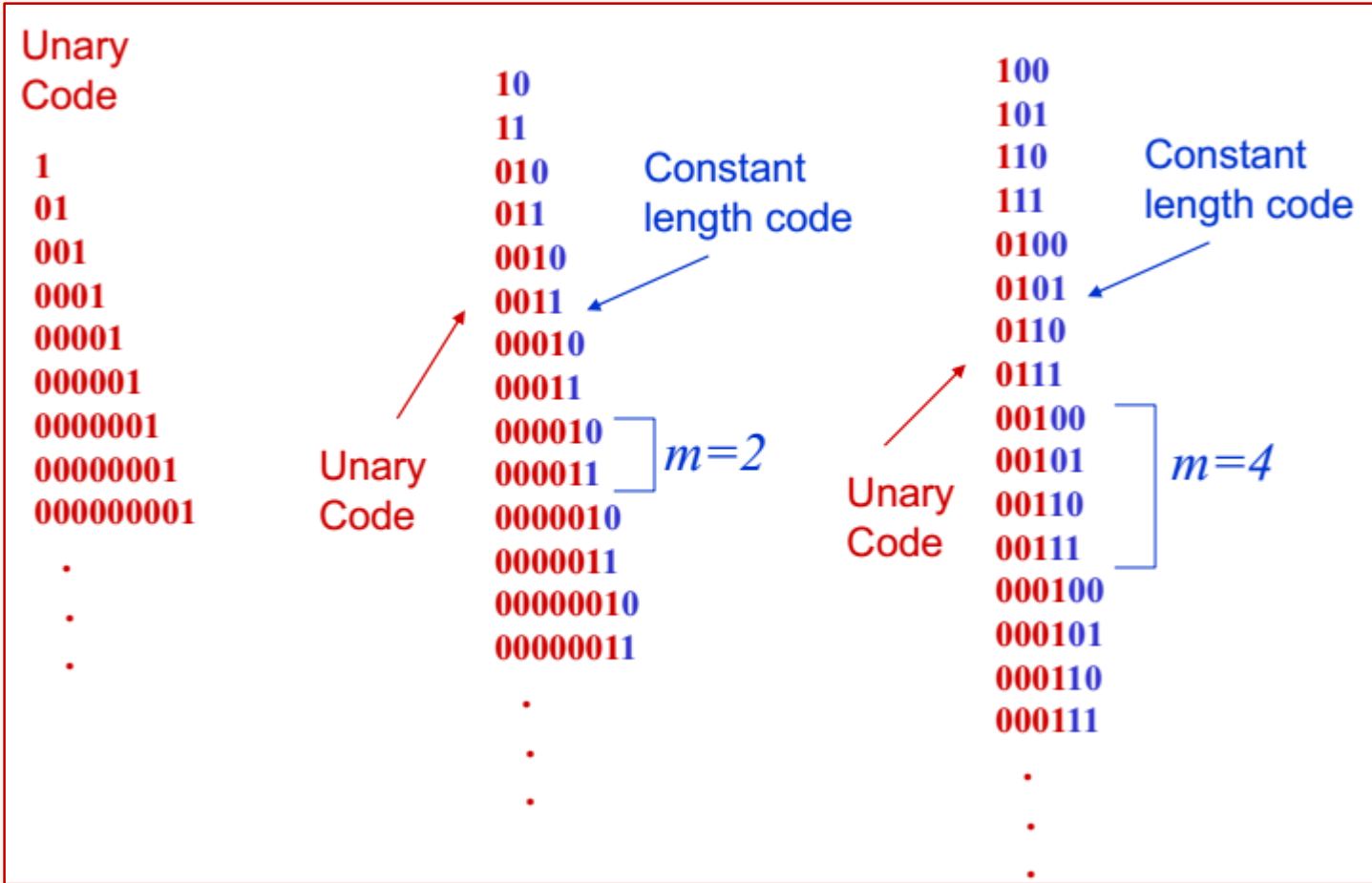
- $q$  (خارج قسمت)، با استفاده از کدهای unary کد می‌شوند.

- باقیمانده‌ها در بازه‌ی 0 تا  $m-1$  قرار دارد.

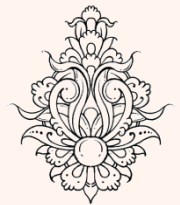
- در صورتی که  $m$  توانی از ۲ باشد، می‌توان از کدهای باینری استفاده کرد.



# مثال



Bernd Girod: EE398A Image and Video Compression



به صورت مشابه می‌توان کدها را به صورت معکوس هم تعریف کرد.



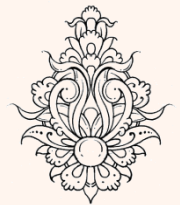
## Truncated binary encoding

در غیر این صورت از Truncated binary encoding استفاده می‌شود.

در این حالت هدف کدگذاری  $m$  نماد است. چنانچه  $m$  توانی از  $۲$  می‌بود و از کدگذاری دودویی معمولی استفاده می‌شد، به  $\log_2 m$  بیت نیاز می‌بود. در این حالت که  $2^n < m < 2^{n+1}$  برای  $2^{n+1} - m$  نماد اول از  $n$  بیت و برای مابقی نمادها از  $n+1$  بیت استفاده شود.

truncated binary encoding for numbers from 0 to 5

0	00	4	110
1	01	5	111
2	100		
3	101		

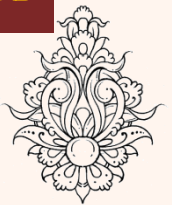


## Truncated binary encoding

- این شیوه برای نمادهایی که توزیع یکنواخت دارند، مناسب است.
- نمادهای بزرگ‌تر (از نظر الفبایی) با رشته‌ی بزرگ‌تر کد می‌شود.
- هیچ کد  $n$  بیتی پیشوند کد  $n+1$  بیتی نیست.

تمرین: برای  $m=5$  کلمه‌کدها را به دست آورید.

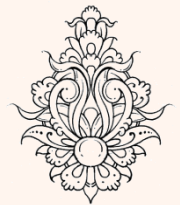
0	00
1	01
2	10
3	110
4	111



# مثال-کدگذاری Golomb

$m=5$

$n$	$q$	$r$	Codeword	$n$	$q$	$r$	Codeword
0	0	0	000	08	1	3	10110
1	0	1	001	09	1	4	10111
2	0	2	010	10	2	0	11000
3	0	3	0110	11	2	1	11001
4	0	4	0111	12	2	2	11010
5	1	0	1000	13	2	3	110110
6	1	1	1001	14	2	4	110111
7	1	2	1010	15	3	0	111000



# الفبای نامتوازن

- در مواردی که با الفبایی سروکار داریم که در آن حروف دارای احتمال وقوع نامتوازن هستند:

$\alpha_i$	Prob.	Code word
$\alpha_1$	0.95	0
$\alpha_2$	0.02	10
$\alpha_3$	0.03	11

$$Ent = 0.335$$

$$\text{Average Bit Rate} = 0.95 \times 1 + 0.05 \times 2 = 1.05 \text{ bits/symbol}$$

نرخ بیت بیش از دو برابر بیشتر از مدی که آنتروپی تعیین کرده



# کدگذاری دنباله‌ای از نمادها

Letter	Probability	Code
$a_1a_1$	0.9025	0
$a_1a_2$	0.0190	111
$a_1a_3$	0.0285	100
$a_2a_1$	0.0190	1101
$a_2a_2$	0.0004	110011
$a_2a_3$	0.0006	110001
$a_3a_1$	0.0285	101
$a_3a_2$	0.0006	110010
$a_3a_3$	0.0009	110000

Average Bit Rate=0.611 bits/symbols

با ادامه دادن همین روند (بلوک‌های هشت‌تایی) به حداقل نرخ بیت نزدیک خواهیم شد. در این صورت جدولی بزرگ خواهیم داشت (۶۵۶۱).



- به حجم حافظه‌ی بالایی نیاز است که در بسیاری از کاربردها عملی نیست.
- کدگذاری هزینه‌ی مناسباتی بالایی دارد.
- در صورتی که احتمال رخداد نمادها تغییر کند، اثر آن بر روی کارایی بالا خواهد بود.

# کدگذاری دنباله‌ای از نمادها (ادامه...)

تولید یک کلمه‌کد برای گروهی از نمادها منجر به افزایش کارایی می‌شود.

پیاده‌سازی Huffman در چنین حالتی دشوار است. چرا که باید برای همه‌ی ترکیب‌های نمادها معادل کد آن را در اختیار داشت. رشد تعداد کلمه‌کدها با گروه نمادها نمایی خواهد بود.

کدهای مناسب‌تری راهی است که بتوان مجموعه‌ای از نمادها برچسب زد بدون این که نیاز باشد برای همه‌ی ترکیب‌های ممکن برچسب تهیه کرد.



- برای اولین بار در مقاله‌ی معروف Shannon استفاده از تابع توزیع تجمعی مطرح شد.
- اولین بار توسط Elias مطرح شد، هرچند این مطلب را منتشر نکرد.
- در کتاب Abramson این مسأله آورده شد.
- به شیوه‌ی کنونی به صورت جداگانه توسط Pasco و Rissanen کشف شد.
- در این شیوه به جای اختصاص کلمه‌کد برای هر نماد، به هر دنباله از نمادها یک **برچسب** نسبت داده می‌شود.

۱۹۴۸

۱۹۶۳

۱۹۷۴



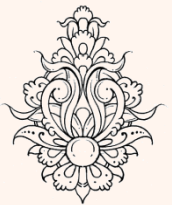
# فرآیند کدگذاری

- به هر دنباله‌ای از نمادها باید یک «برچسب یکتا» نسبت داده شود.
- این برچسب یک عدد در بازه‌ی  $(0,1)$  است.
- برای محاسبه‌ی این برچسب از تابع توزیع تجمعی (cdf) استفاده می‌شود.
- در این روش کل نمادهای موجود در نظر گرفته می‌شود و بر حسب احتمال وقوع به بازه‌ی  $[0, 1]$  تقسیم می‌شود.

$$A = \{a_0, a_1, \dots, a_{M-1}\}$$

$$P(X = i) = P(a_i) = \rho_i \quad \text{probability density function}$$

$$F_X(i) = \sum_{k=1}^i \rho_k \quad \text{cumulative density function}$$





# فرآیند کدگذاری (ادامه...)

*arithmetic coding is a practical way of implementing entropy coding*

- فرض کنید  $M$  نماد با احتمال وقوع  $\rho_i$  برای هر یک وجود دارد.

- در این صورت بازه  $[0, 1]$  را می‌توان به صورت زیر تقسیم نمود:

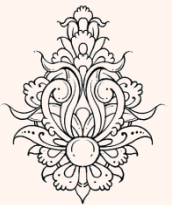
$$[0, \rho_1, \rho_1 + \rho_2, \dots, \sum_{i=1}^{M-1} \rho_i, 1]$$

- در این صورت نماد  $k$ -ام در محدوده زیر قرار دارد:

$$\left( \sum_{i=1}^{k-1} \rho_i, \sum_{i=1}^k \rho_i \right)$$



$$(F_X(i-1), F_X(i))$$



# مثال ۱

- فرض کنید دنباله‌ای از نمادها به صورت زیر در دست باشد و بخواهیم از این روش کد گردد:

$aab$

امتثال وقوع

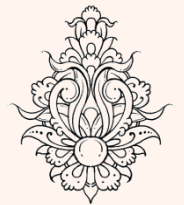
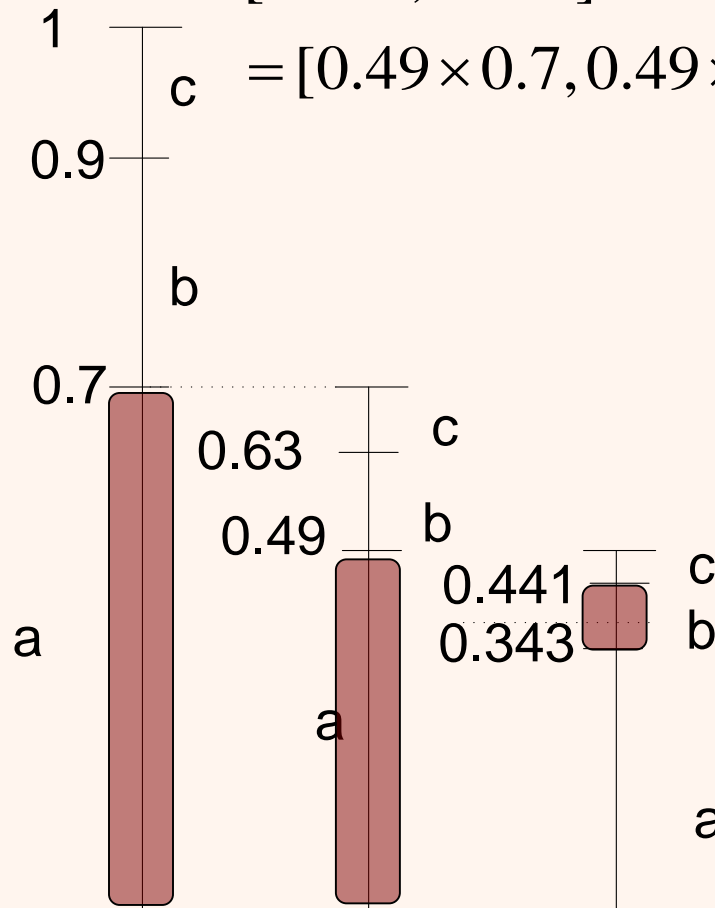
$a \rightarrow 0.7$

$b \rightarrow 0.2$

$c \rightarrow 0.1$

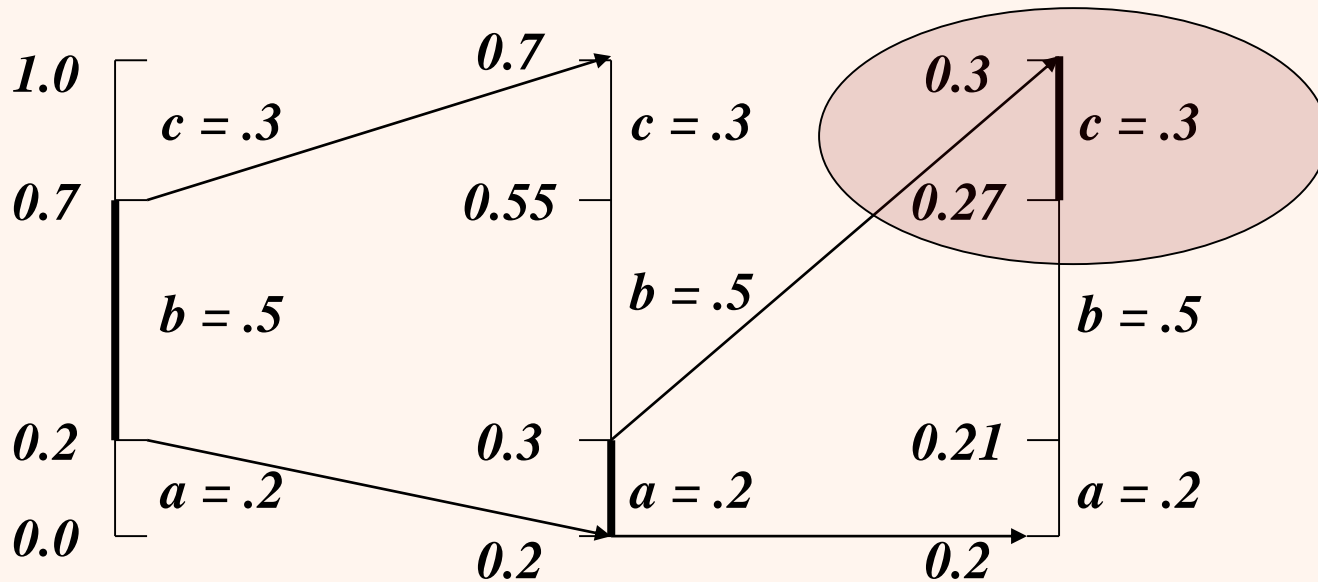
$[0.343, 0.441]$

$$c = [0.49 \times 0.7, 0.49 \times 0.7 + 0.49 \times 0.2]$$



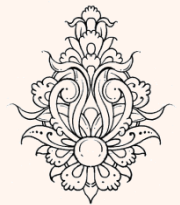
# مثال ۲

کدگذاری رشته: bac



$[0.27, 0.3)$

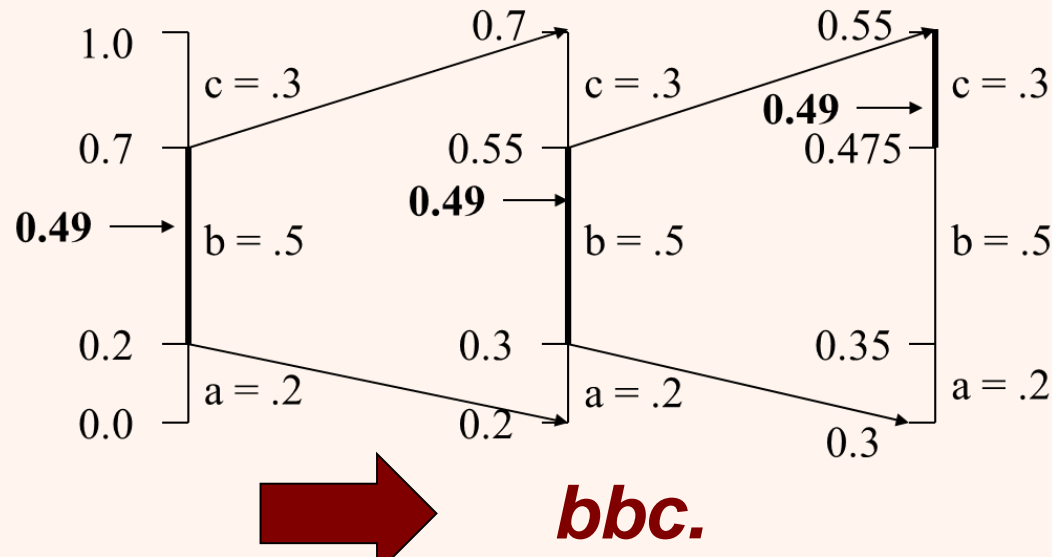
محدوده‌ی نهایی



محدوده‌ی به دست آمده برای هر جمله یکتا، به صورت *non overlapped* خواهد بود برای بازگشایی کد نیز از این خاصیت استفاده می‌شود.

# کدگشایی-مثال

کدگشایی عدد ۰.۴۹ برای جریان داده با طول ۳



1. Initialize  $l^{(0)} = 0$  and  $u^{(0)} = 1$ .
2. For each  $k$  find  $t^* = (tag - l^{(k-1)}) / (u^{(k-1)} - l^{(k-1)})$ .
3. Find the value of  $x_k$  for which  $F_X(x_k - 1) \leq t^* < F_X(x_k)$ .
4. Update  $u^{(k)}$  and  $l^{(k)}$ .
5. Continue until the entire sequence has been decoded.

# فرآیند کدگذاری (ادامه...)

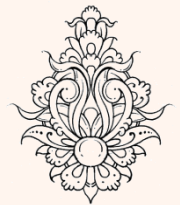
- هر نقطه‌ای از این بازه را می‌توان به عنوان برچسب یکتا در نظر گرفت.
- نقطه‌ی میانی این بازه می‌تواند انتخاب مناسبی باشد.
- برای یک نماد، این عدد را می‌توان به صورت زیر در نظر گرفت:

$$\begin{aligned}\bar{T}_X(a_i) &= \sum_{k=1}^{i-1} P(X = k) + \frac{1}{2}P(X = i) \\ &= F_X(i - 1) + \frac{1}{2}P(X = i)\end{aligned}$$

- و برای دنباله‌ای به طول  $m$

$$\bar{T}_X^{(m)}(\mathbf{x}_i) = \sum_{\mathbf{y} < \mathbf{x}_i} P(\mathbf{y}) + \frac{1}{2}P(\mathbf{x}_i)$$

داده‌ها به صورت الفبایی مرتب شده‌اند



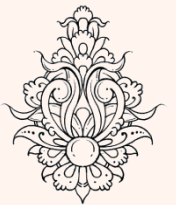
# مثال

- در صورتی که الفبا شامل شش نشانه با احتمال یکسان باشد، خواهیم داشت:

$$\bar{T}_X(2) = P(X = 1) + \frac{1}{2}P(X = 2) = \frac{1}{6} + \frac{1}{12} = 0.25$$

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

$$\begin{aligned}\bar{T}_X(13) &= P(x = 11) + P(x = 12) + 1/2P(x = 13) \\ &= 1/36 + 1/36 + 1/2(1/36) \\ &= 5/72\end{aligned}$$



# محاسبه‌ی برچسب

- برای هر دنباله باید کران پایین و بالای بازه را مشخص کرد.

– برای نماد اول

$$(l^{(1)}, u^{(1)}) = (F_X(x_1 - 1), F_X(x_1))$$

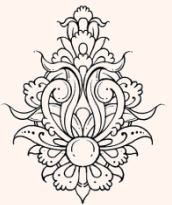
و برای سایر نمادها:

$$l^{(n)} = l^{(n-1)} + (u^{(n-1)} - l^{(n-1)})F_X(x_n - 1)$$
$$u^{(n)} = l^{(n-1)} + (u^{(n-1)} - l^{(n-1)})F_X(x_n)$$

و در نهایت نقطه‌ی میانی:

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

برای کدگذاری تنها تابع توزیع تجمعی مورد نیاز است.



# مثال

• الفبای زیر را در نظر بگیرید:

$$A = \{a_1, a_2, a_3\}$$

$$P(a_1) = 0.8 \quad P(a_2) = 0.02 \quad P(a_3) = 0.18$$

هدف کد کردن دنباله‌ی 1 3 2 1

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

$$l^{(1)} = 0 + (1 - 0)0 = 0 \quad l^{(2)} = 0 + (0.8 - 0)F_X(2) = 0.8 \times 0.82 = 0.656$$

$$u^{(1)} = 0 + (1 - 0)(0.8) = 0.8 \quad u^{(2)} = 0 + (0.8 - 0)F_X(3) = 0.8 \times 1.0 = 0.8$$

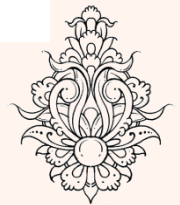
$$l^{(3)} = 0.656 + (0.8 - 0.656)F_X(1) = 0.656 + 0.144 \times 0.8 = 0.7712$$

$$u^{(3)} = 0.656 + (0.8 - 0.656)F_X(2) = 0.656 + 0.144 \times 0.82 = 0.77408$$

$$l^{(4)} = 0.7712 + (0.77408 - 0.7712)F_X(0) = 0.7712 + 0.00288 \times 0.0 = 0.7712$$

$$u^{(4)} = 0.7712 + (0.77408 - 0.1152)F_X(1) = 0.7712 + 0.00288 \times 0.8 = 0.773504$$

$$\bar{T}_X(1321) = \frac{0.7712 + 0.773504}{2} = 0.772352$$



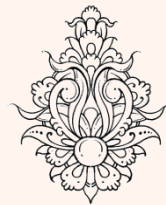


# تولید کد دودویی

- هدف یافتن یک کد دودویی یکتا به روشی کاراست.
- در برخی موارد تعداد ارقام برچسب به دست آمده، بی‌شمار است.
- چنان چه  $p(x)$  احتمال رخداد یک دنباله از نمادها باشد. طول معادل دودویی آن از رابطه‌ی زیر به دست می‌آید:

$$l(x) = \left\lceil \log_2 \left( \frac{1}{p(x)} \right) \right\rceil + 1$$

- می‌توان نشان داد پس از این نوع نمایش کد یکتا و قابل کدگشایی است.



# مثال

$$\mathcal{A} = \{a_1, a_2, a_3, a_4\}$$

$$P(a_1) = \frac{1}{2}, \quad P(a_2) = \frac{1}{4}, \quad P(a_3) = \frac{1}{8}, \quad P(a_4) = \frac{1}{8}$$

Symbol	$F_X$	$\bar{T}_X$	In Binary	$\lceil \log \frac{1}{P(x)} \rceil + 1$	Code
1	.500	.2500	.0100	2	01
2	.750	.6250	.1010	3	101
3	.875	.8125	.1101	4	1101
4	1.000	.9375	.1111	4	1111



# یکتایی کد محاسباتی

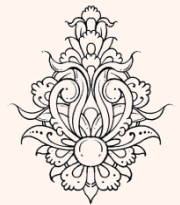
- برچسب به دست آمده با طول  $l(x)$  بریده شود.

$$T_X(x) \rightarrow \lfloor T_X(x) \rfloor_{l(x)}$$

- در این صورت  $\lfloor T_X(x) \rfloor_{l(x)}$  نیز یکتاست.
- باید ثابت کنیم:

$$F_X(\mathbf{x}-1) < \lfloor T_X(x) \rfloor_{l(x)} < F_X(\mathbf{x})$$

- درستی بخش راست این نامساوی واضح است!



# یکتایی کد محاسباتی

$$F_X(\mathbf{x}-1) < \lfloor T_X(x) \rfloor_{l(x)}$$

$$0 \leq T_X(x) - \lfloor T_X(x) \rfloor_{l(x)} < \frac{1}{2^{l(\mathbf{x})}}$$

$$\lfloor T_X(x) \rfloor_{l(x)} > T_X(x) - \frac{1}{2^{l(\mathbf{x})}}$$

$$\frac{1}{2^{l(\mathbf{x})}} = \frac{1}{2^{\lceil \log_2 \left( \frac{1}{p(\mathbf{x})} \right) \rceil + 1}}$$

$$< \frac{1}{2^{\log_2 \left( \frac{1}{p(\mathbf{x})} \right) + 1}} = \frac{P(\mathbf{x})}{2}$$

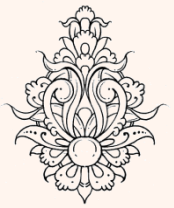
$$T_X(x) - F_X(\mathbf{x}-1) > \frac{1}{2^{l(\mathbf{x})}}$$

$$\frac{P(\mathbf{x})}{2} = T_X(x) - F_X(\mathbf{x}-1)$$

$$T_X(x) = F_X(\mathbf{x}-1) + \frac{P(\mathbf{x})}{2}$$

$$F_X(\mathbf{x}-1) < T_X(x) - \frac{1}{2^{l(\mathbf{x})}}$$

$$F_X(\mathbf{x}-1) < \lfloor T_X(x) \rfloor_{l(x)}$$

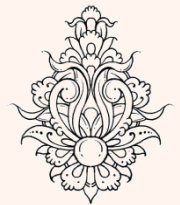


# کد پیشوندی

• ثابت می‌شود که کد به دست آمده پیشوندی است:

– در صورتی که  $a$  یک عدد در بازه‌ی  $[0,1)$  با نمایش دودویی  $[b_1b_2\dots b_n]$  باشد، اگر عدد دیگری مانند  $b$  دارای پیشوند یکسان  $[b_1b_2\dots b_n]$  با  $a$  باشد خواهیم داشت:

$$b \in [a, a + \frac{1}{2^n})$$

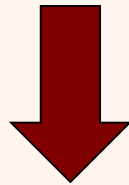


# کد پیشوندی

می‌فهمیم ثابت کنیم که تمام کدهایی که پیشوند آن‌ها کد به دست آمده‌است در همین بازه قرار می‌گیرند.

$$F_X(\mathbf{x}-1) < \lfloor T_X(x) \rfloor_{l(x)} < F_X(\mathbf{x}) \quad \bullet \text{ می‌دانیم}$$

$$F_X(\mathbf{x}) - \lfloor T_X(x) \rfloor_{l(x)} > F_X(\mathbf{x}) - T_X(x) = \frac{P(x)}{2} > \frac{1}{2^{l(x)}}$$



$$\frac{P(\mathbf{x})}{2} = T_X(x) - F_X(\mathbf{x}-1)$$

$$F_X(\mathbf{x}) - \lfloor T_X(x) \rfloor_{l(x)} > \frac{1}{2^{l(x)}}$$

در نتیجه

$$\left[ \lfloor T_X(x) \rfloor_{l(x)}, \lfloor T_X(x) \rfloor_{l(x)} + \frac{1}{2^{l(x)}} \right) \in [F_X(\mathbf{x}-1), F_X(\mathbf{x}))$$

کد به دست  
آمده پیشوندی  
است

# کارایی کد محاسباتی

طول متوسط دنباله‌های کد شده به طول  $m$

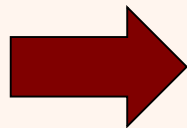
- پیش از به کارگیری کدهای محاسباتی باید میزان سودمندی آن‌ها را سنجید:

$$l_{A^{(m)}} = \sum P(\mathbf{x}) l(\mathbf{x})$$
$$= \sum P(\mathbf{x}) \left[ \log \frac{1}{P(\mathbf{x})} \right] + 1$$

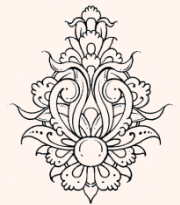
$$l(\mathbf{x}) = \left\lceil \log \frac{1}{P(\mathbf{x})} \right\rceil + 1$$

$$< \sum P(\mathbf{x}) \left[ \log \frac{1}{P(\mathbf{x})} + 1 + 1 \right]$$

$$= \sum P(\mathbf{x}) \log \frac{1}{P(\mathbf{x})} + 2 \sum P(\mathbf{x})$$



$$l_{A^{(m)}} < H(\mathbf{x}^m) + 2$$



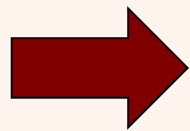
# کارایی کد محاسباتی (ادامه...)

- با توجه به این که طول متوسط همواره از آنتروپی بزرگتر است، خواهیم داشت:

$$H(\mathbf{x}^m) < l_{A^{(m)}} < H(\mathbf{x}^m) + 2$$

- میزان متوسط نرخ بیت به ازای هر نماد:

$$l_A = \frac{l_{A^{(m)}}}{m}$$



$$\frac{H(\mathbf{x}^m)}{m} < l_A < \frac{H(\mathbf{x}^m) + 2}{m}$$

$$H(\mathbf{x}^m) = mH(x)$$

- در نتیجه



$$H(x) < l_A < H(x) + \frac{2}{m}$$

